Software Technology Roadmap

What was the STR?

The Software Technology Roadmap (STR) was a directed guide containing the information on more than 69 software technologies. It was of interest to anyone acquiring, building, or maintaining software intensive systems.

When was the STR last updated?

The STR was last updated in 2002. The Software Engineering Institute no longer maintains or coordinates changes to the STR.

URL: http://www.sei.cmu.edu/str/index.html
Last Modified: 24 July 2008

Software Technology Roadmap

Software Technology Roadmap

What was the STR?

The Software Technology Roadmap (STR) was a directed guide containing the information on more than 69 software technologies. It was of interest to anyone acquiring, building, or maintaining software intensive systems.

When was the STR last updated?

The STR was last updated in 2002. The Software Engineering Institute no longer maintains or coordinates changes to the STR.

URL: http://www.sei.cmu.edu/str/index.html
Last Modified: 24 July 2008

About The STR

Software Technology Roadmap

- Background
- Target Audiences
- Sponsors & Contributors

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/about/index.html
Last Modified: 24 July 2008

STR Technology Descriptions

Software Technology Roadmap

About the Technology Descriptions

- Defining Software Technology
- Technology Categories
- Template for Technology Descriptions

List of Technology Descriptions

An alphabetical list of approximately 69 software technologies is below. Browse to find the topic that interests you, or search on key words or phrases to see a list of relevant technologies.

search Help

- Ada 95
- Algorithm Formalization
- Application Programming Interface
- Architecture Description Languages
- Argument-Based Design Rationale Capture Methods for Requirements Tracing
- Maintenance of Operational Systems--An Overview
- Message-Oriented Middleware
- Middleware
- Model-Based Verification
- Module Interconnection Languages
- Multi-Level Secure Database Management Schemes

- Black-box Modernization of Information Systems
- Capability Maturity Model Integration (CMMI)
- Cleanroom Software Engineering
- Client/Server Software Architectures--An Overview
- Common Management Information Protocol
- Common Object Request Broker Architecture
- Component-Based Software Development / COTS Integration
- Component Object Model (COM), DCOM, and Related Capabilities
- Computer System Security--An Overview
- COTS and Open Systems--An Overview
- Cyclomatic Complexity
- Database Two Phase Commit
- Defense Information Infrastructure Common Operating Environment (DII COE)
- Multi-Level Secure One Way Guard with Random Acknowledgment
- Network Management--An Overview
- Nonrepudiation in Network Communications
- Object-Oriented Analysis
- Object-Oriented Database
- Object-Oriented Design
- Object-Oriented Programming Languages
- Object Request Broker
- Organization Domain Modeling
- People Capability Maturity Model (P-CMM)
- Personal Software Process for Module-Level Development
- Public Key Cryptography
- Public Key Digital Signatures
- Rate Monotonic Analysis
- Reference Models, Architectures, Implementations--An Overview
- Remote Procedure Call
- Requirements Tracing--An Overview

- Digital Certificates
- Distributed/Collaborative Enterprise Architectures
- Distributed Computing Environment
- Domain Engineering and Domain Analysis
- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Feature-Oriented Domain Analysis
- Firewalls and Proxies
- Function Point Analysis
- Graphic Tools for Legacy Database Migration
- Graphical User Interface Builders
- Halstead Complexity Measures
- Intrusion Detection
- Java
- Mainframe Server Software Architectures
- Rule-Based Intrusion Detection
- Simple Network Management Protocol
- Six Sigma
- Simplex Architecture
- Software Inspections
- Statistical-Based Intrusion Detection
- Statistical Process Control for Software
- TAFIM Reference Model
- Team Software Process
- Three Tier Software Architectures
- Transaction Processing Monitor Technology
- Trusted Operating Systems
- Two Tier Software Architectures
- Virus Detection

- Maintainability Index Technique for Measuring Program Maintainability

---

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/index.html
Last Modified: 24 July 2008

Taxonomies

Software Technology Roadmap

- About the Taxonomies
- View the Application Taxonomy
- View the Quality Measures Taxonomy
- Search the Taxonomies

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/taxonomies/index.html
Last Modified: 24 July 2008

Glossary & Indexes

Software Technology Roadmap

- Glossary
- Keyword Index

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/indexes/index.html
Last Modified: 24 July 2008

- Software Technology Roadmap
- Background & Overview
  - Background
  - Target Audiences
  - Sponsors & Contributors
- Technology Descriptions
- Taxonomies
- Glossary & Indexes

Background

Software Technology Roadmap

The Air Force acquisition community tasked the Software Engineering Institute (SEI) to create a reference document that would provide the Air Force with a better understanding of software technologies. This knowledge will allow the Air Force to systematically plan the research and development (R&D) and technology insertion required to meet current and future Air Force needs, from the upgrade and evolution of current systems to the development of new systems.

Scope

The initial release of the Software Technology Roadmap is a prototype to provide initial capability, show the feasibility, and examine the usability of such a document. This prototype generally emphasizes software technology[1] of importance to the C4I (command, control, communications, computers, and intelligence) domain. This emphasis on C4I neither narrowed nor broadened the scope of the document; it did, however, provide guidance in seeking out requirements and technologies. It served as a reminder that this work is concerned with complex, large-scale, distributed, real-time, software-intensive, embedded systems in which reliability, availability, safety, security, performance, maintainability, and cost are major concerns.

We note, however, that these characteristics are not only applicable to military command and control systems, they apply as well to commercial systems, such as financial systems for electronic commerce. Also, for a variety of reasons, commercial software will play an increasingly important role in defense systems. Thus, it is important to understand trends and opportunities in software technology -- including commercial software practice and commercially-available software components -- that may affect C4I systems.

Vision

Our long-term goal is to create a continuously-updated, community "owned," widely-available reference document that will be used as a shared knowledge base. This shared knowledge base will assist in the tradeoff and selection of appropriate technologies to meet system goals, plan technology insertions, and possibly establish research agendas. While we use the term "document," we anticipate that this product will take many shapes, including a Web-based, paper-based, or CD-ROM based reference.

With the release of this document we are seeking comment and feedback from

the software community. We will use this feedback as we plan an ongoing effort to expand and evolve this document to include additional software technology descriptions. The Feedback Section provides vehicles by which readers can contribute to the further development of this effort.

Goal

The document is intended to be a guide to specific software technologies of interest to those building or maintaining systems, especially those in command, control, and/or communications applications. The document has many goals:

- to provide common ground by which contractors, commercial companies, researchers, government program offices, and software maintenance organizations may assess technologies
- to serve as *Cliff's Notes* for specific software technologies; to encapsulate a large amount of information so that the reader can rapidly read the basics and make a preliminary decision on whether further research is warranted
- to achieve objectivity, balance,[2] and a quantitative focus, bringing out both shortcomings as well as advantages, and provide insight into areas such as costs, risks, quality, ease of use, security, and alternatives
- to layer information so that readers can find subordinate technology descriptions (where they exist) to learn more about the topic(s) of specific interest, and to provide references to sources of more detailed technical information, to include usage and experience

Limitations/Caveats

While the document provides balanced coverage of a wide scope of technologies, there are certain constraints on the content of the document:

- *Coverage, accuracy and evolution.* Given the number of software technologies and the time available for this first release, this document covers a relatively small set of technologies. As such, there are many topics that have not been addressed; we plan to address these in subsequent versions. This document is, by nature, a snapshot that is based on what is known at the time of release. We have diligently worked to make the document as accurate as possible. Each technology description is rated as to its completeness. Subsequent versions will include corrections and updates based on community feedback.
- *Not prescriptive.* This document is not prescriptive; it does not make recommendations, establish priorities, or dictate a specific path/approach.[3] The reader must make decisions about whether a technology is appropriate for a specific engineering and programmatic context depending on the planned intended use, its maturity, other technologies that will be used, the specific time frame envisioned, and funding

constraints.

For example, a specific technology may not be applicable to a particular program because the need is current and evaluations indicate that the technology is immature under certain circumstances. However, given a program that initiates in 3-5 years, the same technology may be an appropriate choice assuming that the areas of immaturity will be corrected by then (and, if necessary, directed action to ensure the maturation or to remedy deficiencies).

- *Not a product reference.* This document is not a survey or catalog of products. There are many reasons for this, including the rapid proliferation of products, the need to continually assess product capabilities, questions of perceived endorsement, and the fact that products are almost always a collection of technologies. It is up to the reader to decide which products are appropriate for their context. DataPro and Auerbach would likely be better sources of product-specific information.

- *Not an endorsement.* Inclusion or exclusion of a topic in this document does not constitute an endorsement of any type, or selection as any sort of "best technical practice." Judgments such as these must be made by the readers based on their contexts; our goal is to provide the balanced information to enable those judgments.

- *Not a focused analysis of specific technical areas.* Various sources such as Ovum, Ltd. and The Standish Group offer reports on a subscription or one-time basis on topics such as workflow, open systems, and software project failure analyses, and may also produce specialized analyses and reporting on a consulting basis.

Footnotes

[1] This spectrum of technologies includes past, present, under-used, and emerging technologies.

[2] As an example of balanced coverage, let's briefly look at information hiding of object-oriented inheritance, which reduces the amount of information a software developer must understand. Substantial evidence exists that such object-oriented technologies significantly increase productivity in the early stages of software development; however, there is also growing recognition that these same technologies may also encourage larger and less efficient implementations, extend development schedules beyond the "90% complete" point, undermine maintainability, and preclude error free implementations.

[3] Similar to a roadmap for highways, the review prescribes neither the destination nor the most appropriate route. Instead, it identifies a variety of alternative routes that are available, gives an indication of their condition, and describes where they may lead. Specific DoD applications must chart their own route through the technological advances.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/about/background_body.html
Last Modified: 24 July 2008

Target Audiences

Software Technology Roadmap

We envisioned that this document would be relevant to many audiences. The audiences and a description of how each audience can use this document are shown in the table below.

User	Job Roles/Tasks	Document Capabilities/ Value
PEO/Executive Pentagon Action Officer	Acquisition oversight, funding advocacy Motivate introduction of new/commercial technologies Policy issues	Overview/introductory info Baseline reference document *Cliff's Notes* approach -- provides high-level, 4-6 page quick study Tradeoff information
System Program Manager (SPM) and Technical Staff (Includes FFRDCs (MITRE, etc.) and may include government laboratories)	Writes Request for Proposal (RFP) or some form of solicitation based on user requirements Reviews proposals and selects developers Manages development and/ or maintenance work	All of previous category, plus: Taxonomies to aid in identifying alternatives Back pointers to high-level, related technologies Criteria and guidance for decision-making Tech transfer/insertion guidelines Selected high-value references to more technical information, to include usage and

		experience data Generally the sort of analysis and survey information that would not be accomplished under normal project circumstances
Developer (to include research and development (R&D) activity)	Performs advanced development, prototyping, and technology investigation focused on risk reduction and securing competitive advantage Concerned about transition and insertion issues Writes a proposal in response to solicitations Performs engineering development and provides initial operational system	Same as previous category.
Maintainer	Maintains operational system until the end of the life cycle Responds to user requirements for corrections or enhancements Concerned about inserting new technologies and migrating to different approaches	Same as previous category.
User	Communicates operational needs End customer for operational system	Communicates alternatives and risks, and provides perspective of what technology can (reasonably) provide

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/about/audiences.html
Last Modified: 24 July 2008

Sponsors and Contributors

Software Technology Roadmap

Original Sponsors:

- Ms. Darleen Druyun
  Principal Deputy Assistant Secretary of the Air Force for Acquisition (SAF/AQ)
- Mr. John Willison
  Chief, Software Architecture and Technology Division
  CECOM Software Engineering Center
  Ft. Monmouth, NJ

Original Contributors:

- Air Force Office of Scientific Research
- E-Systems, Inc.
- GTE Government Systems
- Kaman Sciences Corporation
- Lockheed Martin
- Loral (Lockheed Martin)
- Don O'Neill, Independent Consultant
- Peterson AFB
- Air Force Rome Laboratory
- TRW, Inc.

URL: http://www.sei.cmu.edu/str/about/sponsors.html
Last Modified: 24 July 2008

Defining Software Technology

Software Technology Roadmap

This document addresses software technology in its broadest interpretation. Technology is the practical application of scientific knowledge in a particular domain or in a particular manner to accomplish a task. For the purposes of this document, software technology is defined as: the theory and practice of various sciences (to include computer, cognitive, statistical sciences, and others) applied to software development, operation, understanding, and maintenance.

More specifically, we view software technology as any concept, process, method, algorithm, or tool, whose primary purpose is the development, operation, and maintenance of software or software-intensive systems. Technology is not just the technical artifacts, but the knowledge embedded in those artifacts and the knowledge required for their effective use. Software technology may include the following:

- Technology directly used in operational systems, for example: two tier/ three tier software architectures, public key digital signatures, remote procedure calls (RPCs), rule-based intrusion detection.
- Technology used in tools that produce (or help produce) or maintain operational systems, for example: graphical user interface (GUI) builders, cyclomatic complexity, Ada 95 programming language, technologies for design rationale capture.
- Process technologies that make people more effective in producing and maintaining operational systems and tools by structuring development approaches or enabling analysis of systems/product lines. Examples include: Personal Software Process[1] (PSP) for Module-Level Development, Cleanroom Software Engineering, Domain Engineering and Domain Analysis.

[1] Personal Software Process and PSP are service marks of Carnegie Mellon University

URL: http://www.sei.cmu.edu/str/descriptions/defining.html
Last Modified: 24 July 2008

Technology Categories

Software Technology Roadmap

To indicate just how broad our definition of software technology is, we identify below the various categories of entries that are found within this document. A technology description will not explicitly identify the category into which its subject falls, but the reader should be able to infer the category from the information in the entry.

- *Elemental Technology.* An elemental technology can (in general) be traced to a single, identifiable theory or concept related to software development, understanding, operation, or maintenance.
- *Composite Technology.* A composite technology is the integration of several elemental technologies. These component technologies each contribute in some substantive way to the overall composite. The component technologies may or may not have separate descriptions if they do, this is noted in the description of the composite technology.
- *Group of Technologies.* The document treats technologies as a group in three cases, depending on whether or not the technologies within the group are further distinguished and how the technologies differ from one another:
  - The group as a whole has important and distinguishing characteristics that make it worthy of consideration. But the document doesn't distinguish among technologies within the group, because the internal, external, or usage characteristics that distinguish them are unknown, inaccessible, proprietary, insignificant, or irrelevant to the purposes of the document.
  - Sometimes information is necessary to make a decision about whether or not to use any technology within the group, based on common characteristics of the technology group. In such cases, it is prudent to first consider the technologies in the aggregate before looking at individual technologies within the group.
  - Non-competing technologies that nevertheless contribute to the same application area are grouped together into a tutorial that describes how the technologies can be applied in that particular context.

  In any case, we define the group and describe common characteristics of the group. In the case where members within the group are further distinguished (in separate technology descriptions), we provide cross-references to those technologies.

- *Other Software Technology Topics.* There are certain issues of concern

that don't fit into the above categories, yet they are important to software technology. These include certain high-level concepts, such as COTS, component based development/integration, and open systems. In descriptions of these topics, we point to (and explain the relationship to) related technologies.

URL: http://www.sei.cmu.edu/str/descriptions/categories.html
Last Modified: 24 July 2008

Template for Technology Descriptions

Software Technology Roadmap

The purpose of a technology description is to uniquely identify the technology, to characterize the technology in terms of the properties of systems and measures of software quality that it affects, and to point out tradeoffs, benefits, risks and limitations that may arise in various situations of use. Each technology description also provides reference(s) to literature, indications of the current maturity of the technology, and cross references to related technologies.

Technology descriptions are not meant to be comprehensive--each description provides the reader with enough knowledge to decide whether to investigate the technology further, to find out where to go for more information, and to know what questions to ask in gathering more information.

Typically, technology descriptions range in size from four to six pages, depending on the amount of information available or the maturity of the technology.

Each technology description has a common format and includes these major sections:

- Status
- Note
- Purpose and Origin
- Technical Detail
- Usage Considerations
- Maturity
- Costs and Limitations
- Dependencies
- Alternatives
- Complementary Technologies
- Index Categories
- References and Information Sources
- Current Author/Maintainer
- External Reviewer
- Modifications

Pending

URL: http://www.sei.cmu.edu/str/descriptions/template/template_body.html
Last Modified: 24 July 2008

Section Explanations

Explanations of the various technology description sections are displayed in this frame.

- Software Technology Roadmap
- Background & Overview
- Technology Descriptions
  - Defining Software Technology
  - Technology Categories
  - Template for Technology Descriptions
- Taxonomies
- Glossary & Indexes

Ada 95

Software Technology Roadmap

Status

Complete

Purpose and Origin

Ada is a general-purpose, internationally-standardized computer programming language developed by the U.S. Department of Defense (DoD) to help software designers and programmers develop large, reliable applications. The Ada language enhances *portability*, *maintainability*, *flexibility*, *reliability*, and provides *interoperability* by standardization [Lawlis 96].

The Ada 95 (1995) version [AdaLRM 95] supersedes the 1983 standard Ada 83. It corrects some shortcomings uncovered from nearly a decade of using Ada 83, and exploits developments in software technology that were not sufficiently mature at the time of Ada's original design. Specifically, Ada 95 provides extensive support for object-oriented programming (OOP) (see Object-Oriented Programming Languages), efficient real-time concurrent programming, improved facilities for programming in the large, and increased ability to interface with code written in other languages.

When distinguishing between the two versions of the language, the 1983 version is referred to as Ada 83, and the revised version is referred to as Ada or Ada 95.

Technical Detail

Ada 95 consists of a core language that must be supported by all validated compilers, and a set of specialized needs annexes that may or may not be implemented by a specific compiler. However, if a compiler supports a special needs annex, all features of the annex must be supported. The following is the set of annexes [AdaLRM 95]:

Required annexes (i.e., part of core language)
A. Predefined Language Environment
B. Interface to Other Languages
J. Obsolescent Features

Optional special needs annexes

C. Systems Programming
D. Real-time Programming
E. Distributed Systems
F. Information Systems
G. Numerics
H. Safety and Security

Annexes K - P are for informational purposes only and are not part of the standard.

As in Ada 83, Ada 95 compilers are validated against established written standards- all standard language features exist in every validated Ada compiler. To become validated, a compiler must comply with the Ada Compiler Validation Capability (ACVC) suite of tests [AdaIC 97a, 97b]. Because of language standardization and required compiler validation, Ada provides an extremely high degree of support for interoperability and portability.

Like Ada 83, the Ada 95 language is independent of any particular hardware or operating system; the interface to any given platform is defined in a specific "System" package. Ada 95 improves on the Ada 83 features that support portability, which include the ability to define numerical types using system-independent declarations and the ability to encapsulate dependencies.

By requiring specifications such as type specifications, by performing consistency checks across separately compiled units, and by providing exception handling facilities, Ada 95, like Ada 83, provides a high degree of reliability when compared to other programming languages.

The Ada language was developed explicitly to support software engineering- it supports principles of good software engineering and discourages poor practices by prohibiting them where possible. Features supporting code clarity and encapsulation (use of packages, use of generic packages and subprograms with generic parameters, and private and limited private types) provide support for maintenance and *reusability*. Ada 95 also provides full support for object-oriented programming, which allows for a high level of reusability:

- encapsulation of objects and their operations
- OOP inheritance- allowing new abstractions to be built from existing ones by inheriting their properties at either compile time or runtime
- an explicit pointer approach to polymorphism- the programmer must decide to use pointers to represent objects [Brosgol 93]
- dynamic binding

Ada 95 also provides special features (hierarchical libraries and partitions) to assist in the development of very large and distributed software components and systems.

Ada 95 improves on the flexibility provided by Ada 83 for interfacing with other programming languages by better standardizing the interface mechanism and

providing an Interface to Other Languages Annex.

Ada 95 improves the specification of previous Ada features that explicitly support concurrency and real-time processing, such as tasking, type declarations, and low-level language features. A Real-Time Programming Annex has been added to better specify the language definition and model for concurrency. Ada 95 has paid careful attention to avoid runtime overhead for the new object-oriented programming (OOP) features and incurs runtime costs commensurate with the generality actually used. Ada 95 also provides the flexibility for the programmer to specify the desired storage reclamation technique that is desired for the application.

Usage Considerations

Ada 95 is essentially an upwardly-compatible extension to Ada 83 with improved support for embedded software systems, real-time systems, computationally-intensive systems, communication systems, and information systems [Lawlis 96]. In revising Ada 83 to Ada 95, incompatibilities were catalogued, tracked, and assessed by the standard revision committee [Taylor 95]. These incompatibilities have proven to be mostly of academic interest, and they have not been a problem in practice.[1]

Combined with at least static code analysis or formal proofs, Ada 95, like Ada 83, is particularly appropriate for use in safety-critical systems.

The Ada Joint Program Office (AJPO) supports Ada 95 by providing an Ada 95 Adoption Handbook [AJPO 95] and an Ada 95 Transition Planning Guide [AJPO 94], and helping form Ada 95 early adoption partnerships with DoD and commercial organizations. The Handbook helps managers understand and assess the transition from Ada 83 to Ada 95 and the Transition Guide is designed to assist managers in developing a transition plan tailored for individual projects [Patton 95]. Another valuable source for Ada 95 training is a multimedia CD-ROM titled *Discovering Ada*. This CD-ROM contains tutorial information, demo programs, and video clips [AdaIC 95].

Ada 95 is the standard programming language for new DoD systems; the use of any other language would require a waiver. Early DoD adoption partnerships who are working Ada 95 projects include the Marine Corps Tactical Systems Support Activity (MCTSSA), Naval Research and Development (NRAD), and the Joint Strike Fighter (JSF) aircraft program [AdaIC 96a].

The AJPO supported the creation of an Ada 95-to-Java J-code compiler. This means that Java programs can be created by using Ada. The compiler generates Java "class" files just as a Java language compiler does. Ada and Java components can even call each other [Wheeler 96]. This capability gives Ada, like Java, extensive portability across platforms and allows Internet programmers to take advantage of Ada 95 features unavailable in Java.

Maturity

On February 15, 1995, Ada 95 became the first internationally-standardized object-oriented programming language. As of April 1997, 51 validated compilers were available [Compilers 97]. The current validation suite (Version 2.1) provides the capability to validate the core language as well as the additional features in the annexes [AdaIC 97b].

Results from early projects, such as the Joint Automated Message Editing Software (JAMES) and Airfields [AdaIC 96a], indicate that Ada 95 is upwardly-compatible with Ada 83 and that some Ada 95 compilers are mature and stable enough to use on fielded projects [Patton 95]. However, as of the spring of 1996, Ada 95 tool sets and development environments were, in general, still rather immature as compared to Ada 83 versions. As such, platform compatibility, bindings (i.e., database, user interface, network interface) availability, and tool support should be closely evaluated when considering Ada 95 compilers.

Costs and Limitations

Common perceptions and conventional wisdom regarding Ada 83 and Ada 95 have been shown to be incorrect or only partially correct. These perceptions include the following:

- Ada is far too complex.
- Ada is too difficult to teach, to learn, to use.
- Ada is too expensive.
- Using Ada causes inefficiencies.
- Training in Ada is too expensive.
- Ada is old-fashioned.
- Ada is not object-oriented.
- Ada does not fit into COTS software.

Mangold examines these perceptions in some detail [Mangold 96].

Alternatives

Other programming languages to consider are Ada 83, C, C++, FORTRAN, COBOL, Pascal, Assembly Language, LISP, Smalltalk, or Java.

Complementary Technologies

The Ada-95-to-Java J-code compiler (discussed in Usage Considerations) enables applications for the Internet to be developed in Ada 95.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Ada 95
Application category	Programming Language (AP.1.4.2.1), Compiler (AP.1.4.2.3)
Quality measures category	Reliability (QM.2.1.2), Maintainability (QM.3.1), Interoperability (QM.4.1), Portability (QM.4.2), Scalability (QM.4.3), Reusability (QM.4.4)
Computing reviews category	Programming Languages (D.3)

References and Information Sources

[AdaLRM 95] *Ada95 Language Reference Manual*, International Standard ISO/IEC 8652: 1995(E), Version 6.0 [online]. Available WWW <URL: http://www.adahome.com/rm95/> (1995).

[AdaIC 95] AdaIC News Brief: November 3, 1995 [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/news/weekly/1995/95-11-03.html >(1995).

[AdaIC 96a] *AdaIC NEWS* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/news/> (1996).

[AdaIC 97a] *Validation and Evaluation Test Suites: The Ada compiler certification process* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/testing/> (1997).

[AdaIC 97b] Ada Compiler Validation Capability, Version 2.1 (ACVC 2.1) [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/compilers/acvc/95acvc/acvc2_1> (1997).

[AJPO 94] Ada Joint Program Office. *Ada 9X Transition Planning Guide: A Living Document and Working Guide for PEOs and PMs* Version 1.0. Falls Church, VA: Ada Joint Program Office, 1994.

[AJPO 95] Ada Joint Program Office. *Ada 95 Adoption Handbook: A Guide to Investigating Ada 95 Adoption* Version 1.1. Falls Church, VA: Ada Joint Program Office, 1995.

[Brosgol 93] Brosgol, Benjamin. "Object-Oriented Programming in Ada 9X." *Object Magazine 2*, 6 (March-April 1993): 64-65.

[Compilers 97] *Ada 95 Validated Compilers List* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/cgi-bin/vcl/report95.pl> (1997).

[HBAP 96] *Ada Home: The Home of the Brave Ada Programmers (HBAP)* [online]. Available WWW <URL: http://lglwww.epfl.ch:80/Ada/> (1996).

[Lawlis 96] Lawlis, Patricia K. *Guidelines for Choosing a Computer Language: Support for the Visionary Organization* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/> (1996).

[Mangold 96] Mangold, K. "Ada95-An Approach to Overcome the Software Crisis?" 4-10. *Proceedings of Ada in Europe 1995*. Frankfurt, Germany, October 2-6, 1995. Berlin, Germany: Springer-Verlag, 1996.

[Patton 95] Patton II, I. Lee. "Early Experiences Adopting Ada 95," 426-34. *Proceedings of TRI-Ada '95*. Anaheim, CA, November 5-10, 1995. New York, NY: Association for Computing Machinery, 1995.

[Taylor 95] Taylor, B. *Ada Compatibility Guide* Version 6.0 [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/docs/compat-guide/compat-guide6-0.txt> (1995).

[Tokar 96] Tokar, Joyce L. "Ada 95: The Language for the 90's and Beyond." *Object Magazine 6*, 4 (June 1996): 53-56.

[Wheeler 96] Wheeler, David A. *Java and Ada* [online]. Available WWW <URL: http://www.adahome.com/Tutorials/Lovelace/java.htm> (1996).

Current Author/Maintainer

Capt Gary Haines, AFMC SSSG
Cory Vondrak, TRW, Redondo Beach, CA

External Reviewers

Charles (Chuck) Engle (former AJPO director)

John Goodenough, SEI

Modifications

2 October 97: updated URL for [Compilers 97].
20 June 97: updated URLs for [AdaIC 96a] and [AdaLRM 95].
14 April 97: updated number of validated Ada compilers and validation suite information.
10 Jan 97 (original)

Pending

In March 1997, changes to Ada policy were directed by Mr. Emmett Page (ASD/C31). This technology does *not* reflect those changes.

A revised assessment of toolset maturity (see Maturity section) is also needed.

Footnotes

[1] From John Goodenough, SEI, in email to John Foreman, Re: Ada 95, August 16, 1996.

URL: http://www.sei.cmu.edu/str/descriptions/ada95_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Algorithm Formalization

Software Technology Roadmap

Status

Advanced

Purpose and Origin

In an effort to better understand computer algorithms, researchers in this area began to formally characterize the properties of various classes of algorithms. Initially, research centered on divide-and-conquer and global search algorithms. This initial research proved that these formal algorithm characterizations, called algorithm theories, could be used to synthesize implementations (code) for well-defined functions. Used in program generation or synthesis systems, the purpose of algorithm formalization is two-fold:

- The synthesis of *consistent*, highly CPU efficient algorithms for well-defined functions.
- The formal characterization of algorithm theory notions [Smith 93b]. A by-product of this formalization is the creation of a taxonomy of algorithm theories in which relationships between algorithm theories are formally characterized. These formal characterizations allow a developer to exploit more effectively the structure inherent in the problem space, and thereby allow him to derive or synthesize more *efficient* implementations.

To synthesize an algorithm for a problem using this technology, the essence of the problem and its associated problem domain must be captured in a collection of formal specifications.

Technical Detail

Algorithm synthesis is an emerging correct-by-construction methodology in which algorithm theories are refined to satisfy the constraints represented in an algebraic specification of the problem space [Smith 90]. These algorithm theories represent the structure common to a class of algorithms and abstract out concerns about the specific problem to be solved, the control strategy, the target language and style (e.g., functional versus imperative), and the target architecture. Because theorem provers are used to refine the algorithm theories, the resulting synthesized algorithm is guaranteed to be consistent with the problem specification. In other words, the synthesized algorithm is guaranteed to find solutions to the specified problem provided such solutions exist. If multiple

solutions are possible, an algorithm can be synthesized to return one, some, or all of them.

Synthesis systems incorporating formal algorithm theories operate as follows. The developer supplies a formal specification of the problem for which an algorithm is needed, and supplies formal specifications for the operations referenced in the problem specification (i.e., the domain theory). These specifications must be in a prescribed format and language. Using syntactic information drawn from the problem specification, the synthesis system selects candidate algorithm theories from a library of such theories. The developer selects one of these for refinement. The synthesis system then uses the semantic information provided by the problem and domain theories and- using a theorem prover- completes the refinement process. After the algorithm is generated, a developer will typically apply several computer assisted optimizations to the algorithm before compilation.

Coupling a theorem prover to the algorithm synthesis environment enables computer management of the inherent complexity of the problem and solution spaces, permitting computer management of complex code optimizations. For example, a synthesized algorithm (or implementation) is modified by a user-requested optimization only if the theorem prover is able to verify the consistency of the resulting code. For example, simplification of conditionals, function unfolding (inline expansion), and finite differencing are all possible.

Usage Considerations

The use of this technology encourages reuse of codified knowledge. Specifically, once a domain theory has been developed, it can be used to help define additional problem specifications within that domain, or it can be combined with other domain theories to characterize larger domains. Note, however, that the characterization of large and/or complex domains is non-trivial and may take considerable effort. With respect to the synthesis system itself, a developer is free to add additional algorithm theories to its library. However, the development of such algorithm theories is complex and will require in-depth knowledge of that class of algorithm.

Synthesizing algorithms from formal specifications involves a paradigm shift from traditional programming practice. Because formal specifications are used, developers must formally characterize what the operations in the problem domain do rather than stating how they do it. In addition, maintenance is not performed on the synthesized code. Instead, the problem specification is modified to reflect the new requirement(s), and an implementation is rederived.

Synthesis of algorithms from formal specifications is independent of the target programming language. However, the synthesis environments themselves may need to be modified to support particular target languages, or code translators may be needed to translate the code generated by the synthesis environment to the desired target language.

Algorithms for non-real time, well-defined deterministic functions- such as sorting or complex scheduling- can be synthesized using this technology. However,

additional work is required to determine whether this technology can be extended with notions state and nondeterminism.

Maturity

This technique, along with an algorithm synthesis prototype environment called Kestrel Interactive Development System (KIDS), was developed around 1986 [Smith 86, Smith 91]. Although it initially supported divide-and-conquer and global search algorithm theories, KIDS has been extended with more powerful algorithm theories and with more sophisticated constraint propagation mechanisms. KIDS has been used to synthesize a transportation scheduling algorithm used by US Transportation Command; this scheduling algorithm is able to schedule 10,000 movement requests in approximately one minute, versus hours for competitive scheduling algorithms [Smith 93c]. Ongoing research in this area includes a formalization of local search and formalizations of complex scheduling algorithms. Proof-of-concept scheduling algorithms have been synthesized for the nuclear power-plant domain in which

- scheduled activities can have complex interactions
- timing constraints are represented by earliest start/finish times

This technology is also being extended to address the synthesis of parallel algorithms [Smith 93b].

Costs and Limitations

Like all software development efforts, specification inconsistency may result in implementations that do not meet users' needs. However, the formal nature of problem specifications permits semi-automated investigation of problem specification properties. Adaptation of this technology requires knowledge of discrete mathematics at the level of first order logic and experience in developing formal specifications. Knowledge of constraint propagation, category theory, and resolution-based theorem proving is also required. In addition, formalization of various problem domains may be difficult; to effectively use this technology, special training may be required. However, there are currently no commercially-available, regularly-scheduled courses offered on this subject.

Dependencies

Constraint propagation, resolution-based theorem proving, finite differencing technology (used in verifiably correct optimizations), algebraic specification techniques, and specification construction techniques are enablers for this technology.

Alternatives

Other approaches to developing demonstrably correct algorithm implementations are based on formal verification or deductive synthesis. Software generation systems can be used to select and specialize an algorithm

implementation from a library of implementations, to assemble an algorithm for a collection of reusable code fragments, or to generate algorithm implementation stubs (i.e., they can generate code for some parts of an algorithm using syntactic rather than semantic information), but generally such implementations are not be guaranteed to be consistent with the problem specification.

Complementary Technologies

Category-theoretic specification construction methodologies are useful for developing and refining algorithm, domain, and problem theories. In addition, various domain analysis technologies can be used to investigate the structure of the problem domain.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Algorithm Formalization
Application category	Select or Develop Algorithms (AP.1.3.4)
Quality measures category	Consistency (QM.1.3.2), Provably Correct (QM.1.3.4), Throughput (QM.2.2.3)
Computing reviews category	Algorithms (I.1.2), Automatic Programming (D.1.2), Numerical Algorithms and Problems (F.2.1), Nonumerical Algorithms and Problems (F.2.2), Specifying and Verifying and Reasoning about Programs (F.3.1)

References and Information Sources

[Gomes 96]	Gomes, Carla P.; Smith, Douglas; & Westfold, Stephen. "Synthesis of Schedulers for Planned Shutdowns of Power Plants," 12-20. *Proceedings of the Eleventh Knowledge-Based Software Engineering Conference*. Syracuse, NY, September 25-28, 1996. Los Alamitos, CA: IEEE Computer Society Press, 1996.
[Smith 86]	Smith, Douglas R. "Top-Down Synthesis of Divide-and-Conquer Algorithms," 35-61. *Readings in Artificial Intelligence and Software Engineering*. Palo Alto, CA: Morgan Kaufmann, 1986.

[Smith 90] Smith, Douglas R. & Lowry, Michael R. "Algorithm Theories and Design Tactics." *Science of Computer Programming 14*, 2-3 (1990): 305-321.

[Smith 91] Smith, Douglas R. "KIDS-A Knowledge-Based Software Development System," 483-514. *Automating Software Design*. Menlo Park, CA: AAAI Press, 1991.

[Smith 93a] Smith, Douglas R. *Classification Approach to Design* (KES. U.93.4). Palo Alto, CA: Kestrel Institute, 1993.

[Smith 93b] Smith, Douglas R. "Derivation of Parallel Sorting Algorithms," 55-69. *Parallel Algorithm Derivation and Program Transformation*. New York, NY: Kluwer Academic Publishers, 1993.

[Smith 93c] Smith, Douglas R. "Transformational Approach to Transportation Scheduling," 60-68. *Proceedings of the Eighth Knowledge-Based Software Engineering Conference*. Chicago, IL, September 20-23, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

Current Author/Maintainer

Mark Gerken, Air Force Rome Laboratory

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/algorithm_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Application Programming Interface

Software Technology Roadmap

Status

Advanced

Purpose and Origin

Application Programming Interface (API) is an older technology that facilitates exchanging messages or data between two or more different software applications. API is the virtual interface between two interworking software functions, such as a word processor and a spreadsheet. This technology has been expanded from simple subroutine calls to include features that provide for *interoperability* and system *modifiability* in support of the requirement for data sharing between multiple applications. An API is the software that is used to support system-level integration of multiple commercial-off-the-shelf (COTS) software products or newly-developed software into existing or new applications. APIs are also a type of Middleware that provide for data sharing across different platforms; this is an important feature when developing new or upgrading existing distributed systems. This technology is a way to achieve the total cross-platform consistency that is a goal of open systems (see COTS and Open Systems-An Overview) and standards [Krechmer 92].

Technical Detail

An API is a set of rules for writing function or subroutine calls that access functions in a library. Programs that use these rules or functions in their API calls can communicate with any others that use the API, regardless of the others' specifics [Hines 96]. APIs work with a wide spectrum of application dialogues (i. e., interprogram communication schemes) to facilitate information exchange. These include database access, client/server, peer-to-peer, real-time, event-driven, store and forward, and transaction processing. APIs combine error recovery, data translation, security, queuing, and naming with an easy-to-learn interface that comprises simple but powerful actions/commands (verbs). To invoke an API, a program calls a SEND-type function, specifying parameters for destination name, pointers to the data, and return confirmation options. The API takes the data and does all the communications-specific work transparent to the application.

There are four types of APIs that are enablers of data sharing between different software applications on single or distributed platforms:

- Remote Procedure Calls (RPCs)
- Standard Query Language (SQL)
- file transfer
- message delivery

Using RPCs, programs communicate via procedures (or tasks) that act on shared data buffers. SQL is a non-procedural data access language that allows data sharing between applications by access into a common database. File transfer allows for data sharing by sending formatted files between applications. Message delivery provides data sharing by direct interprogram communications via small formatted messages between loosely- or tightly-coupled applications. Current standards that apply to APIs include the ANSI standard SQL API. There are ongoing efforts to define standards for the other types.

Usage Considerations

APIs can be developed for all computing platforms and operating systems or purchased for most platforms and operating systems. All four API types can be used both on homogeneous and multi-platform applications. However, because of the added complexity required to share data across multiple platforms, RPC, SQL or file transfer APIs are better used to facilitate communication between different applications on homogenous platform systems. These APIs communicate data in different formats (e.g., shared data buffers, database structures, and file constructs). Each data format requires different network commands and parameters to communicate the data properly and can cause many different types of errors. Therefore, in addition to the knowledge required to perform the data sharing tasks, these types of APIs must account for hundreds of network parameters and hundreds of possible error conditions that each application must understand if it is to deliver robust interprogram communications. A message delivery API, in contrast, will offer a smaller subset of commands, network parameters, and error conditions because this API deals with only one format (messages). Because of this reduced complexity, message delivery APIs are a better choice when applications require data sharing across multiple platforms.

Maturity

Many examples of data sharing between different applications have been successfully implemented:

- Covia Technologies, in early 1983, supplied the Communication Integrator (CI), which was the enabler technology for the Apollo airline reservation system used by a consortium of United, British Air, Lufthansa, and other international airlines [King 95].
- DECMessageQ is part of the DECnet infrastructure and has been available since the early 1980s.
- Creative Systems Interface's (CSI) Application to Application Interface (AAI) is a full featured API that is suitable for both client-server and peer-to-peer applications.
- Horizon Strategies' Message Express was initially developed for LU6.2

(IBM generic System Network Architecture protocol) host and VAX/VMS communications. In a typical Message Express manufacturing application, remote plants with VAX, DOS/VSE, and AS/400 machines conduct work-order scheduling and inventory assessments via peer-to-peer messaging.

Costs and Limitations

APIs may "exist" in many forms; the potential user should comprehend the implications of each. APIs may be

- a bundled part of commercial software packages
- separately-licensed COTS software package(s) (license costs)
- uniquely-developed by a project using the internal capabilities/features of the applications that must communicate

In the last case, which should generally be the exception, the development staff will incur analysis and engineering costs to understand the internal features of the software applications, in addition to the cost to develop and maintain the unique API. In all cases, there are training costs associated with learning how to use the APIs as part of the development and maintenance activity. Additional costs are associated with developing and using APIs to communicate across multiple platforms. As already described, network communications add complexity to the development or use of the APIs. The kinds of costs associated with network applications include additional programming costs, training costs, and licenses for each platform.

Complementary Technologies

APIs can be used in conjunction with the Common Object Request Broker Architecture, Component Object Model (COM), DCOM, and Related Capabilities, Distributed Computing Environment, Two Tier Software Architectures, and Three Tier Software Architectures.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Application Programming Interface
Application category	Application Program Interfaces (AP.2.7)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1)

Computing reviews category	Distributed Systems (C.2.4) Software Engineering Tools and Techniques (D.2.2) Database Management Languages (H.2.3)

References and Information Sources

[Bernstein 96]	Bernstein, Philip A. "Middleware: A Model for Distributed Services." *Communications of the ACM 39*, 2 (February 1996): 86-97.
[Hines 96]	Hines, John R. "Software Engineering." *IEEE Spectrum* (January 1996): 60-64.
[King 95]	King, Steven S. "Message Delivery APIs: The Message is the Medium." *Data Communications 21*, 6 (April 1995): 85-90.
[Krechmer 92]	Krechmer, K. "Interface APIs for Wide Area Networks." *Business Communications Review 22*, 11 (November 1992): 72-4.

Current Author/Maintainer

Mike Bray, Lockheed-Martin Ground Systems (michael.w.bray@den.mmc.com)

External Reviewers

Paul Clements, SEI
John Kereschen, Lockheed Martin Command and Control Systems

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/api_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Architecture Description Languages

Software Technology Roadmap

Status

Complete

Purpose and Origin

When describing a computer software system, software engineers often talk about the *architecture* of the system, where an architecture is generally considered to consist of components and the connectors (interactions) between them.[1] Although architectural descriptions are playing an increasingly important role in the ability of software engineers to describe and understand software systems, these abstract descriptions are often informal and ad hoc.[2] As a result

- Architectural designs are often poorly understood and not amenable to formal analysis or simulation.
- Architectural design decisions are based more on default than on solid engineering principles.
- Architectural constraints assumed in the initial design are not enforced as the system evolves.
- There are few tools to help the architectural designers with their tasks [Garlan 93].

In an effort to address these problems, formal languages for representing and reasoning about software architecture have been developed. These languages, called architecture description languages (ADLs), seek to increase the *understandability* and *reusability* of architectural designs, and enable greater degrees of analysis.

Technical Detail

In contrast to Module Interconnection Languages (MILS), which only describe the structure of an implemented system, ADLs are used to define and model system architecture prior to system implementation. Further, ADLs typically address much more than system structure. In addition to identifying the components and connectors of a system, ADLs typically address:

- *Component behavioral specification.* Unlike MILs, ADLs are concerned with component functionality. ADLs typically provide support for specifying both functional and non-functional characteristics of

components. (Non-functional requirements include those associated with safety, security, reliability, and performance.) Depending on the ADL, timing constraints, properties of component inputs and outputs, and data accuracy may all be specified.

- *Component protocol specification.* Some ADLs, such as Wright [Garlan 94a] and Rapide [Luckham 95], support the specification of relatively complex component communication protocols. Other ADLs, such as UniCon [Shaw 95], allow the type of a component to be specified (e.g., filter, process, etc.) which in turn restricts the type of connector that can be used with it.
- *Connector specification.* ADLs contain structures for specifying properties of connectors, where connectors are used to define interactions between components. In Rapide, connector specifications take the form of partially-ordered event sequences, while in Wright, connector specifications are expressed using Hoare's Communicating Sequential Processes (CSP) language [Hoare 85].

As an example, consider the component shown in Figure 1. This component defines two data types, two operations (op), and an input and an output communication port. The component also includes specifications constraining the behavior of its two operations.

```
Component Simple is
      type in_type is ...
      type out_type is ...
      op f : in_type -> out_type
      op valid_input? : in_type -> Boolean
      port input_port : in_type
      port output_port : out_type
      axiom f-specification is
        (behavioral specification for the operation f)
      end axiom
      axiom valid_input?-specification is
        (behavioral specification for the operation
        valid_input?)
      end axiom
      interface is input_port!(x) ->
        ((output_port!f(x) -> Skip)
        < valid_input?(x) >
        (output_port!(Invalid_Data) -> Skip))
end Simple
```

Figure 1: Component

A protocol specification for this component, written in CSP, defines how it interacts with its environment. Specifically, component Simple will accept a data value x of type in_type on its input port, and, if the data value is valid, will output f (x) on its output port. If the data value is not valid, Simple will output an error message on its output port. Note that component Simple is a specification, not an implementation. Implementations of ADL components and connectors are expressed in traditional programming languages such as Ada (see Ada 83 and Ada 95) or C. Facilities for associating implementations with ADL entities vary between ADLs.

Usage Considerations

ADLs were developed to address a need that arose from programming in the large; they are well-suited for representing the architecture of a system or family of systems. Because of this emphasis, several changes to current system development practices may occur:

- *Training*. ADLs are formal, compilable languages that support one or more architectural styles; developers will need training to understand and use ADL technology and architectural concepts/styles effectively (e.g., the use of dataflow, layered, or blackboard architectural styles).
- *Change/emphasis in life-cycle phases*. The paradigm currently used for system development and maintenance may be affected. Specifically, architectural design and analysis will precede code development; results of analysis may be used to alter system architecture. As such, a growing role for ADLs is expected in evaluating competing proposed systems during acquisitions. An ADL specification should provide a good basis for programming activities [Shaw 95].
- *Documentation.* Because the structure of a software system can be explicitly represented in an ADL specification, separate documentation describing software structure is not necessary. This implies that if ADLs are used to define system structure, the architectural documentation of a given system will not become out of date.[3] Additionally, ADLs document system properties in a formal and rigorous way. These formal characterizations can be used to analyze system properties statically and dynamically. For example, dynamic simulation of Rapide [Luckham 95] specifications can be analyzed by automated tools to identify such things as communication bottlenecks and constraint violations. Further, these formal characterizations provide information that can be used to guide reuse.
- *Expanding scope of architecture*. ADLs are not limited to describing the software architecture; application to system architecture (to include hardware, software, and people) is also a significant opportunity.

Maturity

Several ADLs have been defined and implemented that support a variety of architectural styles, including

- Aesop, which supports the specification and analysis of architectural styles (formal characterizations of common architectures such as pipe and filters, and client-server) [Garlan 94b].
- Rapide, which uses event posets to specify component interfaces and component interaction [Luckham 95].
- Wright, which supports the specification and analysis of communication protocols [Garlan 94a].
- MetaH, which was developed for the real-time avionics domain [Vestal 96].
- LILEAnna, which is designed for use with Ada and generalizes Ada's notion of generics [Tracz 93].

- UniCon, which addresses packaging and functional issues associated with components [Shaw 95].

Further information about these and other languages used to describe software architectures can be found in the *Software Architecture Technology Guide* and *Architectural Description Languages* [SATG 96, SEI 96].

Because ADLs are an emerging technology, there is little evidence in the published literature of successful commercial application. However, Rapide and UniCon have been used on various problems,[4] and MetaH appears to be in use in a commercial setting [Vestal 96]. ADLs often have graphical tools that are similar to CASE tools.

Costs and Limitations

The lack of a common semantic model coupled with differing design goals for various ADLs complicates the ability to share tool suites between them. Researchers are addressing this problem; an ADL called ACME is being developed with the goal that it will serve as an architecture interchange language.[5] Some ADLs, such as MetaH, are domain-specific.

In addition, support for asynchronous versus synchronous communication protocols varies between ADLs, as does the ability to express complex component interactions.

Dependencies

Simulation technology is required by those ADLs supporting event-based protocol specification.

Alternatives

The alternatives to ADLs include Module Interconnection Languages (which only represent the defacto structure of a system), object-oriented CASE tools, and various ad-hoc techniques for representing and reasoning about system architecture.

Another alternative is the use of VHSIC Hardware Description Language (VHDL) tools. While VHDL is often thought of exclusively as a hardware description language, its modularization and communication protocol modeling capabilities are very similar to the ones under development for use in ADLs.

Complementary Technologies

Behavioral specification technologies and their associated theorem proving environments are used by several ADLs to provide capabilities to define component behavior. In addition, formal logics and techniques for representing relationships between them are being used to define mappings between architectures within an ADL and to define mappings between ADLs.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Architecture Description Languages
Application category	Architectural Design (AP.1.3.1), Compiler (AP.1.4.2.3), Plan and Perform Integration (AP.1.4.4)
Quality measures category	Correctness (QM.1.3), Structuredness (QM.3.2.3), Reusability (QM.4.4)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2), Organization and Design (D.4.7), Performance (D.4.8), Systems Programs and Utilities (D.4.9)

References and Information Sources

[Garlan 93] Garlan, David & Shaw, Mary. "An Introduction to Software Architecture," 1-39. *Advances in Software Engineering and Knowledge Engineering* Volume 2. New York, NY: World Scientific Press, 1993.

[Garlan 94a] Garlan, D. & Allen, R. "Formalizing Architectural Connection," 71-80. *Proceedings of the 16th International Conference on Software Engineering*. Sorrento, Italy, May 16-21, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[Garlan 94b] Garlan, D.; Allen, R.; & Ockerbloom, J. "Exploiting Style in Architectural Design Environments." *SIGSOFT Software Engineering Notes 19*, 5 (December 1994): 175-188.

[Luckham 95] Luckham, David C., et al. "Specification and Analysis of System Architecture Using Rapide." *IEEE Transactions on Software Engineering 21*, 6 (April 1995): 336-355.

[Hoare 85] Hoare, C.A.R. *Communicating Sequential Processes*. Englewood Cliffs, NJ: Prentice Hall International, 1985.

[Paulisch 94] Paulisch, Frances. "Software Architecture and Reuse- An Inherent Conflict?" 214. *Proceedings of the 3rd International Conference on Software Reuse*. Rio de Janeiro, Brazil, November 1-4, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[Perry 92] Perry, D.E. & Wolf, A.L. "Foundations for the Study of Software Architectures."*SIGSOFT Software Engineering Notes 17*,4 (October 1992): 40-52.

[SATG 96] *Software Architecture Technology Guide* [online]. Available WWW <URL: http://www-ast.tds-gn.lmco.com/arch/guide.html> (1996).

[SEI 96] *Architectural Description Languages* [online]. Available WWW <URL: http://www.sei.cmu.edu/architecture/adl.html> (1996).

[Shaw 95] Shaw, Mary, et al. "Abstractions for Software Architecture and Tools to Support Them." *IEEE Transactions on Software Engineering 21*, 6 (April 1995): 314-335.

[Shaw 96] Shaw, M. & Garlan, D. *Perspective on an Emerging Discipline: Software Architecture*. Englewood Cliffs, NJ: Prentice Hall, 1996.

[STARS 96] *Scenarios for Analyzing Architecture Description Languages* Version 2.0 [online]. Originally available WWW <URL: http://www.asset.com/WSRD/abstracts/ABSTRACT_1183.html> (1996).

[Tracz 93] Tracz, W. "LILEANNA: a Parameterized Programming Language," 66-78. *Proceedings of the Second International Workshop on Software Reuse*. Lucca, Italy, March 24-26, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

[Vestal 93] Vestal, Steve. *A Cursory Overview and Comparison of Four Architecture Description Languages* [online]. Originally available FTP <URL: ftp://ftp.htc.honeywell.com/pub/dssa/papers/four_adl.ps> (1996).

[Vestal 96] Vestal, Steve. *Languages and Tools for Embedded Software Architectures* [online]. Available WWW <URL: http://www.htc.honeywell.com/projects/dssa/dssa_tools.html> (1996).

Current Author/Maintainer

Mark Gerken, Air Force Rome Laboratory

External Reviewers

Paul Clements, SEI
Paul Kogut, Lockheed Martin, Paoli, PA
Will Tracz, Lockheed Martin Federal Systems, Owego, NY

Modifications

10 Jan 97 (original)

Footnotes

[1] While definitions of *architecture*, *component*, and *connector* vary among researchers, this definition of architecture serves as a baseline for this technology description. A generally accepted definition describing the difference between a "design" and an "architecture" is that while a design explicitly addresses functional requirements, an architecture explicitly addresses functional and non-functional requirements such as reusability, maintainability, portability, interoperability, testability, efficiency, and fault-tolerance [Paulisch 94].

[2] Source: Garlan, David, et al. "ACME: An Architecture Interchange Language." Submitted for publication.

[3] However, one can easily imagine a case where an ADL is used to document the architecture, but then the project moves to the implementation phase and the ADL is forgotten. The code or low-level design migrates, but the architecture is lost. This is often referred to as architectural drift [Perry 92].

[4] For example, Rapide has been used to specify/ analyze the architecture model of the Sparc Version 9 64-bit instruction set, a standard published by Sparc International. Models of the extensions for the Ultra Sparc have also been done; they are used extensively in benchmarking Rapide simulation algorithms. Further information is available via the World Wide Web at http://anna.stanford.edu/rapide/rapide.html.

[5] Source: Garlan, David, et al. "ACME: An Architecture Interchange Language." Submitted for publication.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/adl_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections

- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Argument-Based Design Rationale Capture Methods for Requirements Tracing

Software Technology Roadmap

Status

Advanced

Note

We recommend Requirements Tracing--An Overview *as prerequisite reading for this technology description.*

Purpose and Origin

A design rationale is a representation of the reasoning behind the design of an artifact. The purpose of argument-based design rationale capturing methods is to track

- the discussions and deliberations that occur during initial requirements analysis
- the reasons behind design decisions
- the changes in the system over the course of its life, whether they are changes in requirements, design, or code (i.e., any software artifact)
- the reasons for and impact of the changes on the system

Replaying the history of design decisions facilitates the understanding of the evolution of the system, identifies decision points in the design phase where alternative decisions could lead to different solutions, and identifies dead-end solution paths. The captured knowledge should enhance the *evolvability* of the system.

The study of argument-based design rationale capture originated during the late 1950s and early 1960s with D. Englebart, who developed a conceptual framework called Humans Using Language, Artifacts, and Methodology in which they are Trained (H-LAM/T) and with Stephen Toulmin and his work concerning the representational form for arguments [Shum 94].

Technical Detail

There are two general approaches to argument-based design rationale capture,

both of which are based upon the entity-relationship paradigm:

1. The Issue Based Information Systems (IBIS) that deals with issues, positions, and arguments for which the emphasis is on recording the argumentation process for a single design [Ramesh 92].
2. The Questions, Options, and Criteria (QOC) notation [Shum 94], for which assessments are relationships between options, and criteria and arguments are used to conduct debate about the status of the entities and relationships.

Decision Representation Language (DRL) combines and extends the two approaches to provide support for computational services like dependency management, precedence management, and plausibility management. All of the approaches provide mechanisms for a breadth-first analytic understanding of issues, thus setting the context for concrete refinement of the design.

All of the information gathered using the above mentioned methods/languages is generally called process knowledge. The process knowledge is cross-referenced to the requirements created during the requirements engineering phase. The entities and relationships provide for the structuring of design problems, and they provide a consistent mechanism for decision making and tracking and communication among team members.

Laboratory and small-scale field experiments have been conducted to determine the utility and effectiveness of design rationale capturing methods. Potential benefits include the following:

- Revision becomes a natural process.
- Design rationale capture methods can help to keep the design meetings on track and help maintain a shared awareness of the meeting's process.
- The design rationale record can help identify interrelated issues that need to be resolved. Related arguments enable team members to prepare for the meeting and lead to a better solution.
- The methods can help originators of ideas understand how they are understood by the rest of the team. Note: More analysis is required before the utility of the methods for communicating understandings is fully demonstrated.

The records can be a valuable resource when it becomes necessary to reanalyze a previous decision. Note: There is no data on how frequently the revisitation is necessary, therefore, the benefits may invalidate the effort necessary to capture the information.

Potential pitfalls include the following:

- Care must be taken to avoid prolonged reflective processes and the extensive analysis of high-level or peripheral issues.
- There may be inconsistencies in categorizing the design rationale information in the database because one person's assumptions may be another person's rationale and yet another person's decision.
- Because of the nature of the semiformal language, the reader may need

to be familiar with the design to understand the design rationale as represented.

Usage Considerations

The use of this technology requires the development of a shared, consistent, and coherent requirements traceability policy by a project team. Each of the team members must provide commitment to the policy and procedures. A procedure for overall coordination must be developed. To date, these procedures are project-dependent and there is no consistent policy. It will require effort to generate and maintain the entities and relationships in the design rationale database for a given system.

Maturity

To date, there is at least one commercially-available tool to support the IBIS notation. The vendor also provides training and support for their tool. Proprietary tools to support the IBIS method are being used on government projects (e.g., a database exists with over 100,000 requirements under management) [Ramesh 92]. Tools to support the other methods are in various prototype stages.

Costs and Limitations

Argument-based design rationale capture methods and supporting tools require additional time and effort throughout the software life cycle. Individuals must generate and maintain the entity relationship diagrams for any and all of the methods. Training is essential to make effective use of the methods.

Dependencies

This technology makes use of entity-relationship modeling as the basis for the methods.

Alternatives

There are several alternative approaches to requirements traceability methods. Examples include: Process Knowledge Method, an extension of the argument-based approach that includes a formal representation to provide two way traceability between requirements and artifacts and facilities for temporal reasoning (i.e., mechanisms to use the captured knowledge), and Feature-Based Design Rationale Capture Method for Requirements Tracing , an approach that is centered around the distinctive features of a system.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Argument-Based Design Rationale Capture Methods for Requirements Tracing
Application category	Requirements Tracing (AP.1.2.3)
Quality measures category	Completeness (QM.1.3.1) Consistency (QM.1.3.2) Traceability (QM.1.3.3) Effectiveness (QM.1.1) Reusability (QM.4.4) Understandability (QM.3.2) Maintainability (QM.3.1)
Computing Reviews Category	Software Engineering Tools and Techniques (D.2.2) Software Engineering Design (D.2.10) Project and People Management (K.6.1)

References and Information Sources

[Gotel 95] Gotel, Orlena. *Contribution Structures for Requirements Traceability*. London, England: Imperial College, Department of Computing, 1995.

[Ramesh 92] Ramesh, Balasubramaniam & Dhar, Vasant. "Supporting Systems Development by Capturing Deliberations During Requirements Engineering." *IEEE Transactions on Software Engineering 18*, 6 (June 1992): 498-510.

[Ramesh 95] Ramesh, Bala; Stubbs, Lt Curtis; & Edwards, Michael. "Lessons Learned from Implementing Requirements Traceability." *Crosstalk, Journal of Defense Software Engineering 8*, 4 (April 1995): 11-15.

[Shum 94] Shum, Buckingham Simon & Hammond, Nick. "Argumentation-Based Design Rationale: What Use at What Cost?" *International Journal of Human-Computer Studies 40*, 4 (April 1994): 603-52.

Current Author/Maintainer

Liz Kean, Air Force Rome Laboratory

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/argument_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Maintenance of Operational Systems--An Overview

Software Technology Roadmap

Status

Complete

Note

This description provides background information for technologies for optimizing maintenance environments. We recommend Cyclomatic Complexity; Halstead Complexity Measures; Maintainability Index Technique for Measuring Program Maintainability; and Function Point Analysis as concurrent reading, as they contain information about specific technologies.

Purpose and Origin

Technologies specific to the maintenance of software evolved (and are still evolving) out of development-oriented technologies. As large systems have proliferated and aged, the special needs of the operational environment have begun to emerge. Maintenance is defined here as the modification of a software product after delivery to correct faults, improve performance or other attributes, or to adapt the product to a changed environment [IEEE 83]. Historically, the software lifecycle has usually focused on development. However, so much of a system's cost is incurred during its operational lifetime that maintenance issues have become more important and, arguably, this should be reflected in development practices. Systems are required to last longer than originally planned; inevitably, the percentage of costs going to maintenance has been steadily climbing. Hewlett-Packard estimates that 60% to 80% of its R&D personnel are involved in maintaining existing software, and that 40% to 60% of production costs were directly related to maintenance [Coleman 94]. There was a rule of thumb that eighty percent of a Department of Defense (DoD) system's cost is in maintenance; older Cheyenne Mountain Complex systems may have surpassed ninety percent. Yet software development practices still do not put much emphasis on making the product highly maintainable.

Cost and risk of maintenance of older systems are further exacerbated by a shortage of suitable maintenance skills; analysts and programmers are not trained to deal with these systems. Industry wide, it is claimed that 75%-80% of all operational software was written without the discipline of structured programming [Coleman 95]. Only a minuscule fraction of current operational systems were built using the object-oriented techniques taught today.

The purpose of this description is to provide a framework or a contextual reference for some of the maintenance and reengineering technologies described in this document.

Technical Detail

The operational system lifecycle. The operational environment has its own lifecycle that, while connected to the development lifecycle, has specific and unique characteristics and needs. As shown in Figure 15, a system's total lifecycle is defined as having four major phases:

- the development or pre-delivery phase
- the early operational phase
- the mature operational phase
- the evolution/replacement phase

Each of the phases has typical characteristics and problems. The *operational* phases are most of the lifecycle and cost. The narrative following describes each phase, and identifies specific technologies in (or planned for) this document that can be applied to correct or improve the situation. In almost every case, taking the proper action in a given phase can eliminate, or greatly reduce, problems in a later phase- at much less cost.

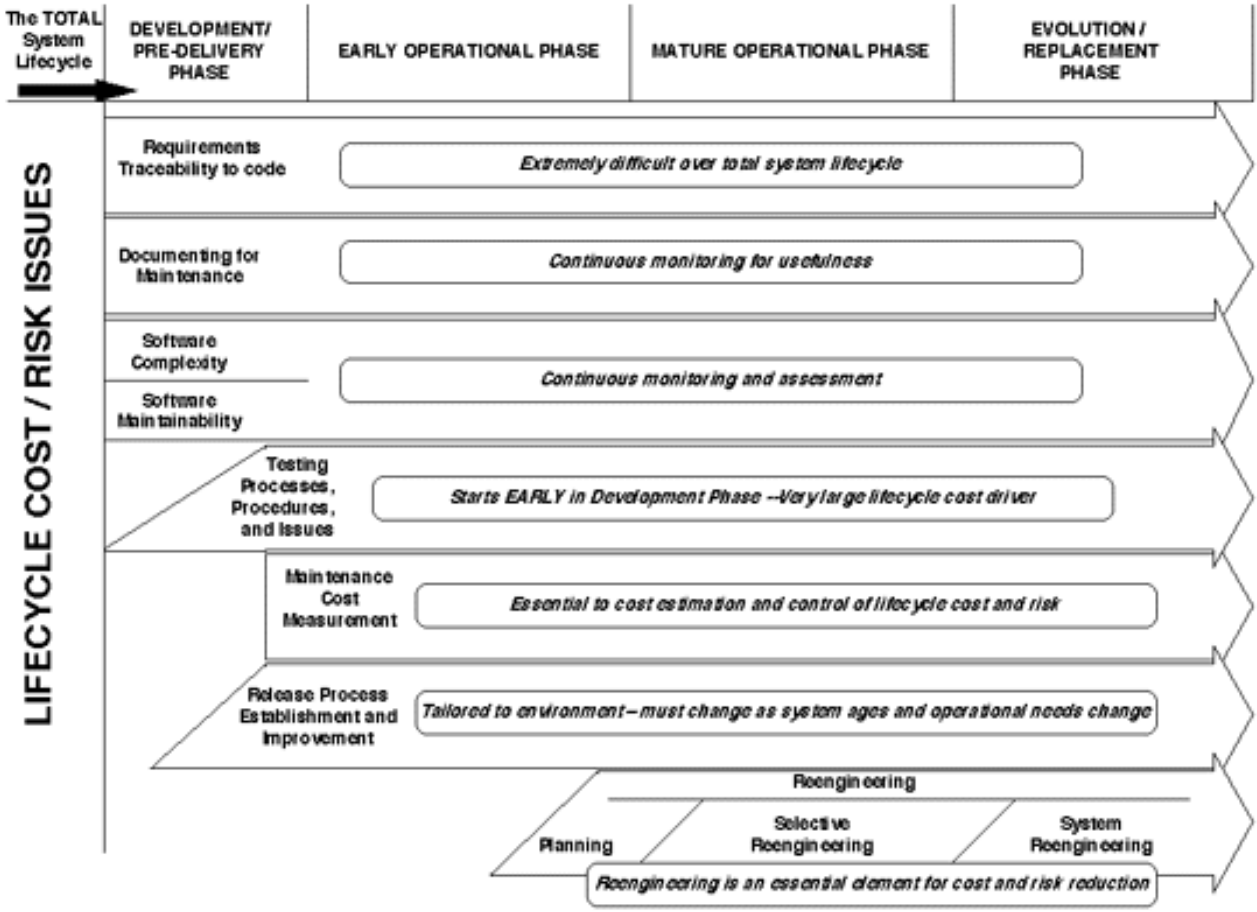


Figure 15: Total System Life Cycle

Terminology. To set a baseline for the descriptions of these phases, the following definitions are used:

Reengineering: rebuilding a piece of software to suit some new purpose (to work on another platform, to switch to another language, to make it more maintainable, etc.); often preceded by reverse engineering. Examination and alteration of a subject system to reconstitute it in a new form. Any activity that improves one's understanding of software, or prepares or improves the

software itself for increased maintainability, reusability, or evolvability.

Restructuring: transformation of a program from one representation to another at the same relative abstraction level, usually to simplify or clarify it in some way (e.g., remove GOTOs, increase modularity), while preserving external behavior.

Reverse engineering: the process of analyzing a system's code, documentation, and behavior to identify its current components and their dependencies to extract and create system abstractions and design information. The subject system is not altered; however, additional knowledge about the system is produced. Redocumenting and design recovery are techniques associated with reverse engineering.

Software complexity: some measure of the mental effort required to understand a piece of software.

Software maintainability: some measure of the ease and/or risk of making a change to a piece of software. The measured complexity of the software is often used in quantifying maintainability.

Translation: conversion of a program from one language to another, often as a companion action to restructuring the program.

Phase 1: The development or pre-delivery phase, when the system is not yet operational. Most of the effort in this phase goes into making Version One of the system function. But if total lifecycle costs are to be minimized, planning and preparation for maintenance during the development phase are essential. Most currently operational systems did not receive this attention during development. Several areas should be addressed:

- *Requirements traceability to code*. Requirements are the foundation of a system, and one of the most common faults of an operational system is that the relationship between its requirements and its code cannot be determined. Recovering this information for a system after it goes operational is a costly and time-consuming task. See Requirements Tracing, Feature-Based Design Rationale Capture Method for Requirements Tracing, and Argument-Based Design Rationale Capture Methods for Requirements Tracing for assistance in creating initial mappings from requirements to code.
- *Documentation and its usefulness in maintenance*. The ostensible purpose of documentation is to aid in understanding what the system does, and (for the maintenance programmer) how the system does it. There is at least anecdotal evidence that
    - Classical specification-type documentation is not a good primary source of information for the maintenance programmer looking for a problem's origin, especially since the documentation is frequently inconsistent with the code.
    - The most useful maintenance information is derived directly and automatically from the code; examples include structure charts, program flow diagrams, and cross-reference lists. This suggests that tools that create and maintain these documentation forms should be used during development of the code, and delivered with it.
- *The complexity of the software*. If the software is too complex to understand when it is first developed, it will only become more complex and brittle as it is changed. Measuring complexity during code development is useful for checking code condition, helps in quantifying testing costs, and aids in forecasting future maintenance costs (see Cyclomatic Complexity, Halstead Complexity Measures, and Maintainability Index

Technique for Measuring Program Maintainability).

- *The maintainability of the software*. This is perhaps the key issue for the maintainer. The ability to measure a system's maintainability directly affects the ability to predict future costs and risks. Maintainability Index Technique for Measuring Program Maintainability describes a practical approach to such a measurement, applicable throughout the lifecycle.

Phase 2: The early operational phase, when the delivered system is being maintained and changed to meet new needs and fix problems. Typically the tools and techniques used for maintenance are those that were used to develop the system. In this phase, the following issues are critical:

- *Complexity and maintainability* must be measured and controlled in this phase if the major problems of Phase 3 are to be avoided. Ideally, this a continuation of the same effort that began in Phase 1, and it depends on the same tools and techniques (see Cyclomatic Complexity, Halstead Complexity Measures, and Maintainability Index Technique for Measuring Program Maintainability). In a preventative maintenance regime, use of these types of measures will help establish guidelines about how much complexity and/or deterioration of maintainability is tolerable. If a critical module becomes too complex under the guidelines, it should be considered for rework before it becomes a problem. Early detection of problems, such as risk due to increasing complexity of a module, is far cheaper than waiting until a serious problem arises.
- *A formal release-based maintenance process* that suits the environment must be established. This process should always be subject to inspection, and should be revised when it does not meet the need.
- *The gathering of cost data* must be part of the maintenance process if lifecycle costs are to be understood and controlled. The cost of each change (e.g., person-hours, computer-hours) should be known down to a suitable granularity such as phase within the release (e.g., design, code and unit test, integration testing). Without this detailed cost information, it is very hard to estimate future workload or the cost of a proposed change.

Phase 3: Mature operational phase, in which the system still meets the users' primary needs but is showing signs of age. For example

- The incidence of bugs caused by changes or "day-one errors" (problems that existed at initial code delivery) is rising, and the documentation, especially higher-level specification material, is not trustworthy. Most analyses of changes to the software must be done by investigating the code itself.
- Code "entropy" and complexity are increasing and, even by subjective measures, its maintainability is decreasing.
- New requirements increasingly uncover limitations that were designed into the system.
- Because of employee turnover, the programming staff may no longer be intimately familiar with the code, which increases both the cost of a change and the code's entropy.
- A change may have a ripple effect: Because the true nature of the code is not well known, coupling across modules has increased and made it more likely that a change in one area will affect another area. It may be appropriate to restructure or reengineer selected parts of the system to lessen this problem.
- Testing has become more time-consuming and/or risky because as code complexity increases, test path coverage also increases. It may be appropriate to consider more sophisticated test approaches (see Preventive Maintenance).
- The platform is obsolete: The hardware is not supported by the manufacturer and parts are not readily available; the COTS software is not supported through new releases (or

the new releases will not work with the application, and it is too risky to make the application changes needed to align with the COTS software).

At this point, the code has not been rewritten en masse or reverse engineered to recover design, but the risk and cost of evolution by modification of the system have increased significantly. The system has become brittle with age. It may be appropriate to assess the system's condition. Sittenauer describes a quick methodology for gauging the need for reengineering, and the entire approach for measuring maintainability (see Maintainability Index Technique for Measuring Program Maintainability) allows continuous or spot assessment of the system's maintainability [Sittenauer 92].

Phase 4: Evolution/Replacement Phase, in which the system is approaching or has reached insupportability. The software is no longer maintainable. It has become so "entropic" or brittle that the cost and/or risk of significant change is too high, and/or the host hardware/software environment is obsolete. Even if none of these is true, the cost of implementing a new requirement is not tolerable because it takes too long under the maintenance environment. It is time to consider reengineering (see Cleanroom Software Engineering and Graphical User Interface Builders).

Usage Considerations

Software maintainability factors. The characteristics influencing or determining a system's maintainability have been extensively studied, enumerated, and organized. One thorough study is described in Oman; such characteristics were analyzed and a simplified maintainability taxonomy was constructed [Oman 91]. Maintainability Index Technique for Measuring Program Maintainability describes an approach to measuring and controlling code maintainability that was founded on several years of work and analysis and includes analysis of commercial software maintenance. References to other maintainability research results also appear in that technology description.

Preventive maintenance approaches. The approaches listed below are a few of the ways current technology can help to enhance system maintainability.

- *Complexity analysis*. Before attempting to reach a destination, it is essential to know where you are. For a software system, a good first step is measuring the complexity of the component modules (see Cyclomatic Complexity and Halstead Complexity Measures). Maintainability Index Technique for Measuring Program Maintainability describes a method of assessing maintainability of code using those complexity measures. Test path coverage can also be determined from complexity measures, which can help in optimizing system testing (see Test generation and optimization).
- *Functionality analysis*. Function Point Analysis describes the uses and limitations of function point analysis (also known as functional size measurement) in measuring software. By measuring a program's functionality, one can arrive at some estimate of its value in a system, which is of use when making decisions about rewriting the program or reengineering the system. Measures of functionality can also guide decisions about where to put testing effort (see Test generation and optimization).
- *Reverse engineering / design recovery*. Over time, a system's code diverges from the documentation; this is a well-known tendency of operational systems. Another phenomenon that is frequently underestimated or ignored is that (regardless of the divergence effect) the information required to make a given change is often found only in the code. Several approaches are possible here. Various tools offer the ability to

construct program flow diagrams (PFDs) from code. More sophisticated techniques, often classified as program understanding, are emerging. These technologies are implemented as tools that act as agents for the human analyst to assist in gathering information about a program's function at higher levels of abstraction than a program flow diagram (e.g., retask a satellite).

- *Piecewise reengineering*. If the system's known lifetime is sufficiently short, and if the evolutionary changes needed are sufficiently bounded, the system may benefit from a piecewise reengineering approach:
    - Brittle, high-risk modules that are likely to need changes are identified and reengineered to make them more maintainable. Techniques such as wrappers, an emerging technology, are expected to aid here.
    - For the sake of prudence, other risky modules are "locked," so that a prospective change to them can be made only after thoroughly assessing the risks involved.
    - For database systems, it may be possible to retrofit a modern relational or object-oriented database to the system; Common Object Request Broker Architecture and Graphic Tools for Legacy Database Migration describe technologies of possible use here. Piecewise reengineering can generally be done at a lower cost than complete reengineering of the system. If it is the right choice, it delays the inevitable obsolescence. The downsides of piecewise reengineering include the following:
    - Platform obsolescence is not reversed. Risks arising from the platform's software are unchanged; if the original database or operating system has risks, the application using them will also.
    - Unforeseen requirements changes still carry high risk if they affect the old parts of the system.
    - Performance may suffer because of the interface structures added to splice reengineered functions to old ones.
- *Translation/restructuring/modularizing*. Translation and/or restructuring of code are often of interest when migrating software to a new platform. Frequently the new environment will not support the old language or dialect. Restructuring/modularizing, or rebuilding the code to reduce complexity, can be done simply to improve the code's maintainability, but code to be translated is often restructured first so that the result will be less complex and more easily understood. There are several commercial tools that do one or more of these operations, and energetic research to achieve more automated approaches is being done. Welker cites evidence that translation does little or nothing to enhance maintainability [Welker 95]. Most often, it simply continues the existing problem in a different syntactical form; the mechanical forms output by translators decrease understandability, which is a key component of maintainability. None of these technologies is a cure-all, and none of them should be applied without first assessing the quality of the output and the amount of programmer resources required.

Test generation and optimization. Mission criticality of many DoD systems drives the maintenance activity to test very thoroughly. Boehm reported integration testing activities consuming only 16-34% of project totals [Boehm 81], but other evidence is available to show that commercial systems testing activity can take half of a development effort's resources [Alberts 76, DeMillo 87, Myers 79]. Recent composite post-release reviews of operational Cheyenne Mountain Complex system releases show that testing consumed 60-70% of the total release effort.[1] Any technology that can improve testing efficiency will have high leverage on the system's life-cycle costs. Technologies that can possibly help include: automatic test case generation; generation of test and analysis tools; redundant test case elimination; test data generation by chaining; techniques for software regression testing; and techniques for statistical test plan generation and coverage analysis.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Maintenance of Operational Systems--An Overview
Application category	Requirements Tracing (AP.1.2.3) Cost Estimation (AP.1.3.7) Test (AP.1.4.3) System Testing (AP.1.5.3.1) Regression Testing (AP.1.5.3.4) Reapply Software Lifecycle (AP.1.9.3) Reverse Engineering (AP.1.9.4) Reengineering (AP.1.9.5)
Quality measures category	Maintainability (QM.3.1)
Computing reviews category	Software Engineering Distribution and Maintenance (D.2.7) Software Engineering Metrics (D.2.8) Software Engineering Management (D.2.9)

References and Information Sources

[Alberts 76] Alberts, D. "The Economics of Software Quality Assurance." *National Computer Conference*. New York, NY, June 7-10, 1976. Montvale, NJ: American Federation of Information Processing Societies Press, 1976.

[Boehm 81] Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[Coleman 94] Coleman, Don, et al. "Using Metrics to Evaluate Software System Maintainability." *Computer 27,* 8 (August 1994): 44-49.

[Coleman 95] Coleman, Don; Lowther, Bruce; & Oman, Paul. "The Application of Software Maintainability Models in Industrial Software Systems." *Journal of Systems Software 29,* 1 (April 1995): 3-16.

[DeMillo 87] DeMillo, R., et al. *Software Testing and Evaluation*. Menlo Park, CA: Benjamin/ Cummings, 1987.

[IEEE 83] *IEEE Standard Glossary of Software Engineering Terminology*. New York, NY: Institute of Electrical and Electronic Engineers, 1983.

[Myers 79] Myers, G. *The Art of Software Testing.* New York, NY: John Wiley and Sons, 1979.

[Oman 91]	Oman, P.; Hagermeister, J.; & Ash, D. *A Definition and Taxonomy for Software Maintainability* (91-08-TR). Moscow, ID: Software Engineering Test Laboratory, University of Idaho, 1991.
[Sittenauer 92]	Sittenauer, Chris & Olsem, Mike. "Time to Reengineer?" *Crosstalk, Journal of Defense Software Engineering* 32 (March 1992): 7-10.
[Welker 95]	Welker, Kurt D. & Oman, Paul W. "Software Maintainability Metrics Models in Practice." *Crosstalk, Journal of Defense Software Engineering 8,* 11 (November/ December 1995): 19-23.

Current Author/Maintainer

Edmond VanDoren, Kaman Sciences, Colorado Springs

External Reviewers

Brian Gallagher, SEI
Ed Morris, SEI
Dennis Smith, SEI

Modifications

10 Jan 97 (original)

Footnotes

[1] Source: Kaman Sciences Corp. *Minutes of the 96-1 Composite Post-Release Review (CPRR), Combined CSS/CSSR and ATAMS Post-Release Review and Software Engineering Post-Release Review* KSWENG Memo # 96-03, 26 July, 1996.

URL: http://www.sei.cmu.edu/str/descriptions/mos_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms

- Expansion of footnotes
- Lists of related topics

Message-Oriented Middleware

Software Technology Roadmap

Status

Advanced

Note

We recommend Middleware *as prerequisite reading for this technology description.*

Purpose and Origin

Message-oriented middleware (MOM) is a client/server infrastructure that increases the *interoperability*, *portability*, and *flexibility* of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the *complexity* of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces- Application Programming Interfaces (APIs) that extend across diverse platforms and networks are typically provided by the MOM [Rao 95].

Technical Detail

Message-oriented middleware, as shown in Figure 22 [Steinke 95], is software that resides in both portions of a client/server architecture and typically supports asynchronous calls between the client and server applications. Message queues provide temporary storage when the destination program is busy or not connected. MOM reduces the involvement of application developers with the *complexity* of the master-slave nature of the client/server mechanism.

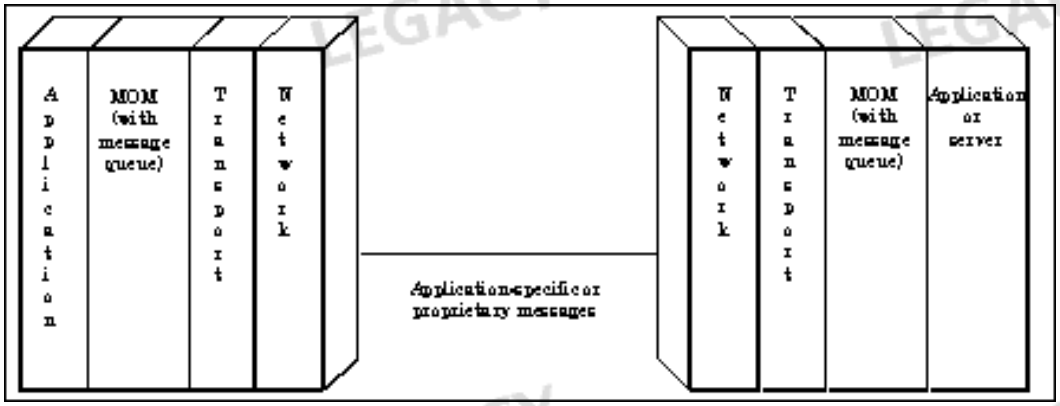


Figure 22: Message-Oriented Middleware

MOM increases the flexibility of an architecture by enabling applications to exchange messages with other programs without having to know what platform or processor the other application resides on within the network. The aforementioned messages can contain formatted data, requests for action, or both. Nominally, MOM systems provide a message queue between interoperating processes, so if the destination process is busy, the message is held in a temporary storage location until it can be processed. MOM is typically asynchronous and peer-to-peer, but most implementations support synchronous message passing as well.

Usage Considerations

MOM is most appropriate for event-driven applications. When an event occurs, the client application hands off to the messaging middleware application the responsibility of notifying a server that some action needs to be taken. MOM is also well-suited for object-oriented systems because it furnishes a conceptual mechanism for peer-to-peer communications between objects. MOM insulates developers from connectivity concerns- the application developers write to APIs that handle the complexity of the specific interfaces.

Asynchronous and synchronous mechanisms each have strengths and weaknesses that should be considered when designing any specific application. The asynchronous mechanism of MOM, unlike Remote Procedure Call (RPC) , which uses a a synchronous, blocking mechanism, does not guard against overloading a network. As such, a negative aspect of MOM is that a client process can continue to transfer data to a server that is not keeping pace. Message-oriented middleware's use of message queues, however, tends to be more flexible than RPC-based systems, because most implementations of MOM can default to synchronous and fall back to asynchronous communication if a server becomes unavailable [Steinke 95].

Maturity

Implementations of MOM first became available in the mid-to-late 1980s. Many MOM implementations currently exist that support a variety of protocols and operating systems. Many implementations support multiple protocols and operating systems simultaneously.

Some vendors provide tool sets to help extend existing interprocess communication across a heterogeneous network.

Costs and Limitations

MOM is typically implemented as a proprietary product, which means MOM implementations are nominally incompatible with other MOM implementations. Using a single implementation of a MOM in a system will most likely result in a dependence on the MOM vendor for maintenance support and future enhancements. This could have a highly negative impact on a system's flexibility, maintainability, portability, and interoperability.

The message-oriented middleware software (kernel) must run on every platform of a network. The impact of this varies and depends on the characteristics of the system in which the MOM will be used:

- Not all MOM implementations support all operating systems and protocols. The flexibility to choose a MOM implementation may be dependent on the chosen application platform or network protocols supported, or vice versa.
- Local resources and CPU cycles must be used to support the MOM kernels on each platform. The performance impact of the middleware implementation must be considered; this could possibly require the user to acquire greater local resources and processing power.
- The administrative and maintenance burden would increase significantly for a network manager with a large distributed system, especially in a mostly heterogeneous system.
- A MOM implementation may cost more if multiple kernels are required for a heterogeneous system, especially when a system is maintaining kernels for old platforms and new platforms simultaneously.

Alternatives

Other infrastructure technologies that allow the distribution of processing across multiple processors and platforms are

- Object Request Broker (ORB)
- Distributed Computing Environment (DCE)
- Remote Procedure Call (RPC)
- Transaction Processing Monitor Technology
- Three Tier Software Architectures

Complementary Technologies

MOM can be effectively combined with remote procedure call (RPC) technology-RPC can be used for synchronous support by a MOM.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Message-Oriented Middleware
Application category	Client/Server (AP.2.1.2.1) Client/Server Communication (AP.2.2.1)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1) Portability (QM.4.2)
Computing reviews category	Distributed Systems (C.2.4) Network Architecture and Design (C.2.1)

References and Information Sources

[Rao 95]	Rao, B.R. "Making the Most of Middleware." *Data Communications International 24*, 12 (September 1995): 89-96.
[Steinke 95]	Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

Current Author/Maintainer

Cory Vondrak, TRW, Redondo Beach, CA

External Reviewers

Ed Morris, SEI

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/momt_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Middleware

Software Technology Roadmap

Status

Advanced

Purpose and Origin

Middleware is connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. Middleware is essential to migrating mainframe applications to client/server applications and to providing for communication across heterogeneous platforms. This technology has evolved during the 1990s to provide for *interoperability* in support of the move to client/server architectures (see Client/Server Software Architectures). The most widely-publicized middleware initiatives are the Open Software Foundation's Distributed Computing Environment (DCE) , Object Management Group's Common Object Request Broker Architecture (CORBA), and Microsoft's COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities) [Eckerson 95].

Technical Detail

As outlined in Figure 17, middleware services are sets of distributed software that exist between the application and the operating system and network services on a system node in the network.

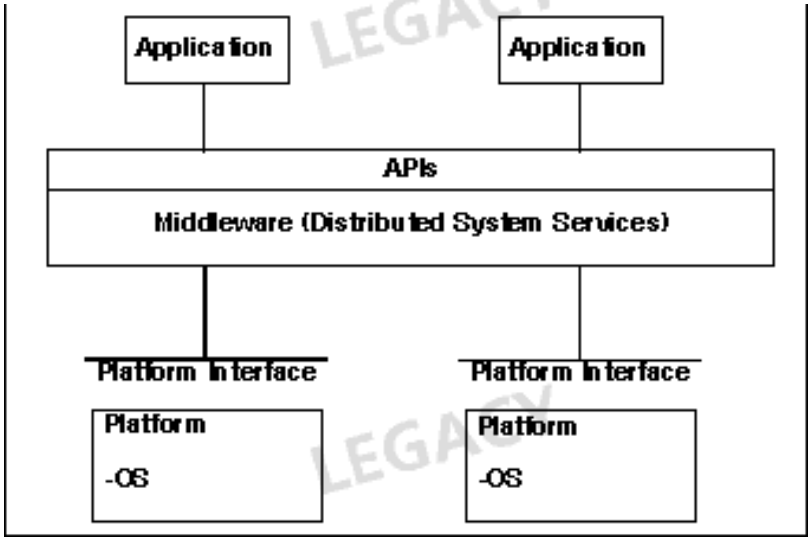


Figure 17: Use of Middleware [Bernstein 96]

Middleware services provide a more functional set of Application Programming Interfaces (API) than the operating system and network services to allow an application to

- locate transparently across the network, providing interaction with another application or service
- be independent from network services
- be reliable and available
- scale up in capacity without losing function [Schreiber 95]

Middleware can take on the following different forms:

- Transaction processing (TP) monitors (see Transaction Processing Monitor Technology), which provide tools and an environment for developing and deploying distributed applications.
- Remote Procedure Call (RPCs), which enable the logic of an application to be distributed across the network. Program logic on remote systems can be executed as simply as calling a local routine.
- Message-Oriented Middleware (MOM), which provides program-to-program data exchange, enabling the creation of distributed applications. MOM is analogous to email in the sense it is asynchronous and requires the recipients of messages to interpret their meaning and to take appropriate action.
- Object Request Brokers (ORBs) , which enable the objects that comprise an application to be distributed and shared across heterogeneous networks.

Usage Considerations

The main purpose of middleware services is to help solve many application connectivity and interoperability problems. However, middleware services are not a panacea:

- There is a gap between principles and practice. Many popular middleware services use proprietary implementations (making applications dependent on a single vendor's product).
- The sheer number of middleware services is a barrier to using them. To keep their computing environment manageably simple, developers have to select a small number of services that meet their needs for functionality and platform coverage.
- While middleware services raise the level of abstraction of programming distributed applications, they still leave the application developer with hard design choices. For example, the developer must still decide what functionality to put on the client and server sides of a distributed application [Bernstein 96].

The key to overcoming these three problems is to fully understand both the application problem and the value of middleware services that can enable the distributed application. To determine the types of middleware services required, the developer must identify the functions required, which fall into one of three classes:

1. Distributed system services, which include critical communications, program-to-program, and data management services. This type of service includes RPCs, MOMs and ORBs.
2. Application enabling services, which give applications access to distributed services and the underlying network. This type of services includes transaction monitors (see Transaction Processing Monitor Technology) and database services such as Structured Query Language (SQL).
3. Middleware management services, which enable applications and system functions to be continuously monitored to ensure optimum performance of the distributed environment [Schreiber 95].

Maturity

A significant number of middleware services and vendors exist. Middleware applications will continue to grow with the installation of more heterogeneous networks. An example of middleware in use is the Delta Airlines Cargo Handling System, which uses middleware technology to link over 40,000 terminals in 32 countries with UNIX services and IBM mainframes. By 1999, middleware sales are expected to exceed $6 billion [Client 95].

Costs and Limitations

The costs of using middleware technology (i.e., license fees) in system development are entirely dependent on the required operating systems and the types of platforms. Middleware product implementations are unique to the vendor. This results in a dependence on the vendor for maintenance support and future enhancements. This reliance could have a negative effect on a system's flexibility and maintainability. However, when evaluated against the cost of developing a unique middleware solution, the system developer and maintainer may view the potential negative effect as acceptable.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Middleware
Application category	Client/Server (AP.2.1.2.1) Client/Server Communication (AP.2.2.1)
Quality measures category	Interoperability (QM.4.1)
Computing reviews category	Distributed Systems (C.2.4) Network Architecture and Design (C.2.1) Database Management Languages (D.3.2)

References and Information Sources

[Bernstein 96]	Bernstein, Philip A. "Middleware: A Model for Distributed Services." *Communications of the ACM 39*, 2 (February 1996): 86-97.
[Client 95]	"Middleware Can Mask the Complexity of your Distributed Environment." *Client/Server Economics Letter 2*, 6 (June 1995): 1-5.
[Eckerson 95]	Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." *Open Information Systems 10*, 1 (January 1995): 3(20).
[Schreiber 95]	Schreiber, Richard. "Middleware Demystified." *Datamation 41*, 6 (April 1, 1995): 41-45.

Current Author/Maintainer

Mike Bray, Lockheed-Martin Ground Systems

Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/middleware_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Model-Based Verification (MBV) for Software

Software Technology Roadmap

Status

Draft

Purpose and Origin

Model-Based Verification (MBV) is a systematic approach for detecting defects (errors) in software requirements, designs, or code [Gluch 98]. The approach incorporates the use of mathematical models to provide a disciplined and logical analysis practice rather than a "proof" of correctness strategy. MBV involves creating essential models of system behavior and analyzing these models against formal representations of expected properties.

Essential models are simplified formal representations that capture the essence of a system rather than providing an exhaustive, detailed description of it. By selecting a system's critical (important or risky) parts and appropriately abstracted perspectives, a reviewer using model-based techniques can focus the analysis on the critical and technically difficult aspects of the system. Driven by the discipline and rigor required in the creation of a formal model, simply building the model uncovers errors.

Once the formal model is built, it can be checked using automated model checking tools. This analysis reveals potential defects while formulating claims about the system's expected behavior. Model checking has been shown to uncover even the most difficult to identify errors&emdash;those that result from the complexity associated with multiple interacting and interdependent components. These include embedded and highly distributed applications.

Technical Detail

MBV consists of a set of engineering practices for identifying and guiding the correction of defects in software artifacts. These practices are founded upon formal modeling and analysis techniques. As shown in Figure 1, MBV practices can be divided into two distinct categories of activities:

1. Project level (team) activities

2. Engineering (individual) activities

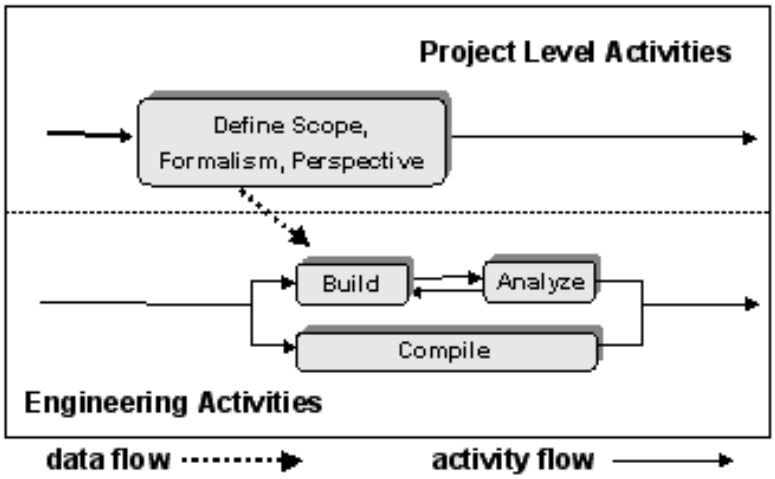


Figure 1. Model-Based Engineering Activities

Project level activities involve both programmatic and technical decisions and engage multiple members of the project team in important decisions. These decisions relate to defining the *scope* (how much of the system is to be modeled and analyzed), *formalism* (what techniques are to be employed), and *perspective* (what aspects of the system are to be considered) involved in the effort.

The bulk of the engineering activities in MBV are principally individual. Each engineer builds and analyzes models and compiles defects independently. While building MBV models, the systematic framework, attention to detail, and discipline demanded by the modeling techniques can lead to the identification of defects. During the analysis activities, automated model checking techniques are used to investigate the system's complexities and behavior and to identify elusive logic errors. While the individual engineer has principal responsibility for MBV activities, it is often valuable to collaborate with other experts. This is especially true during the analysis phase where additional domain expertise can help to focus the effort. Throughout the process, engineers compile the identified defects and the results of their analyses.

Usage Considerations

MBV is a pragmatic application of formal methods that focuses on error (defect) identification rather than formalized specification or proofs. This approach capitalizes on the advantages provided by formal methodologies without incurring the overhead costs normally associated with them.

It is expected that the MBV technology and practices will increase the effectiveness of verification and test activities and will result in higher quality software at a lower cost.

MBV will allow a practitioner to:

- **Detect errors early.** Since it can be used for requirements and preliminary design analysis of a system development, MBV offers the potential to detect errors early in the design process before they

propagate into the design or code.

- **Detect subtle errors.** Especially promising, is the potential for the early detection of subtle and potentially costly errors (defects) that are not identified even in extensive testing. Often these errors are not recognized until the software is in the field.
- **Effectively handle complexity.** MBV technology addresses system complexity that is difficult for humans to verify even through extensive individual review of the problem.
- **Identify basic errors.** The large size and complexity of software systems complicates verification by significantly increasing the amount of mundane checking that is needed (e.g., that there is consistent naming and use; that all the terms are defined) These editing-like activities can be effectively and efficiently accomplished through MBV automated checking.
- **Develop partial and targeted analyses.** The MBV approach can be applied to all or part of the system. This enables its efficient use in complex systems where the analysis can be targeted to only the critical areas of system.
- **Apply in all phases of development.** The modeling checking technique that is part of the MBV approach has been used for error detection in requirements, design, and code. It can be used as the foundation for testing by helping to define test strategies, test cases, and critical areas that require focused or more extensive testing.

Maturity

A number of published studies have cited successful implementation of the MBV approach for model checking for digital hardware and complex protocol systems. Some examples include:

- **IEEE standard Futurebus+ cache coherence protocol**&emdash; Although development began four years earlier and validation efforts had been conducted, model checking identified a number of previously undetected errors [Clarke 95].
- **High Speed Communications IC Chip**&emdash;During field tests of a complex high-speed communications IC chip, errors in the form of duplicated and lost data were observed. Model checking was able to help identify the cause of this error very quickly. Using simulation or other conventional techniques would have been impractical because it would have required an extraordinarily amount of time and resources [Fujita 96].
- **Power PC** &emdash; Model checking was employed to diagnose the cause of a design error found during hardware testing of the PowerPC 620 microprocessor. This error eluded detection despite extensive simulation and standard verification. Model checking techniques could have detected the error early in the design phase [Raimi 97].

The specific techniques and engineering practices of applying MBV to software verification have yet to be fully explored and documented. A number of barriers to MBV adoption have been identified including the lack of tool support, expertise in organizations, good training materials, and process support for formal modeling and analysis.

In order to address some of these issues, the Software Engineering Institute (SEI) has created a process framework for MBV practice. This process framework identifies a number of key tasks and artifacts. Additionally, the SEI is working on a series of technical notes that can be used by MBV practitioners. Each technical note is focused on a particular MBV task, providing guidelines and techniques for one aspect of the MBV practice. Currently, the technical notes that are planned address abstraction in building models, generating expected properties, generating formal claims, and interpreting the results of analysis.

Costs and Limitations

There are some limitations and barriers that determine how and when MBV can be effectively applied:

- **Domains:** Although MBV can in theory be applied to any software system, the technique is more productive in systems that involve real-time constraints, concurrency, distribution, or complex interactions among components.
- **Process maturity:** In order to be effective, MBV requires product artifacts that can easily be converted into models. For example, if MBV is being applied to a requirements specification, the requirements should be stated with a level of detail and rigor sufficient to create meaningful, reasonably complex models. In general, MBV will work better in mature organizations that produce sufficiently detailed and rigorous documents.
- **Training and expertise:** Although MBV does not require the mathematical background necessary to apply heavy formal methods, it does require some basic knowledge of formalisms and formal notations. This will hold true until more commercial tools simplify and automate MBV practices. Even more important than formal methods training is a practitioner's knowledge of the system under consideration.

Alternatives

Traditional formal methods are an alternative to MBV for critical systems. However, they are generally more expensive and require more expertise than MBV. On the other hand, traditional formal methods can be used to help prove the correctness of software whereas MBV can only increase the confidence in the system correctness.

Complementary Technologies

MBV can be included as part of a peer review team process [Software Inspections]. The specific activities of the MBV reviewer(s) are coordinated with those of the traditional review team. As part of a review team, an individual MBV reviewer's principal responsibility is much the same as an individual reviewer in a conventional inspection&emdash;identify defects in the artifact under review [Gluch 99].

MBV can potentially be applied in the context of Cleanroom Software

Engineering. The focus of Cleanroom involves moving from traditional, craft-based software development practices to rigorous, engineering-based practices. Cleanroom software engineering yields software that is correct by mathematically sound design, and software that is certified by statistically-valid testing. Reduced cycle times result from an incremental development strategy and the avoidance of rework.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Model-Based Verification (MBV) for Software
Application category	Requirements Engineering (AP.1.2.2) System Analysis & Optimization (AP.1.3.6) System Testing (AP.1.5.3.1)
Quality measures category	Correctness (QM.1.3) Availability/Robustness (QM.2.1.1) Accuracy (QM.2.1.2.1) Safety (QM.2.1.3) Real-time Responsiveness/Latency (QM.2.2.2)
Computing reviews category	Program Verification (D.2.4) Requirements/Specifications (D.2.1)

References and Information Sources

[Clarke 86] Clarke, E.; Emerson, E.A.; & Sistla, A.P. "Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications," *ACM Transcripts on Program Language Systems 8*, 2 (1986): 244-263.

[Clarke 95] Clarke, Edmund M., et al. "Verification of the Futurebus+ Cache Coherence Protocol." *Formal Methods in System Design 6*, 2 (March 1995): 217-232.

[Fujita 96] Fujita, M. "Debugging a Communications Chip." *IEEE Spectrum 33*, 6 (June 1996): 64.

[Gluch 98] Gluch, D. & Weinstock, C. *Model-Based Verification: A Technology for Dependable System Upgrade* (CMU/SEI-98-TR-009, ADA 354756). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1998. Available WWW: <http://www.sei.cmu.edu/publications/documents/98.reports/98tr009/98tr009abstract.html>

[Gluch 99] Gluch, D. & Brockway, J. *An Introduction to Software Engineering Practices Using Model-Based Verification* (CMU/SEI-99-TR-005, ESC-TR-99-005). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1999. Available WWW: <http://www.sei.cmu.edu/publications/documents/99.reports/99tr005/99tr005abstract.html>

[Harel 87] Harel, D. "Statecharts: A Visual Formalism for Complex Systems." *Science of Computer Programming 8*, 3 (June 1987): 231-274.

[Holt 99] Holt, A. "Formal Verification With Natural Language Specifications: Guidelines, Experiments And Lessons So Far." *South African Computer Journal 24* (November 1999): 253-257.

[McMillan 92] McMillan, K.L. *Symbolic Model Checking: An Approach to the State Explosion Problem* (CMU-CS-92-131). Pittsburgh, Pa.: Computer Science Department, Carnegie Mellon University, 1992.

[Raimi 97] Raimi, R. & Lear, J. "Analyzing a PowerPCTM 620 Microprocessor Silicon Failure Using Model Checking," 964-973. *Proceedings of the International Test Conference 1997*, Washington, D.C., November 1-6, 1997.

Current Author/Maintainer

Chuck Weinstock, Software Engineering Institute

Modifications

20 Mar 2001: Original

URL: http://www.sei.cmu.edu/str/descriptions/mbv_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Module Interconnection Languages

Software Technology Roadmap

Status

Complete

Purpose and Origin

As software system size and complexity increase, the task of integrating independently-developed subsystems becomes increasingly difficult. In the 1970s, manual integration was augmented with various levels of automated support, including support from module interconnection languages (MILs). The first MIL, MIL75, was described by DeRemer and Kron [DeRemer 76], who argued with integrators and developers about the differences between programming in the small, for which typical languages are suitable, and programming in the large, for which a MIL is required for knitting modules together [Prieto-Diaz 86]. MILs provide formal grammar constructs for identifying software system modules and for defining the interconnection specifications required to assemble a complete program [Prieto-Diaz 86]. MILs increase the *understandability* of large systems in that they formally describe the structure of a software system; they consolidate design and module assembly in a single language. MILs can also improve the *maintainability* of a large system in that they can be used to prohibit maintainers from accidentally changing the architectural design of a system, and they can be integrated into a larger development environment in which changes in the MIL specification of a system are automatically reflected at the code level and vice versa.

Technical Detail

A MIL identifies the system modules and states how they fit together to implement the system's function; MILs are *not* concerned with what the system does, how the major parts of the system are embedded in the organization, or how the individual modules implement their functions [Prieto-Diaz 86]. A MIL specification of a system constitutes a written description of the system design. A MIL specification can be used to

- Enforce system integrity and inter-modular compatibility.
- Support incremental modification. Modules can be independently compiled and linked; full recompilation of a modified system is not needed.
- Enforce version control. Different versions (implementations) of a module

can be identified and used in the construction of a software system. This idea has been generalized to allow different versions of subsystems to be defined in terms of different versions of modules. Thus MILs can be used to describe families of modules and systems [Tichy 79, Cooprider 79].

For example, consider the simplified MIL specification shown in Figure 24 and its associated graphical representation shown in Figure 25. The hypothetical MIL used in Figure 24 contains structures for identifying the modules of interest (in this case the modules are ABC, Z, and YBC); structures for identifying required and provided data; provided functions; and structures for identifying module and function versions. The module ABC defined in the figure consists of two parts, a function XA and a module YBC; the structure of each of these entities is also defined. Note that function XA has three versions, a Pascal, an Ada, and a FORTRAN version. These three versions would be written and compiled using their respective language development environments. A compilation system for this hypothetical MIL would process the specification given in Figure 24 to check that all required resources (such as x and z) are provided, and to check data type compatibility between required and provided resources. Provided these checks passed, the MIL compilation system, in conjunction with outside (user or environmental) inputs such as version availability and language choices, would select, compile (if necessary), and link the system. Incremental compilation is supported; for example, if the implementations for function XA change, the MIL compilation system will analyze the system structure and recompile and relink only those portions of the overall system affected by that change.

```
Module ABC
   provides a,b,c
   requires x,y
   consists-of function XA, module YBC

   function XA
      must-provide a
      requires x
      has-access-to Module Z
      real x, integer a
      realization
         version Pascal resources file
         (<Pascal_XA>) end Pascal
         version Ada resources file
         (<Ada_XA>) end Ada
         version FORTRAN resources file
         (<FORTRAN_XA>) end FORTRAN
   end XA

   Module YBC
      must-provide b, c
      requires a, y
      real y, integer a, b, c
   end YBC
end ABC
```

Figure 24: MIL Specification of a Simple Module

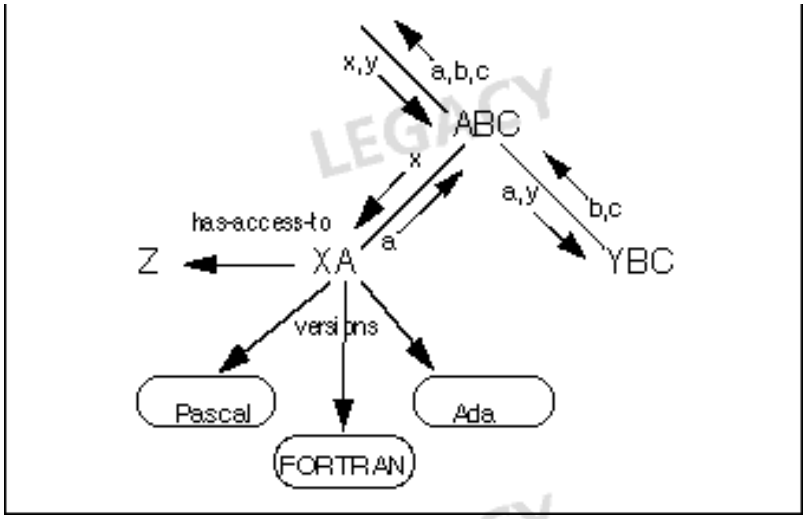


Figure 25: Graphical Representation

MILs do *not* attempt to do the following [Prieto-Diaz 86]:

- *Load compiled images.* This function is left to a separate facility within the development environment.
- *Define system function.* A MIL defines only the structure, not the function, of a system.
- *Provide type specifications.* A MIL is concerned with showing or identifying the separate paths of communication between modules. Syntactic checks along these communications paths may be performed by a MIL, but because MILs are independent of the language chosen to implement the modules they reference, such type checking will be limited to simple syntactic- not semantic- compatibility.
- Define embedded link-edit instructions.

Recently, MILs have been extended with notions of communication protocols [Garlan 94] and with constructs for defining semantic properties of system function. These extended MILs are referred to as Architecture Description Languages (ADLs).

Usage Considerations

MILs were developed to address the need for automated integration support when programming in the large; they are well-suited for representing the structure of a system or family of systems, and are typically used for project management and support. When adopting the use of MILs, an organization will need to consider the effect on its current system development and maintenance philosophy.

Because the structure of a software system can be explicitly represented in a MIL specification, separate documentation describing software structure may be unnecessary. This implies that if MILs are used to define the structure, then the architectural documentation of a given system will not become outdated.

Although some support is provided for ensuring data type compatibility, MILs

typically lack the structures required to define or enforce protocol compatibility between modules, and the structures necessary to enforce semantic compatibility.

Maturity

The MESA system at Xerox PARC was developed during 1975 and has been used extensively within Xerox [Geschke 77, Mitchell 79, Prieto-Diaz 86]. Other MILs have been proposed, defined, and implemented, but most of these appear to have been within a research context. For example, MIL concepts have been used to help design and build software reuse systems such as Goguen's library interconnection language (LIL) that was extended by Tracz for use with parameterized Ada components [Tracz 93]. Zand, et al., describe a system called ROPCO that can be used to "facilitate the selection and integration of reusable modules" [Zand 93].

At the time of publication, however, there are no tools supporting MILs and little research in this area.[1] Recent MIL-based research has shifted focus and now centers around the themes of software reuse and architecture description languages (ADLs). Architecture Description Languages can be viewed as extended MILs in that ADLs augment the structural information of a MIL with information about communication protocols [Garlan 94] and system behavior.

Costs and Limitations

MILs are formal compilable languages. Developers will need training to understand and use a MIL effectively. Training in architectural concepts may also be required.

The lack of a formal semantic for defining module function has at least the following implications:

- *Limited inter-module consistency checking.* MIL-based consistency checking is limited to simple type checking and- if supported- simple protocol checking.
- *Limited consistency checking among module versions.* MILs lack the facilities to ensure that different versions of a module satisfy a common specification, and may potentially lead to inconsistent versions within a family.
- *Limited type checking.* If mixing languages with a system, a developer may need to augment standard MIL tools with more sophisticated type checking utilities. For example, data types may be represented differently in C than in Ada, but the simple type checking found in a typical MIL will not flag unconverted value passing between languages.

Dependencies

Incremental compilers and linkers are required by most MILs.

Alternatives

Alternatives to MILs include documenting the structure of a system externally, such as in an interface control document or a structure chart. Architecture Description Languages (ADLs) can also be used to define the structure of a system, and are believed to be the current direction for this technology area.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Module Interconnection Languages
Application category	Architectural Design (AP.1.3.1) Compiler (AP.1.4.2.3) Plan and Perform Integration (AP.1.4.4)
Quality measures category	Correctness (QM.1.3) Structuredness (QM.3.2.3) Reusability (QM.4.4)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2) Organization and Design (D.4.7) Performance (D.4.8) Systems Programs and Utilities (D.4.9)

References and Information Sources

[Cooprider 79]	Cooprider, Lee W. *The Representation of Families of Software Systems* (CMU-CS-79-116). Pittsburgh, PA: Computer Science Department, Carnegie Mellon University, 1979.
[DeRemer 76]	DeRemer, F. & Kron, H. "Programming-in-the-Large Versus Programming-in-the-Small." *IEEE Transactions on Software Engineering SE-2*, 2 (June 1976): 321-327.
[Garlan 94]	Garlan, David & Allen, Robert. "Formalizing Architectural Connection," 71-80. *Proceedings of the 16th International Conference on Software Engineering*. Sorrento, Italy, May 16-21, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[Geschke 77]	Geschke, C.; Morris, J.; & Satterthwaite, E. "Early Experience with MESA." *Communications of the ACM 20*, 8 (August 1977): 540-553.
[Mitchell 79]	Mitchell, J.; Maybury, W.; & Sweet, R. *MESA Language Manual* (CSL-79-3). Palo Alto, CA: Xerox Palo Alto Research Center, April 1979.
[Prieto-Diaz 86]	Prieto-Diaz, Ruben & Neighbors, James. "Module Interconnection Languages." *Journal of Systems and Software 6*, 4 (1986): 307-334.
[Tichy 79]	Tichy, W. F. "Software Development Control Based on Module Interconnection," 29-41. *Proceedings of the 4th International Conference on Software Engineering*. Munich, Germany, September 17-19, 1979. New York, NY: IEEE Computer Society Press, 1979.
[Tracz 93]	Tracz, W. "LILEANNA: a Parameterized Programming Language," 66-78. *Proceedings of the Second International Workshop on Software Reuse*. Lucca, Italy, March 24-26, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.
[Zand 93]	Zand, M., et al. "An Interconnection Language for Reuse at the Template/Module Level." *Journal of Systems and Software 23*, 1 (October 1993): 9-26.

Current Author/Maintainer

Mark Gerken, Air Force Rome Laboratory

External Reviewers

Will Tracz, Lockheed Martin Federal Systems, Owego, NY

Modifications

10 Jan 97 (original)

Footnotes

[1] Source: Will Tracz in *Re: External Review - MILS*, email to Bob Rosenstein (1996).

Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Multi-Level Secure Database Management Schemes

Software Technology Roadmap

Status

Advanced

Note

We recommend Computer System Security--An Overview *as prerequisite reading for this technology description.*

Purpose and Origin

Conventional database management systems (DBMS) do not recognize different security levels of the data they store and retrieve. They treat all data at the same security level. Multi-level secure (MLS) DBMS schemes provide a means of maintaining a collection of data with mixed security levels. The access mechanisms allow users or programs with different levels of security clearance to store and obtain only the data appropriate to their level.

Technical Detail

As shown in Figure 20, multi-level secure DBMS architecture schemes are categorized into two general types:

- the Trusted Subject architecture
- the Woods Hole architectures

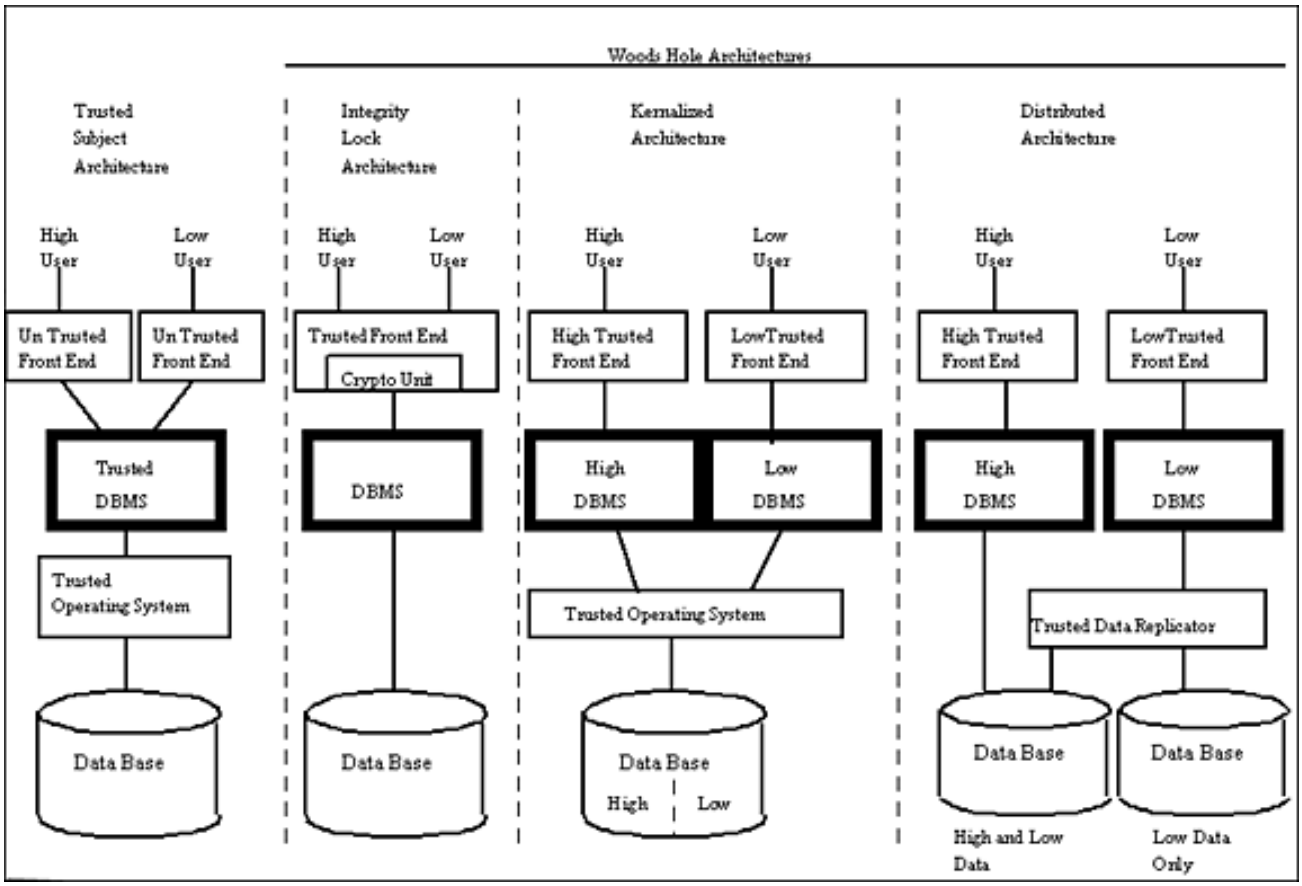


Figure 20: MLS DBMS Schemes

The Woods Hole architectures are named after an Air Force-sponsored study on multi-level data management security that was conducted at Woods Hole, Massachusetts.

The Trusted Subject architecture is a scheme that contains a trusted DBMS and operating system (see Trusted Operating Systems). The DBMS is custom-developed with all the required security policy (the security rules that must be enforced) developed in the DBMS itself. The DBMS uses the associated trusted operating system to make actual disk data accesses. This is the traditional way of developing MLS DBMS capabilities and can achieve high mandatory assurance for a particular security policy at the sacrifice of some DBMS functionality [Abrams 95]. This scheme results in a special purpose DBMS and operating system that requires a large amount of trusted code to be developed and verified along with the normal DBMS features.Trusted code provides security functionality and has been designed and developed using a rigorous process, tested, and protected from tampering in a manner that ensures the Designated Approving Authority (DAA) that it performs the security functions correctly. The DAA is the security official with the authority to say a system is secure and is permitted to be used. A benefit of the trusted subject architecture is that the DBMS has access to all levels of data at the same time, which minimizes retrieval and update processing. This scheme also can handle a wide range of sensitivity labels and supports complex access control. A sensitivity label identifies the classification level (e.g., confidential, secret) and a set of categories or compartments that apply to the data associated with the label.

The Woods Hole architectures assume that an untrusted (usually commercial-off-the-shelf (COTS)) DBMS is used to access data and that trusted code is developed around that DBMS to provide an overall secure DBMS system. The three different Woods Hole architectures address three different ways to wrap code around the untrusted DBMS.

The Integrity Lock architecture scheme places a trusted front end filter between the users and the DBMS. The filter provides security for the MLS. When data is added to the database, the trusted

front end filter adds an encrypted integrity lock to each unit of data added to the database. The lock is viewed by the DBMS as just another element in the unit stored by the DBMS. The encrypted lock is used to assure that the retrieved data has not been tampered with and contains the security label of the data. When data is retrieved, the filter decrypts the lock to determine if the data can be returned to the requester. The filter is designed and trusted to keep users separate and to store and provide data appropriate to the user. A benefit of this scheme is that an untrusted COTS DBMS can perform most indexed data storage and retrieval.

The Kernalized architecture scheme uses a trusted operating system and multiple copies of the DBMS; each is associated with a trusted front end. The trusted front end-DBMS pair is associated with a particular security level. Between the DBMS and the database, a portion of the trusted operating system keeps the data separated by security level. Each trusted front end is trusted to supply requests to the proper DBMS. The database is separated by security level. The trusted operating system separates the data when it is added to the database by a DBMS and combines the data when it is retrieved (if allowed by the security rules it enforces for the requesting DBMS). The high DBMS gets data combined from the high and low segments of the database. The low DBMS can only get data from the low segment of the database. A benefit of this scheme is that access control and separation of data at different classification levels is performed by a trusted operating system rather than the DBMS. Data at different security levels is isolated in the database, which allows for higher level assurance. Users interact with a DBMS at the user's single-session level.

The Distributed architecture scheme uses multiple copies of the trusted front end and DBMS, each associated with its own database storage. In this architecture scheme, low data is replicated in the high database. When data is retrieved, the DBMS retrieves it only from its own database. A benefit of this architecture is that data is physically separated into separate hardware databases. Since separate replicated databases are used for each security level, the front end does not need to decompose user query data to different DBMSs.

Castano and Abrams provide thorough discussions of these alternative architecture schemes and their merits [Castano 95, Abrams 95].

Usage Considerations

This technology is most likely to be used when relational databases must be accessed by users with different security clearances. This is typical of Command and Control systems. The different architectures suit different needs. The Trusted Subject architecture is best for applications where the trusted operating system and the hardware used in the architecture already provide an assured, trusted path between applications and the DBMS [Castano 95]. The Integrity Lock architecture provides the ability to label data down to the row (or record) level, the ability to implement a wide range of categories, and is easiest to validate [Castano 95]. The Kernalized architecture scheme is suited to MLS DBMS systems with more simple table structures because it is economical and easier to implement for simple structures [Castano 95]. The Distributed architecture is best suited for DBMSs where physical separation of data by security level is required [Abrams 95].

Maturity

The four different architectures have different maturity characteristics. As of August 1996, an R&D A1[1] system and six commercial[2] DBMSs have been implemented using the Trusted Subject architecture scheme for different assurance levels and security policies. One R&D system and one commercial DBMS have been implemented using the Integrity Lock architecture scheme. One R&D system and one commercial DBMS have been implemented using the Kernalized architecture scheme [Castano 95]. The Distributed architecture scheme has only been used in prototype

systems because of the high performance cost of the replicater, although one commercial DBMS claims to have this feature [Abrams 95]. This DBMS however, has not been evaluated by the National Computer Security Center (NCSC) [TPEP 96].

Costs and Limitations

Each of the different MLS architecture schemes has different costs and limitations. The Trusted Subject architecture scheme has a closely linked DBMS and Operating System that must be proven trusted together. This makes it hardest to validate and gives it the highest accreditation cost compared to the other schemes. The Integrity Lock architecture scheme requires that a Crypto Key management system is implemented and supported in operation. The Kernalized architecture requires a DBMS for each security level, which makes it expensive as more than two or three levels are considered. The Distributed architecture requires a different hardware platform for each security level and the data replicater provides a heavy processor and I/O load for high access data.

Dependencies

The MLS architecture schemes have individual dependencies. The Trusted Subject scheme is dependent on trusted schemes for a related DBMS and operating system. The Integrity Lock scheme is dependent on cryptographic technologies to provide the integrity lock. The Kernalized architecture scheme depends on Trusted Operating Systems technologies. The Distributed architecture scheme is dependent on efficient automatic data replication techniques.

Alternatives

The alternative to these technologies is to use a single-level DBMS and use manual review of retrieved data or have every user cleared for the data in the database. That may not be feasible in a Command and Control system.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Multi-Level Secure Database Management Schemes
Application category	Data Management Security (AP.2.4.2)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating Systems Security & Protection (D.4.6) Security & Protection (K.6.5) Computer-Communications Network Security and Protection (C.2.0)

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Castano 95] Castano, Silvana, et al. *Database Security*. New York, NY: ACM Press, 1995.

[DoD 85] *Department of Defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC)* (DoD 5200.28-STD 1985). Fort Meade, MD: Department of Defense, 1985. Also available WWW <URL: http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html> (1985).

[TPEP 96] *Trusted Product Evaluation Program Evaluated Product List* [online]. Available WWW <URL: http://www.radium.ncsc.mil/tpep/index.html> (1996).

Current Author/Maintainer

Tom Mills, Lockheed Martin

Modifications

10 Jan 97 (original)

Footnotes

[1] An A1 system is one that meets the highest (most stringent) set of requirements in the Department of Defense Trusted Computer Systems Evaluation Criteria (the Orange Book) [DoD 85]. See Trusted Operating Systems for a further description of the classes of trusted operating systems.

[2] A commercial DBMS does not imply a general-purpose DBMS. It means that it can be packaged and sold to other people. If a MLS DBMS has been developed to provide specific security functions that customers need, and the customer is willing to be restricted to that set of functions and use the same hardware and support software, then it can be sold as a product. It is then a commercial DBMS. The six commercial DBMSs that have been implemented with the Trusted Subject architecture are all different from each other, as they have been developed with different security policies for different hardware and software environments.

URL: http://www.sei.cmu.edu/str/descriptions/mlsdms_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Black-box Modernization of Information Systems

Software Technology Roadmap

Status

Draft

Note

We recommend Maintenance of Operational Systems--An Overview as prerequisite reading for this technology description.

Purpose and Origin

The criticality of enterprise information systems (EISs) in today's businesses requires organizations to manage system evolution as business practices change and new information technologies providing competitive advantage become available. EIS evolution becomes more difficult with time as systems are repeatedly modified and become increasingly outdated. Managing the evolution of outdated systems requires periodically modernizing these legacy systems to support evolving business practices and to incorporate modern information technologies.

System evolution is a broad term that covers a continuum, ranging from adding a field in a database to completely re-implementing a system. These system evolution activities can be divided into three categories [Weiderman 97]: maintenance, modernization, and replacement. Figure 1 illustrates how different evolution activities are applied at different phases of the operational system lifecycle. The dotted line represents growing business needs while the solid line represents the functionality provided by the information system. Repeated system maintenance supports the business needs sufficiently for a time, but as the system becomes increasingly outdated, maintenance falls behind the business needs. A modernization effort is then required that represents a greater effort, both in time and functionality, than the maintenance activity. Finally, when the old system can no longer be evolved, it must be replaced.

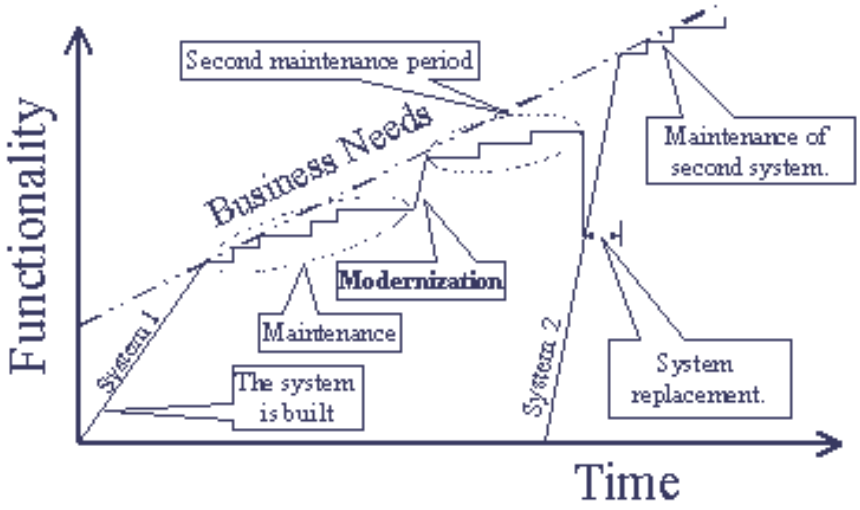


Figure 1 Information System Lifecycle

This description focuses on one phase in the life of a system: modernization. Modernization involves more extensive changes than maintenance, but conserves a significant portion of the existing system. These changes often include system restructuring, important functional enhancements, or new software attributes. Modernization is used when a legacy system requires more pervasive changes than previously possible during maintenance, but still embodies business value that must be preserved.

Technical Detail

System modernization can be classified by the level of system understanding required to support the modernization effort [Weiderman 97]. Modernization that requires knowledge of the internals of a legacy system is called white-box modernization, while modernization that only requires knowledge of the external interfaces of a legacy system is called black-box modernization. We concentrate on black-box (or non-intrusive) modernization techniques because they provide a useful way to leverage the existing investment in the legacy systems with limited effort.

Black-box modernization technologies possess various goals, strengths, and weaknesses. However, all are based on a similar approach consisting of wrapping the interface exported by the legacy system with a new, more homogeneous and usable interface. Wrapping, or encapsulation, is a technique to remove mismatches between the interface exported by a software artifact and the interfaces required by current integration practices [Weiderman 97]. Enumerated below are several of the most extended, non-intrusive modernization techniques in the market.

Screen scraping

A common technique for user interface (UI) modernization is screen scraping. Screen scraping [Carr 98], as shown in Figure 2, consists of wrapping old, text-based interfaces with new graphical interfaces. The old interface is often a set of text screens running in a terminal. In contrast, the new interface can be a PC-based, graphical user interface (GUI), or even a hypertext markup language (HTML) light client running in a Web browser. This technique can be extended easily, enabling one new UI to wrap a number of legacy systems. The new graphical interface communicates with the old one using a specialized commercial tool.

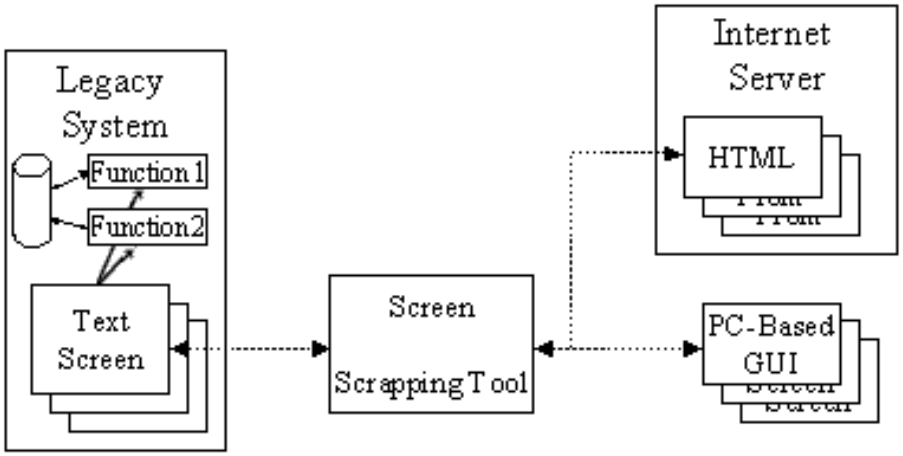


Figure 2 Legacy System Wrapping Using Screen Scraping

XML Wrapping

The Extensible Markup Language (XML™) is a broadly adopted format for structured documents and data on the Web. XML is a simple and flexible text format derived from standard generalized markup language (SGML) (ISO 8879) and developed by the World Wide Web Consortium® (W3C). XML is expanding from its origin in document processing and becoming a solution for data integration [Karpinski 98].

The keystone in the XML wrapping architecture is the XML server (Figure 3). The XML server acts as the contact point between the corporate infrastructure and the rest of the world. The XML server communicates by various means with the internal infrastructures including Enterprise Resource Planning (ERP) systems, databases, Electronic Data Interchange (EDIs), and other legacy systems. The XML server also interoperates with external organization by exchanging XML messages. There is an active market of solutions for non-intrusive integration of legacy infrastructures into XML servers. In addition, most commercial XML servers support multiple communication protocols enabling cost-effective integration with common legacy applications.

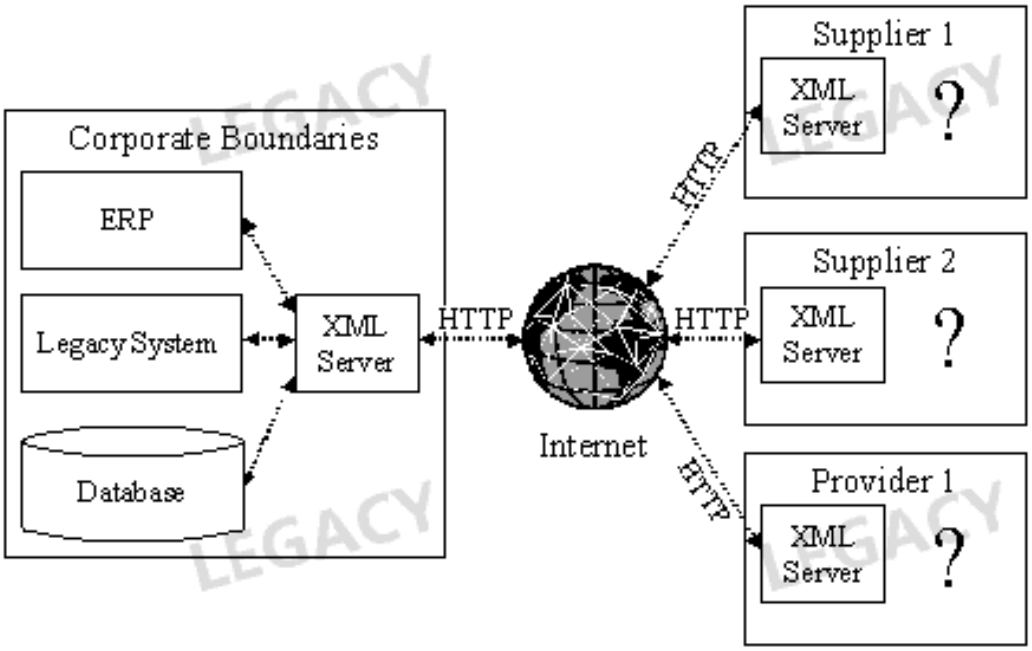


Figure 3 XML Integration

CGI integration

The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. Legacy integration using the CGI [Shklar] [Eichman 95] is often used to provide fast Web access to existing assets including mainframes and transaction monitors. As in screen scraping, a new graphical user interface (in this case always HTML pages) is created, but instead of wrapping the old user interface, the new GUI communicates directly with the core business logic or data of the legacy system.

A typical CGI access configuration is shown in Figure 4. A Web server, powered with a CGI extension to access legacy systems, invokes a function in the legacy system and generates HTML pages to be served to remote browsers. Although not depicted in the figure, CGI is used to access legacy data in addition to the logic.

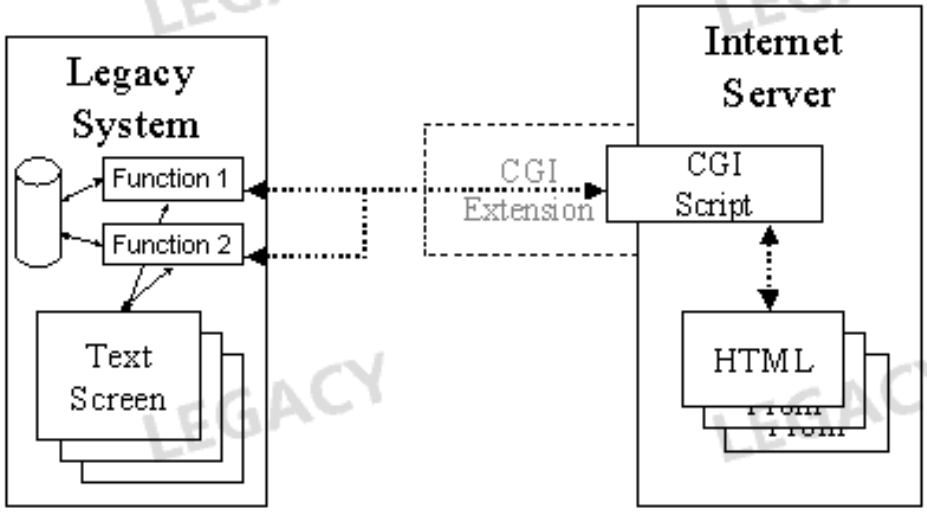


Figure 4 Legacy System Wrapping Using CGI Extensions

Object-Oriented Wrapping

Objects have been used to implement complex software systems successfully. Object-oriented (OO) systems can be designed and implemented in a way that closely resembles the business processes they model [Phoenix Group]. Additionally, the use of abstraction, encapsulation, inheritance, and other object-oriented techniques make object-oriented systems easier to understand.

The conceptual model of object-oriented wrapping is deceptively simple: individual applications are represented as objects; common services are represented as objects; and business data is represented as objects. In reality, object-oriented wrapping is far from simple and involves several tasks including code analysis, decomposition, and abstraction of the OO model. The project ERCOLE (Encapsulation, Reengineering, and Coexistence of Object with Legacy) describes an exemplifying process to wrap legacy applications with OO systems [De Lucia 97].

Usage Considerations

We have presented several techniques to support legacy system modernization. It would be naive to affirm any of these techniques as superior to the others. Each presented technique has strengths, weaknesses, and tradeoffs between cost, flexibility, and other variables. Table 1 summarizes the discussions of each presented technique.

	Artifact Modernized	Target	Strengths	Weaknesses
Screen Scraping	Text-based user interface	Graphical or web-based user interface	• Cost • Time to market • Internet support	• Flexibility • Limited impact on maintainability
XML Wrapping	Proprietary access protocol	XML server	• Flexibility • Tool support (future) • B2B	• Tool support (present) • Evolving technology
CGI Integration	Mainframe Data or TM services	HTML pages	• Cost • Internet support	• Flexibility • Applicability
OO Wrapping	Any Enterprise Resource	OO Model	• Flexibility	• Cost

Table 1 Comparison of Integration Techniques

Maturity

Screen scraping, CGI Integration and OO wrapping are mature technologies widely used and with multiple fielded systems. XML wrapping is newer and consequently less mature. However, XML technology is gaining momentum as XML vocabularies emerge in specific business domains such as finance, supply chains and e-commerce. In addition, a growing number of commercial enterprise application solutions are embracing XML.

Dependencies

Screen scraping, XML wrapping, CGI Integration and OO wrapping require some sort of middleware to connect the wrapper with the legacy system and the wrapper with the user entities. In the case of OO wrapping this middleware is often an Object Request Broker like the Common Object Request Broker Architecture (CORBA), or the Component Object Model (COM).

Alternatives

In some occasions, black-box modernization is not viable. This can be the case, for example, when the legacy interface is so cryptic that it cannot be understood without a previous examination of the legacy system internals. In those cases we need a more pervasive, white-box approach. White-box modernization requires an initial reverse engineering process to gain an understanding of the internal system operation. After the code is analyzed and understood, white-box modernization often includes some system or code restructuring.

Replacement (AKA big bang approach or cold turkey) [Bisdal 97] is appropriate for legacy systems that cannot keep pace with business needs and for which any kind of modernization is not possible or cost effective. Replacement is normally used with systems that are undocumented, outdated, or not extensible.

Complementary Technologies

Screen scraping can benefit by using Graphical User Interface Builders to generate the new graphical screens.

When deciding between modernization and system replacement the Maintainability Index Technique can provide insight into how fragile the system has become after repeated modifications.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Black-box Modernization of Information Systems
Application category	Adaptive Maintenance (AP.1.9.3.2) Perfective Maintenance (AP 1.9.3.3)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1) Reusability (QM.4.4)
Computing reviews category	Software Engineering Distribution and Maintenance (D.2.7) Software Engineering Management (D.2.9)

References and Information Sources

[Bisdal 97]	Bisdal, Jesus; Lawless, Deirdre; Wu, Bing; Grimson, Jane; Wade, Vincent; Richardson, Ray; & O'Sullivan, D. *An Overview of Legacy Information System Migration,* Proceedings of the 4th Asian-Pacific Software Engineering and International Computer Science Conference (APSEC 97, ICSC 97), 1997.
[Carr 98]	Carr, David F. *Web-Enabling Legacy Data When Resources Are Tight.* Internet World. August 10 1998.
[Comella-Dorda 00]	Comella-Dorda, Santiago; Wallnau, Kurt; Seacord, Robert C.; Robert, John. *Survey of Legacy System Modernization Approaches, A (CMU/SEI-00-TR-003).* Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University [online]. Available WWW: URL: http://www.sei.cmu.edu/publications/documents/00.reports/00tn003.html (2000).
[De Lucia 97]	De Lucia, A.; Di Lucca, G.A.; Fasolino, A.R.; Guerra, P.; & Petruzzelli, S. *Migrating Legacy Systems towards Object-Oriented Platforms,* International Conference of Software Maintenance (ICSM97), 1997.
[Eichman 95]	David Eichmann. *Application Architectures for Web-Based Data Access* [online]. Available WWW: URL: http://www.cs.rutgers.edu/~shklar/www4/eichmann.html
[Karpinski 98]	Karpinski, Richard. *Databases, Tools Push XML Into Enterprise.* Internet Week Online. Available WWW URL: http://www.internetwk.com/news1198/news111698-3.htm (November 1998).
[Phoenix Group 97]	Phoenix Group. *Legacy Systems Wrapping with Objects* [online]. Available WWW URL: http://www.phxgrp.com/jodewp.htm.
[Shklar]	Shklar, Leon. *Web Access to Legacy Data* [online]. Available WWW: URL: http://athos.rutgers.edu/~shklar/web-legacy/summary.html.
[Weiderman 97]	Weiderman, Nelson; Northrop, Linda; Smith, Dennis; Tilley, Scott; & Wallnau, Kurt; *Implications of Distributed Object Technology for Reengineering (CMU/SEI-97-TR-005 ADA326945).* Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, [online]. Available WWW URL: http://www.sei.cmu.edu/publications/documents/97.reports/97tr005/97tr005abstract.html (1997).

Current Author/Maintainer

Santiago Comella-Dorda, SEI

External Reviewers

Robert C. Seacord, SEI

Modifications

20 Mar 2001: Original

URL: http://www.sei.cmu.edu/str/descriptions/blackbox_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Capability Maturity Model Integration (CMMI)

Software Technology Roadmap

Status

Complete

Purpose and Origin

The purpose of Capability Maturity Model (CMM®) Integration℠ is to provide guidance for improving an organization's processes and its ability to manage the development, acquisition, and maintenance of products and services. CMM Integration places proven practices into a structure that helps an organization assess its organizational maturity and process area capability, establish priorities for improvement, and guide the implementation of these improvements.

CMM Integration was conceived to sort out the problem of using multiple Capability Maturity Models (CMMs). Three source models&emdash;(1) Capability Maturity Model for Software (SW-CMM) v2.0 draft C, (2) Electronic Industries Alliance/Interim Standard (EIA/IS) 731, and (3) Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98&emdash;were combined into a single model to be used in enterprise-wide process improvement and integration activities.

A common framework to support the future integration of other discipline-specific CMMI models was developed. In addition, all CMMI products were developed to be consistent and compatible with the International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) 15504 technical report for software process assessment.

Like other CMMs, CMMI models provide guidance for organizations to use when they develop or revise their processes. CMMI models are not processes or process descriptions. The actual processes used in an organization depend on many factors, including application domain(s) and organization structure and size.

The companion assessment method developed thus far for CMMI models is the Standard CMMI Assessment Method for Process Improvement (SCAMPI). This method is based on the CMM-Based Appraisal for Internal Process Improvement (CBA IPI) V1.1 assessment method and the Electronic Industries Alliance/ Interim Standard (EIA/IS) 731.2 Appraisal Method. SCAMPI satisfies the Assessment Requirements for CMMI (ARC) V1.0.

The SCAMPI method is a diagnostic tool that supports, enables, and encourages an organization's commitment to process improvement. The method helps an organization gain insight into its process area capability and organizational maturity by identifying strengths and weaknesses of its current processes relative to one or more of the CMMI models, including the Capability Maturity Model&emdash;Integrated for Systems Engineering and Software Engineering (CMMI-SE/SW).

Technical Detail

The phrase "CMM Integration" refers to a concept that has translated into a group of products called the CMMI Product Suite. This product suite consists of the CMMI Framework, CMMI models, an assessment method, and training materials.

Each CMMI model, because it is developed within the CMMI Framework and therefore has the architectural requirements of the framework, is designed to be used in concert with other CMMI models. Each model consists of required, expected, and informational elements that aid those pursuing process improvement in their organization.

An organization may choose to approach process improvement from either a process area capability perspective or an organizational maturity perspective. This decision influences the organization's choice of model representation they will use. The continuous representation supports the process area capability approach; whereas the staged representation supports the organizational maturity approach. The differences between these representations are mainly architectural (i.e., how the practices are organized and which are selected for emphasis). However, these differences imply advantages to using one representation over another depending on the organization's approach to process improvement.

Each CMMI model consists of an overview section, process areas, and appendixes. Organizations using a model for process improvement will primarily use the contents of the process areas to guide their improvement efforts. Each process area is a group of related best practices organized into elements such as the purpose, introductory notes, specific goals, generic goals, specific practices, generic practices, subpractices, work products, generic practice elaborations, and discipline amplifications.

The assessment method currently available for use with CMMI models is the Standard CMMI Assessment Method for Process Improvement (SCAMPI). Although SCAMPI is the only assessment method currently available, the intent is to support development of several different assessment methods that differ in cost, time to execute, and rigor. All assessment methods must conform to appropriate clauses of the most current Assessment Requirements for CMMI (ARC). To help ensure useful and credible results are obtained from SCAMPI assessments, a certification and authorization process has been developed for SCAMPI lead assessors.

CMMI training courses are also available. Introductory courses for the CMMI

model (either staged or continuous representation), an Intermediate Concepts of CMMI course, SCAMPI Lead Assessor Training, and a SCAMPI Lead Assessor Upgrade Training are currently available

Usage Considerations

There are two basic transition scenarios to the CMMI Product Suite. The first assumes an organization has begun its improvement efforts using either the Software CMM or the EIA/IS 731. The second scenario assumes that an organization has not used either the SW-CMM or EIA/IS 731 reference model for its improvement efforts, or that there have been no improvement efforts to date.

Those organizations that fall into scenario one may need to decide the best timing for transition to preserve the value of their existing plans toward, for example, a particular maturity level achievement. Organizations may also wish to consider the versatility offered by the continuous and staged representations in planning their long-term process improvement strategy. If the costs of total transition appear high, an interim strategy might be to augment the current plan with selected process areas having greatest business value. In any case, the current improvement effort will not be wasted, as the content of the CMMI models was carefully selected and derived from that of the Software CMM and EIA/IS 731.

Those organizations that fall into scenario two may have process improvement efforts under other quality initiatives such as ISO 9000 or Malcolm Baldrige. Such organizations can leverage their process improvement infrastructure and investment to more rapidly adopt the CMMI Product Suite. They also gain a reference model of effective practices that can be applied&emdash;across the value chain&emdash;to enhance the development of software-intensive products and associated services.

These organizations may wish to begin by considering whether approaching improvement is better served by emphasizing process area capability or organizational maturity. Each approach is complementary. A focus on process area capability allows the building of organizational maturity, and a focus on organizational maturity allows concentration on particular process area capabilities. Neither is mutually exclusive, but the choice will determine which representation of the model (continuous or staged) will best fit the needs of the organization for training and assessment.

Once a representation is chosen, planning can begin with an improvement strategy such as the IDEAL (initiating, diagnosing, establishing, acting, learning) model. Given sufficient senior management sponsorship, establishing a specific, technically competent group to guide and coordinate process improvement efforts has proven to be a best practice.

Regardless of scenario, training is a key element in the ability of any organization to adopt CMMI and is therefore a key part of the CMMI Product Suite. While an initial set of courses is provided by the SEI and its transition partners, organizations may wish to supplement these courses with internal instruction. This approach allows the focus of organizational attention to be

placed on the areas marked for greater attention due to their linkage to the product development value chain.

Maturity

CMMI is established technology evolved to a new level that enables the continued growth and expansion of the CMM technology to multiple disciplines. The CMMI Product Suite is new, Version 1.0 was released in August 2000. Integrated Product and Process Development (IPPD) was added in October 2000. Acquisition has been added to the CMMI Product Suite in draft form and is currently under public review. In late 2001, an updated version of the management, software engineering, systems engineering, and IPPD components of the model will be available as Version 1.1.

The model is in excellent condition for its intended role as a tool to stimulate enterprise-wide process improvement. Nevertheless, there remains a need to use such process tools to benchmark process area capability and organizational maturity. Refinements from actual use will be made to the model, just as refinements were made to the Software CMM when it was introduced. Thus, CMMI long-term plans include updates to the model that are designed to capture needed improvements to ensure that CMMI models continue to provide a rich usable set of best practices that can be the basis for accurate and reliable process assessments.

Costs and Limitations

Successful process improvement initiatives must be driven by the business objectives of the organization. Thus, process improvement objectives are derived from the business objectives. Process improvement objectives identify the processes and their outcomes that the organization wishes to improve.

Process improvement is a significant undertaking that requires senior-level management sponsorship and a firm commitment of resources to be successful. Further, it is a long-term commitment for the organization that cannot be approached and accomplished quickly.

The costs vary depending on the organization and its goals. However, the support of process improvement requires some additions to the organizational structure, such as an engineering process group.

Complementary Technologies

Complementary process improvement technologies include process improvement reference models such as SW-CMM, EIA/IS 731, IPD-CMM, and other CMMs (e.g., FAA-iCMM, SA-CMM, People CMM) as well as systems engineering and software engineering standards such as ISO 9000, ISO 12207, ISO 15504, and ISO 15288. The IDEAL (initiating, diagnosing, establishing, acting, learning) model is another related technology that characterizes process improvement as a sequence of life cycle activities beginning with obtaining senior management commitment and continuing through leveraging what has

been learned from deployed improvements to feed into a new cycle of process improvement. IDEAL enables organizations to use a variety of reference models for improvement.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Capability Maturity Model Integration (CMMI)
Application category	Used to Support Operational Systems (AP.1)
Quality measures category	Need Satisfaction Measures (QM.1) Performance Measures (QM.2) Maintenance Measures (QM.3) Adaptive Measures (QM.4) Organizational Measures (QM.5)
Computing reviews category	Software Management (K.6.3)

References and Information Sources

CMMI Product Development Team. CMMI -SE/SW, V1.0 Capability Maturity Model&endash;Integrated for Systems Engineering/Software Engineering, Version 1.0 Staged Representation (CMU/SEI-2000-TR-018). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, Available WWW <URL: http://www.sei.cmu.edu/publications/documents/00.reports/00tr018.html > August 2000.

CMMI Product Development Team. CMMI -SE/SW, V1.0 Capability Maturity Model&endash;Integrated for Systems Engineering/Software Engineering, Version 1.0 Continuous Representation (CMU/SEI-2000-TR-019). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, Available WWW <URL: http://www.sei.cmu.edu/publications/documents/00.reports/00tr019.html > August 2000.

CMMI Product Development Team. Assessment Requirements for CMMI (ARC): Method Description (CMU/SEI-2000-TR-011). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, Available WWW <URL: http://www.sei.cmu.edu/publications/documents/00.reports/00tr011.html > August 2000.

CMMI Product Development Team. Standard CMMI Assessment Method for Process Improvement (SCAMPI) Method Description, Version 1.0 (CMU/SEI-

2000-TR-009). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, Available WWW <URL: http://www.sei.cmu.edu/publications/documents/00.reports/00tr009.html > October 2000.

CMMI Official Web Site. Available WWW <URL: http://www.sei.cmu.edu/cmmi/>, August, 2000.

Dunaway, D. & Masters, S. *CMM-Based Appraisal for Internal Process Improvement (CBA IPI): Method Description (CMU/SEI-96-TR-007).* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, April 1996.

Electronic Industries Association. *Systems Engineering Capability Model (EIA/IS-731).* Washington, D.C.: Electronic Industries Association, 1998. Available WWW <URL: http://geia.org/sstc/G47/page6.htm >

Paulk, M. C., Weber, C. V., Curtis, B., & Chrissis, M. B. *The Capability Maturity Model: Guidelines for Improving the Software Process,* (SEI series in software engineering.) Addison-Wesley Publishing Company, Inc. 1995.

Software Engineering Institute. *Software CMM, Version 2 (Draft C).* Available WWW <URL: http://www.sei.cmu.edu/cmm/draft-c/c.html>, Oct. 22, 1997.

Software Engineering Institute. *CMMI A-Specification, Version 1.4* Available WWW <URL: http://www.sei.cmu.edu/cmmi/org-docs/aspec1.4.html >, April 19, 1999.

Current Author/Maintainer

Sandy Shrum, SEI
Mike Konrad, SEI

External Reviewers

Dennis Ahern, Northrop Grumman
Mike Phillips, SEI
Bill Peterson, SEI

Modifications

20 Mar 2001: Update
10 Oct 2000: Original

URL: http://www.sei.cmu.edu/str/descriptions/cmmi_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Cleanroom Software Engineering

Software Technology Roadmap

Status

Complete

Purpose and Origin

Cleanroom software engineering is an engineering and managerial process for the development of high-quality software with certified *reliability*. Cleanroom was originally developed by Dr. Harlan Mills [Linger 94, Mills 87]. The name "Cleanroom" was taken from the electronics industry, where a physical clean room exists to prevent introduction of defects during hardware fabrication. Cleanroom software engineering reflects the same emphasis on defect prevention rather than defect removal, as well as certification of reliability for the intended environment of use.

Technical Detail

The focus of Cleanroom involves moving from traditional, craft-based software development practices to rigorous, engineering-based practices. Cleanroom software engineering yields software that is correct by mathematically sound design, and software that is certified by statistically-valid testing. Reduced cycle time results from an incremental development strategy and the avoidance of rework.

It is well-documented that significant differences in cost are associated with errors found at different stages of the software life cycle. By detecting errors as early as possible, Cleanroom reduces the cost of errors during development and the incidence of failures during operation; thus the overall life cycle cost of software developed under Cleanroom can be expected to be far lower than industry average.

The following ideas form the foundation for Cleanroom-based development:

- Incremental development under statistical quality control (SQC). Incremental development as practiced in Cleanroom provides a basis for statistical quality control of the development process. Each increment is a complete iteration of the process, and measures of performance in each

increment (feedback) are compared with preestablished standards to determine whether or not the process is "in control." If quality standards are not met, testing of the increment ceases and developers return to the design stage.

- Software development based on mathematical principles. In Cleanroom development, a key principle is that a computer program is an expression of a mathematical function. The Box Structure Method is used for specification and design, and functional verification is used to confirm that the design is a correct implementation of the specification. Therefore, the specification must define that function before design and functional verification can begin. Verification of program correctness is performed through team review based on correctness questions. There is no execution of code prior to its submission for independent testing.
- Software testing based on statistical principles. In Cleanroom, software testing is viewed as a statistical experiment. A representative subset of all possible uses of the software is generated, and performance of the subset is used as a basis for conclusions about general operational performance. In other words, a "sample" is used to draw conclusions about a "population." Under a testing protocol that is faithful to the principles of applied statistics, a scientifically valid statement can be made about the expected operational performance of the software in terms of reliability and confidence.

Benefits of Cleanroom include significant improvements in *correctness*, *reliability*, and *understandability*. These benefits usually translate into a reduction in field-experienced product failures, reduced cycle time, ease of maintenance, and longer product life.

Usage Considerations

Cleanroom has been documented to be very effective in new development and reengineering (whole system or major subunits) contexts. The following discussion highlights areas where Cleanroom affects or differs from more conventional practice:

- Team-based development. A Cleanroom project team is small, typically six to eight persons, and works in a disciplined fashion to ensure the intellectual control of work in progress. Cleanroom teamwork involves peer review of individual work, but does not supplant individual creativity. Once the system architecture has been established and the interfaces between subunits have been defined, individuals typically work alone on a given system component. Individual designs are working drafts that are then reviewed by the team. In a large project, multiple small teams may be formed, with one for the development of each subsystem, thus enabling concurrent engineering after the top-level architecture has been established.
- Time allocation across life cycle phases. Because one of the major objectives of Cleanroom is to prevent errors from occurring, the amount of time spent in the design phase of a Cleanroom development is likely to be greater than the amount of time traditionally devoted to design.

Cleanroom, however, is not a more time-consuming development methodology, but its greater emphasis on design and verification often yields that concern. Management understanding and acceptance of this essential point- that quality will be achieved by design rather than through testing- must be reflected in the development schedule. Design and verification will require the greatest portion of the schedule. Testing may begin later and be allocated less time than is ordinarily the case. In large Cleanroom projects, where historical data has enabled comparison of traditional and Cleanroom development schedules, the Cleanroom schedule has equaled or improved upon the usual development time.

- Existing organizational practices. Cleanroom does not preclude using other software engineering techniques as long as they are not incompatible with Cleanroom principles. Implementation of the Cleanroom method can take place in a gradual manner. A pilot project can provide an opportunity to "tune" Cleanroom practices to the local culture, and the new practices can be introduced as pilot results to build confidence among software staff.

Maturity

Initial Cleanroom use within IBM occurred in the mid to late 80s, and project use continues to this day. Defense demonstration projects began approximately 1992. Cleanroom has been used on a variety of commercial and defense projects for which reliability was critically important. Some representative examples include the following:

- IBM COBOL/SF product, which has required only a small fraction of its maintenance budget during its operating history [Hausler 94].
- Ericsson OS-32 operating system project, which had a 70% improvement in development productivity, a 100% improvement in testing productivity, and a testing error rate of 1.0 errors per KLOC (represents all errors found in all testing) [Hausler 94].
- USAF Space Command and Control Architectural Infrastructure (SCAI) STARS [1] Demonstration Project at Peterson Air Force Base in Colorado Springs, CO. In this project, Cleanroom was combined with the TRW (spiral) Ada Process Model and some generated and reused code to produce software at a rate of $30-40 per line of code versus industry averages of $130 per line for software of similar complexity and development timeframe (the size of the application is greater than 300 KLOC) [STARSSCAI 95].
- US Army Cleanroom project in the Tank-automotive and Armaments Command at the U.S. Army Picatinny Arsenal. After seven project increments (approximately 90K lines of code), a 4.2:1 productivity increase and a 20:1 return on investment has been documented [Sherer 96a, Sherer 96b]. This project experienced an overall testing error rate (represents all errors found in all testing) of 0.5 errors/KLOC.

In 1995-1996, tools supporting various aspects of the Cleanroom process became commercially available.

Costs and Limitations

Using Cleanroom to accomplish piecemeal, isolated changes to a system not developed using Cleanroom is not considered an effective use of this technology. Training is required and commercially available. Available courses range from overviews to a detailed focus on particular aspects of Cleanroom. For some training classes, it is most productive if software managers and technical staff take the training together. Managers need a thorough understanding of Cleanroom imperatives, and a core group of practitioners needs sufficient orientation in Cleanroom practices to be able to adapt the process to the local environment (this includes establishing a local design language, local verification standards, etc.).

Complementary Technologies

Cleanroom and object-oriented methods. A study/analysis of Cleanroom and three major object-oriented methods: Booch, Objectory, and Shlaer-Mellor (see Object-Oriented Analysis), found that combining object-oriented methods (known for their focus on reusability) with Cleanroom (with its emphasis on rigor, formalisms, and reliability) can define a process capable of producing results that are not only reusable, but also predictable and of high quality. Thus object-oriented methods can be used for front-end domain analysis and Cleanroom can be used for life-cycle application engineering [Ett 96].

Cleanroom and the Capability Maturity Model (CMM). A cleanroom Reference Model [Linger 96b] (a set of Cleanroom Processes for software management, specification, design, and certification) and a detailed mapping of Cleanroom to the CMM for Software [Linger 96a] have been created. The mapping shows that Cleanroom and the CMM [Paulk 93] are fully compatible and mutually reinforcing.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Cleanroom Software Engineering
Application category	Detailed Design (AP.1.3.5), Component Testing (AP.1.4.3.5), System Testing (AP.1.5.3.1), Performance Testing (AP.1.5.3.5), Reengineering (AP.1.9.5)

Quality measures category	Correctness (QM.1.3), Reliability (QM.2.1.2), Understandability (QM.3.2), Availability (QM.2.1.1), Maintainability (QM.3.1)
Computing reviews category	Software Engineering Design (D.2.10)

References and Information Sources

[Cleanroom 96] *Cleanroom Tutorial* [online]. Available WWW <URL: http://source.asset.com/stars/loral/cleanroom/tutorial/cleanroom.html> (1996).

[Ett 96] Ett, William & Trammell, Carmen. *A Guide to Integration of Object-Oriented Methods and Cleanroom Software Engineering* [online]. Originally available WWW <URL: http://www.asset.com/stars/loral/cleanroom/oo/guidhome.htm> (1996).

[Hausler 94] Hausler, P. A.; Linger, R. C.; & Trammel, C. J. "Adopting Cleanroom Software Engineering with a Phased Approach." *IBM Systems Journal 33*, 1 (1994): 89-109.

[Linger 94] Linger, R.C. "Cleanroom Process Model." *IEEE Software 11*, 2 (March 1994): 50-58.

[Linger 96a] Linger, R.C.; Paulk, M.C.; & Trammel, C.J. *Cleanroom Software Engineering Implementation of the CMM for Software* (CMU/SEI-96-TR-023). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

[Linger 96b] Linger, R.C. & Trammel, C.J. *Cleanroom Software Engineering Reference Model* (CMU/SEI-96-TR-022). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute, 1996.

[Mills 87] Mills, H.; Dyer, M.; & Linger, R. "Cleanroom Software Engineering." *IEEE Software 4*, 5 (September 1987): 19-25.

[Paulk 93] Paulk, M.; Curtis B.; Chrissis, M.; & Weber, C. *Capability Maturity Model for Software* Version 1.1 (CMU/SEI-96-TR-24, ADA263403). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

[Sherer 96a] Sherer, S. W. *Cleanroom Software Engineering- the Picatinny Experience* [online]. Available WWW <URL: http://software.pica.army.mil/cleanroom/cseweb.html> (1996).

[Sherer 96b] Sherer, S.W.; Kouchakdjian, A.; & Arnold, P.G. "Experience Using Cleanroom Software Engineering." *IEEE Software 13*, 3 (May 1996): 69-76.

[STARSSCAI 95] Air Force/STARS Demonstration Project Home Page [online]. Available WWW <URL: http://www.asset.com/stars/afdemo/home.html> (1995).

Current Author/Maintainer

John Foreman, SEI

External Reviewers

Wayne Sherer, US Army Picatinny Arsenal
Dave Ceely, Lockheed Martin, Gaithersburg, MD
Dr. Jesse Poore, President, Software Engineering Technologies (SET)
Rick Linger, SEI

Modifications

27 Oct 97: updated URL for [Ett 96]
20 Jun 97: updated URL for [Ett 96]
10 Jan 97: original

Footnotes

[1] STARS: Software Technology for Adaptable Reliable Systems

---

URL: http://www.sei.cmu.edu/str/descriptions/cleanroom_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Client/Server Software Architectures--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

The term client/server was first used in the 1980s in reference to personal computers (PCs) on a network. The actual client/server model started gaining acceptance in the late 1980s. The client/server software architecture is a versatile, message-based and modular infrastructure that is intended to improve *usability*, *flexibility*, *interoperability*, and *scalability* as compared to centralized, mainframe, time sharing computing.

A client is defined as a requester of services and a server is defined as the provider of services. A single machine can be both a client and a server depending on the software configuration. For details on client/server software architectures see Schussel and Edelstein [Schussel 96, Edelstein 94].

This technology description provides a summary of some common client/server architectures and, for completeness, also summarizes mainframe and file sharing architectures. Detailed descriptions for many of the individual architectures are provided elsewhere in the document.

Technical Detail

Mainframe architecture (not a client/server architecture). With mainframe software architectures all intelligence is within the central host computer. Users interact with the host through a terminal that captures keystrokes and sends that information to the host. Mainframe software architectures are not tied to a hardware platform. User interaction can be done using PCs and UNIX workstations. A limitation of mainframe software architectures is that they do not easily support graphical user interfaces (see Graphical User Interface Builders) or access to multiple databases from geographically dispersed sites. In the last few years, mainframes have found a new use as a server in distributed client/server architectures (see Client/Server Software Architectures) [Edelstein 94].

File sharing architecture (not a client/server architecture). The original PC networks were based on file sharing architectures, where the server downloads files from the shared location to the desktop environment. The requested user

job is then run (including logic and data) in the desktop environment. File sharing architectures work if shared usage is low, update contention is low, and the volume of data to be transferred is low. In the 1990s, PC LAN (local area network) computing changed because the capacity of the file sharing was strained as the number of online user grew (it can only satisfy about 12 users simultaneously) and graphical user interfaces (GUIs) became popular (making mainframe and terminal displays appear out of date). PCs are now being used in client/server architectures [Schussel 96, Edelstein 94].

Client/server architecture. As a result of the limitations of file sharing architectures, the client/server architecture emerged. This approach introduced a database server to replace the file server. Using a relational database management system (DBMS), user queries could be answered directly. The client/server architecture reduced network traffic by providing a query response rather than total file transfer. It improves multi-user updating through a GUI front end to a shared database. In client/server architectures, Remote Procedure Calls (RPCs) or standard query language (SQL) statements are typically used to communicate between the client and server [Schussel 96, Edelstein 94].

The remainder of this write-up provides examples of client/server architectures.

Two tier architectures. With two tier client/server architectures (see Two Tier Software Architectures), the user system interface is usually located in the user's desktop environment and the database management services are usually in a server that is a more powerful machine that services many clients. Processing management is split between the user system interface environment and the database management server environment. The database management server provides stored procedures and triggers. There are a number of software vendors that provide tools to simplify development of applications for the two tier client/server architecture [Schussel 96, Edelstein 94].

The two tier client/server architecture is a good solution for distributed computing when work groups are defined as a dozen to 100 people interacting on a LAN simultaneously. It does have a number of limitations. When the number of users exceeds 100, performance begins to deteriorate. This limitation is a result of the server maintaining a connection via "keep-alive" messages with each client, even when no work is being done. A second limitation of the two tier architecture is that implementation of processing management services using vendor proprietary database procedures restricts flexibility and choice of DBMS for applications. Finally, current implementations of the two tier architecture provide limited flexibility in moving (repartitioning) program functionality from one server to another without manually regenerating procedural code. [Schussel 96, Edelstein 94].

Three tier architectures. The three tier architecture (see Three Tier Software Architectures) (also referred to as the multi-tier architecture) emerged to overcome the limitations of the two tier architecture. In the three tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers. The middle tier can perform queuing,

application execution, and database staging. For example, if the middle tier provides queuing, the client can deliver its request to the middle layer and disengage because the middle tier will access the data and return the answer to the client. In addition the middle layer adds scheduling and prioritization for work in progress. The three tier client/server architecture has been shown to improve performance for groups with a large number of users (in the thousands) and improves flexibility when compared to the two tier approach. Flexibility in partitioning can be a simple as "dragging and dropping" application code modules onto different computers in some three tier architectures. A limitation with three tier architectures is that the development environment is reportedly more difficult to use than the visually-oriented development of two tier applications [Schussel 96, Edelstein 94]. Recently, mainframes have found a new use as servers in three tier architectures (see Mainframe Server Software Architectures).

Three tier architecture with transaction processing monitor technology. The most basic type of three tier architecture has a middle layer consisting of Transaction Processing (TP) monitor technology (see Transaction Processing Monitor Technology). The TP monitor technology is a type of message queuing, transaction scheduling, and prioritization service where the client connects to the TP monitor (middle tier) instead of the database server. The transaction is accepted by the monitor, which queues it and then takes responsibility for managing it to completion, thus freeing up the client. When the capability is provided by third party middleware vendors it is referred to as "TP Heavy" because it can service thousands of users. When it is embedded in the DBMS (and could be considered a two tier architecture), it is referred to as "TP Lite" because experience has shown performance degradation when over 100 clients are connected. TP monitor technology also provides

- the ability to update multiple different DBMSs in a single transaction
- connectivity to a variety of data sources including flat files, non-relational DBMS, and the mainframe
- the ability to attach priorities to transactions
- robust security

Using a three tier client/server architecture with TP monitor technology results in an environment that is considerably more scalable than a two tier architecture with direct client to server connection. For systems with thousands of users, TP monitor technology (not embedded in the DBMS) has been reported as one of the most effective solutions. A limitation to TP monitor technology is that the implementation code is usually written in a lower level language (such as COBOL), and not yet widely available in the popular visual toolsets [Schussel 96].

Three tier with message server. Messaging is another way to implement three tier architectures. Messages are prioritized and processed asynchronously. Messages consist of headers that contain priority information, and the address and identification number. The message server connects to the relational DBMS and other data sources. The difference between TP monitor technology and message server is that the message server architecture focuses on intelligent messages, whereas the TP Monitor environment has the intelligence in the monitor, and treats transactions as dumb data packets. Messaging systems are

good solutions for wireless infrastructures [Schussel 96].

Three tier with an application server. The three tier application server architecture allocates the main body of an application to run on a shared host rather than in the user system interface client environment. The application server does not drive the GUIs; rather it shares business logic, computations, and a data retrieval engine. Advantages are that with less software on the client there is less security to worry about, applications are more scalable, and support and installation costs are less on a single server than maintaining each on a desktop client [Schussel 96]. The application server design should be used when security, scalability, and cost are major considerations [Schussel 96].

Three tier with an ORB architecture. Currently industry is working on developing standards to improve interoperability and determine what the common Object Request Broker (ORB) will be. Developing client/server systems using technologies that support distributed objects holds great pomise, as these technologies support interoperability across languages and platforms, as well as enhancing maintainability and adaptability of the system. There are currently two prominent distributed object technologies:

- Common Object Request Broker Architecture (CORBA)
- COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities).

Industry is working on standards to improve interoperability between CORBA and COM/DCOM. The Object Management Group (OMG) has developed a mapping between CORBA and COM/DCOM that is supported by several products [OMG 96].

Distributed/collaborative enterprise architecture. The distributed/ collaborative enterprise architecture emerged in 1993 (see Distributed/ Collaborative Enterprise Architectures). This software architecture is based on Object Request Broker (ORB) technology, but goes further than the Common Object Request Broker Architecture (CORBA) by using shared, reusable business models (not just objects) on an enterprise-wide scale. The benefit of this architectural approach is that standardized business object models and distributed object computing are combined to give an organization flexibility to improve effectiveness organizationally, operationally, and technologically. An enterprise is defined here as a system comprised of multiple business systems or subsystems. Distributed/collaborative enterprise architectures are limited by a lack of commercially-available object orientation analysis and design method tools that focus on applications [Shelton 93, Adler 95].

Usage Considerations

Client/server architectures are being used throughout industry and the military. They provide a versatile infrastructure that supports insertion of new technology more readily than earlier software designs.

Maturity

Client/server software architectures have been in use since the late 1980s. See individual technology descriptions for more detail.

Costs and Limitations

There a number of tradeoffs that must be made to select the appropriate client/server architecture. These include business strategic planning, and potential growth on the number of users, cost, and the homogeneity of the current and future computational environment.

Dependencies

If a distributed object approach is employed, then the CORBA and/or COM/DCOM technologies should be considered (see Common Object Request Broker Architecture and Component Object Model (COM), DCOM, and Related Capabilities).

Alternatives

Alternatives to client/server architectures would be mainframe or file sharing architectures.

Complementary Technologies

Complementary technologies for client/server architectures are computer-aided software engineering (CASE) tools because they facilitate client/server architectural development, and open systems (see COTS and Open Systems--An Overview) because they facilitate the development of architectures that improve scalability and flexibility.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Client/Server Software Architectures
Application category	Software Architecture Models (AP.2.1.1)

Quality measures category	Usability (QM.2.3) Scalability (QM.4.3) Maintainability (QM.3.1) Interoperability (QM.4.1)

References and Information Sources

[Adler 95]	Adler, R. M. "Distributed Coordination Models for Client/Sever Computing." *Computer 28,* 4 (April 1995): 14-22.
[Dickman 95]	Dickman, A. "Two-Tier Versus Three-Tier Apps." *Informationweek* 553 (November 13, 1995): 74-80.
[Edelstein 94]	Edelstein, Herb. "Unraveling Client/Server Architecture." *DBMS 7,* 5 (May 1994): 34(7).
[Gallaugher 96]	Gallaugher, J. & Ramanathan, S. "Choosing a Client/Server Architecture. A Comparison of Two-Tier and Three-Tier Systems." *Information Systems Management Magazine 13,* 2 (Spring 1996): 7-13.
[Louis 95]	*Louis* [online]. Available WWW <URL: http://www.softis.is> (1995).
[Newell 95]	Newell, D.; Jones, O.; & Machura, M. "Interoperable Object Models for Large Scale Distributed Systems," 30-31. *Proceedings. International Seminar on Client/Server Computing*. La Hulpe, Belgium, October 30-31, 1995. London, England: IEE, 1995.
[OMG 96]	Object Management Group home page [online]. Available WWW <URL: http://www.omg.org> (1996).
[Schussel 96]	Schussel, George. *Client/Server Past, Present, and Future* [online]. Available WWW <URL: http://www.dciexpo.com/geos/> (1995).
[Shelton 93]	Shelton, Robert E. "The Distributed Enterprise (Shared, Reusable Business Models the Next Step in Distributed Object Computing)." *Distributed Computing Monitor 8,* 10 (October 1993): 1.

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Frank Rogers, GTE

Modifications

2 August 97: added reference [OMG 96]
25 June 97: Ed Morris (SEI) updated paragraphs containing information about COM/DCOM
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/clientserver_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Common Management Information Protocol

Software Technology Roadmap

Status

Advanced

Note

We recommend Network Management--An Overview *as prerequisite reading for this technology description.*

Purpose and Origin

Common Management Information Protocol (CMIP) is an Open Systems Interconnection (OSI)[1] -based network management protocol that supports information exchange between network management applications and management agents. CMIP is part of the X.700 (CCITT[2] number for the OSI Management Framework, also designated as ISO/IEC 7498-4[3]) OSI series of management standards. Its design is similar to the Simple Network Management Protocol (SNMP). CMIP was developed and funded by government and corporations to replace and makeup for the deficiencies in SNMP, thus improving the capabilities of network management systems.

Technical Detail

CMIP is a well designed protocol that defines how network management information is exchanged between network management applications and management agents. It uses an ISO reliable connection-oriented transport mechanism and has built in security that supports access control, authorization and security logs. The management information is exchanged between the network management application and management agents thru managed objects. Managed objects are a characteristic of a managed device that can be monitored, modified or controlled and can be used to perform tasks. The network management application can initiate transactions with management agents using the following operations:

- ACTION - Request an action to occur as defined by the managed object.
- CANCEL_GET - Cancel an outstanding GET request.
- CREATE - Create an instance of a managed object.
- DELETE - Delete an instance of a managed object.

- GET - Request the value of a managed object instance.
- SET - Set the value of a managed object instance.

A management agent can initiate a transaction with the network management application using the EVENT_REPORT operation. This operation can be used to send notifications or alarms to the network management application based upon predetermined conditions set by the network management application using the ACTION operation.

CMIP does not specify the functionality of the network management application, it only defines the information exchange mechanism of the managed objects and not how the information is to be used or interpreted.

The major advantages of CMIP over SNMP are [Vallillee 96]:

- CMIP variables not only relay information, but also can be used to perform tasks. This is impossible under SNMP.
- CMIP is a safer system as it has built in security that supports authorization, access control, and security logs.
- CMIP provides powerful capabilities that allow management applications to accomplish more with a single request.
- CMIP provides better reporting of unusual network conditions

The CMIP specification for TCP/IP networks is called CMOT (CMIP Over TCP) and the version for IEEE 802 LAN's is called CMOL (CMIP Over LLC) [Stallings 93].

Usage Considerations

CMIP is widely used in the telecommunication domain and telecommunication devices typically support CMIP. The International Telecommunication Union (ITU)[4] endorses CMIP as the protocol for the management of devices in the Telecommunication Management Network (TMN)[5] standard.

The CMIP protocol is designed to run on the ISO protocol stack [Stallings 93]. However, the technology standard used today in most LAN environments is TCP/IP and most LAN devices only support SNMP. Implementations of CMOT are extremely scarce.

CMIP requires a large amount of system resources, this has resulted in very few implementations. Additionally, CMIP is very complex thus making it difficult to program; therefore skilled personnel with specialized training may be required to deploy, maintain and operate a CMIP based network management system.

Maturity

CMIP was developed over a decade ago; however few implementations exist because of the problems described above in Usage Considerations.

Costs and Limitations

Systems may not be capable of supporting the resource requirements of CMIP and difficulties may exist in the procurement of CMIP software because of limited availability.

Alternatives

SNMP is widely available and is the de facto standard network management protocol; however, it does not provide all of the functionality of CMIP. SNMP deficiencies are discussed in Usage Considerations for SNMP.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Common Management Information Protocol (CMIP)
Application category	Protocols (AP.2.2.3) Network Management (AP.2.2.2)
Quality measures category	Maintainability (QM.3.1) Simplicity (QM.3.2.2) Complexity (QM.3.2.1) Efficiency/Resource Utilization (QM.2.2) Scalability (QM.4.3) Security (QM.2.1.5)
Computing reviews category	Network Operations (C.2.3) Distributed Systems (C.2.4)

References and Information Sources

[Korinko 96]	Korinko, Joe. *CMIP-Common Management Information Protocol* [online]. Available WWW <URL: http://www.rit.edu/~jek0539/icsa750/exam2/ex2pg1.htm> (1996).
[Stallings 93]	Stallings, William. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards.* Reading, MA: Addison-Wesley, 1993.

[Vallillee 96]	Vallillee, Tyler. *SNMP & CMIP: An Introduction To Network Management* [online]. Available WWW <URL: http://www.inforamp.net/~kjvallil/t/snmp.html> (1996).
[X.700 96]	*X.700 and Other Network Management Services* [online]. Available WWW <URL: http://ganges.cs.tcd.ie/4ba2/x700/index.html> (1996).

Current Author/Maintainer

Dan Plakosh, SEI

Modifications

9 February 98: minor modifications

13 May 97 (Original)

Footnotes

[1] The OSI model is a framework for defining communications protocols. It consists of seven layers of protocols that range from low level methods for dealing with a physical communications medium, to high level methods for dealing with the communications needs of user applications. Developed by the International Standards Organization (ISO), specific protocols have been designed to implement the functionality specified by the OSI model.

[2] International Telegraph and Telephone Consultative Committee: This organization is part of the United National International Telecommunications Union (ITU) and is responsible for making technical recommendations about telephone and data communications systems.

[3] **International Organization for Standardization (ISO).** A voluntary, non-treaty organization founded in 1946 which is responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of the 89 member countries, including ANSI for the U.S.
International Electrotechnical Commission (NEC). The international standards and conformity assessment body for all fields of electrotechnology. IEC and ISO technical committees collaborate in fields of mutual interest.

[4] The ITU is an international organization within which governments and the private sector coordinate global telecom networks and services. It also develops standards to facilitate the interconnection of telecommunication systems on a worldwide scale regardless of the type of technology used.

[5] A management architecture framework developed by the International Telecommunication Union (ITU), which provides an environment for interfacing a telecommunication network with computer systems to provide different

management functions at several different levels. The framework allows the management of business information between different components (operations systems, communication equipment, network and computer systems) and provides control of service operations and information flow.

URL: http://www.sei.cmu.edu/str/descriptions/cmip_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Common Object Request Broker Architecture

Software Technology Roadmap

Status

ADVANCED

Note

We recommend Object Request Broker, as prerequisite reading for this technology description.

Purpose and Origin

The Common Object Request Broker Architecture (CORBA) is a *specification* of a standard architecture for object request brokers (ORBs) (see Object Request Broker). A standard architecture allows vendors to develop ORB products that support application *portability* and *interoperability* across different programming languages, hardware platforms, operating systems, and ORB implementations:

"Using a CORBA-compliant ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call, and is responsible for finding an object that can implement the request, passing it the parameters, invoking its method, and returning the results of the invocation. The client does not have to be aware of where the object is located, its programming language, its operating system or any other aspects that are not part of an object's interface" [OMG 96]. The "vision" behind CORBA is that distributed systems are conceived and implemented as distributed objects. The interfaces to these objects are described in a high-level, architecture-neutral specification language that also supports object-oriented design abstraction. When combined with the Object Management Architecture (see Technical Detail), CORBA can result in distributed systems that can be rapidly developed, and can reap the benefits that result from using high-level building blocks provided by CORBA, such as *maintainability* and *adaptability*.

The CORBA specification was developed by the Object Management Group (OMG), an industry group with over six hundred member companies representing computer manufacturers, independent software vendors, and a variety of government and academic organizations [OMG 96]. Thus, CORBA specifies an industry/consortium standard, not a "formal" standard in the IEEE/ANSI/ISO sense of the term. The OMG was established in 1988, and the initial

CORBA specification emerged in 1992. Since then, the CORBA specification has undergone significant revision, with the latest major revision (CORBA v2.0) released in July 1996.

Technical Detail

CORBA ORBs are middleware mechanisms (see Middleware), as are all ORBs. CORBA can be thought of as a generalization of remote procedure call (RPC) that includes a number of refinements of RPC, including:

- a more abstract and powerful interface definition language
- direct support for a variety of object-oriented concepts
- a variety of other improvements and generalizations of the more primitive RPC

CORBA and the Object Management Architecture. It is impossible to understand CORBA without appreciating its role in the Object Management Architecture (OMA), shown in Figure 2. The OMA is itself a specification (actually, a collection of related specifications) that defines a broad range of services for building distributed applications. The OMA goes far beyond RPC in scope and complexity. The distinction between CORBA and the OMA is an important one because many services one might expect to find in a middleware product such as CORBA (e.g., naming, transaction, and asynchronous event management services) are actually specified as services in the OMA. For reference, the OMA reference architecture encompasses both the ORB and remote service/object depicted in Figure 21, Middleware.

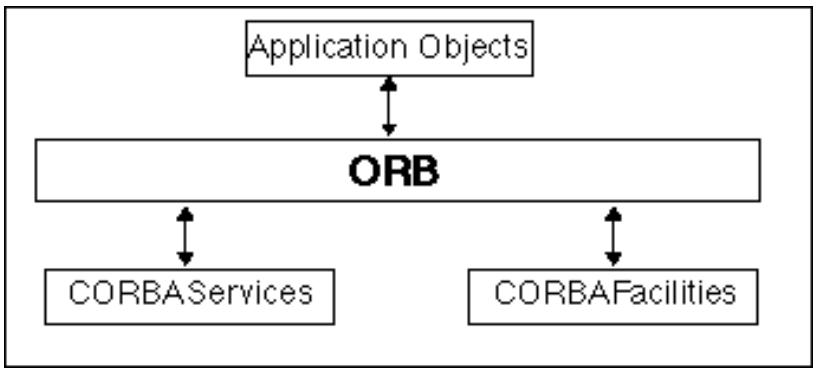


Figure 2: Object Management Architecture

OMA services are partitioned into three categories: CORBAServices, CORBAFacilities, and ApplicationObjects. The ORB (whose details are specified by CORBA) is a communication infrastructure through which applications access these services, and through which objects interact with each other. CORBAServices, CORBAFacilities, and ApplicationObjects define different categories of objects in the OMA; these objects (more accurately object *types*) define a range of functionality needed to support the development of distributed software systems.

- CORBAServices are considered fundamental to building non-trivial distributed applications. These services currently include asynchronous

event management, transactions, persistence, externalization, concurrency, naming, relationships, and lifecycle. Table 1 summarizes the purpose of each of these services.

- CORBAFacilities may be useful for distributed applications in some settings, but are not considered as universally applicable as CORBAServices. These "facilities" include: user interface, information management, system management, task management, and a variety of "vertical market" facilities in domains such as manufacturing, distributed simulation, and accounting.
- Application Objects provide services that are particular to an application or class of applications. These are not (currently) a topic for standardization within the OMA, but are usually included in the OMA reference model for completeness, i.e., objects are either application-specific, support common facilities, or are basic services.

Table 1: Overview of CORBA Services

Naming Service	Provides the ability to bind a name to an object. Similar to other forms of directory service.
Event Service	Supports asynchronous message-based communication among objects. Supports chaining of event channels, and a variety of producer/consumer roles.
Lifecycle Service	Defines conventions for creating, deleting, copying and moving objects.
Persistence Service	Provides a means for retaining and managing the persistent state of objects.
Transaction Service	Supports multiple transaction models, including mandatory "flat" and optional "nested" transactions.
Concurrency Service	Supports concurrent, coordinated access to objects from multiple clients.
Relationship Service	Supports the specification, creation and maintenance of relationships among objects.
Externalization Service	Defines protocols and conventions for externalizing and internalizing objects across processes and across ORBs.

CORBA in detail. Figure 3 depicts most of the basic components and interfaces defined by CORBA. This figure is an expansion of the ORB component of the

OMA depicted in Figure 2.

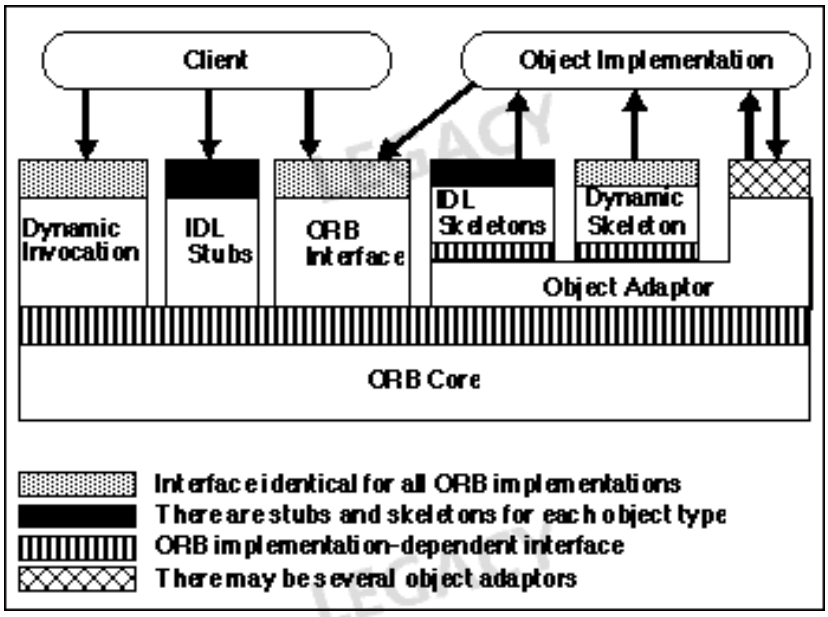


Figure 3: Structure of CORBA Interfaces

One element (not depicted in Figure 2) that is crucial to the understanding of CORBA is the interface definition language (IDL) processor. All objects are defined in CORBA (actually, in the OMA) using IDL. IDL is an object-oriented interface definition formalism that has some syntactic similarities with C++. Unlike C++, IDL can only define interfaces; it is not possible to specify behavior in IDL. Language mappings are defined from IDL to C, C++, Ada95, and Smalltalk80.

An important point to note is that CORBA specifies that clients and object implementations can be written in different programming languages and execute on different computer hardware architectures and different operating systems, and that clients and object implementations can not detect any of these details about each other. Put another way, the IDL interface completely defines the interface between clients and objects; all other details about objects (such as their implementation language and location) can be made "transparent."

Table 2 summarizes the components of CORBA and their functional role.

Table 2: Components of the CORBA Specification

ORB Core	The CORBA runtime infrastructure. The interface to the ORB Core is not defined by CORBA, and will be vendor proprietary.
ORB Interface	A standard interface (defined in IDL) to functions provided by all CORBA- compliant ORBs.

IDL Stubs	Generated by the IDL processor for each interface defined in IDL. Stubs hide the low-level networking details of object communication from the client, while presenting a high-level, object type-specific application programming interface (API).
Dynamic Invocation Interface (DII)	An alternative to stubs for clients to access objects. While stubs provide an object type-specific API, DII provides a generic mechanism for constructing requests at run time (hence "dynamic invocation"). An interface repository (another CORBA component not illustrated in Figure 2) allows some measure of type checking to ensure that a target object can support the request made by the client.
Object Adaptor	Provides extensibility of CORBA- compliant ORBs to integrate alternative object technologies into the OMA. For example, adaptors may be developed to allow remote access to objects that are stored in an object-oriented database. Each CORBA-compliant ORB must support a specific object adaptor called the Basic Object Adaptor (BOA) (not illustrated in Figure 2). The BOA defines a standard API implemented by all ORBs.
IDL Skeletons	The server-side (or object implementation-side) analogue of IDL stubs. IDL skeletons receive requests for services from the object adaptor, and call the appropriate operations in the object implementation.
Dynamic Skeleton Interface (DSI)	The server-side (or object implementation-side) analogue of the DII. While IDL skeletons invoke specific operations in the object implementation, DSI defers this processing to the object implementation. This is useful for developing bridges and other mechanisms to support inter-ORB interoperation.

Usage Considerations

Compliance. As noted, CORBA is a specification, not an implementation. Therefore, the question of compliance is important: How does a consumer know if a product is CORBA-compliant, and, if so, what does that mean? CORBA compliance is defined by the OMG:

"The minimum required for a CORBA-compliant system is adherence to the specifications in CORBA Core and one mapping" [CORBA 96] where "mapping" refers to a mapping from IDL to a programming language (C, C++ or Smalltalk80; Ada95 is specified but has not been formally adopted by the OMG at the time of this writing). The CORBA Core (*not* the same as the ORB Core denoted in Figure 3 and Table 2) is defined for compliance as including the following:

- the interfaces to all of the elements depicted in Figure 3
- interfaces to the interface repository (not shown in Figure 3)
- a definition of IDL syntax and semantics
- the definition of the object model that underlies CORBA (e.g., what is an object, how is it defined, where do they come from)

Significantly, the CORBA Core does *not* include CORBA interoperability, nor does it include interworking, the term used to describe how CORBA is intended to work with Microsoft's COM (see Component Object Model (COM), DCOM, and Related Capabilities). A separate but related point is that CORBA ORBs need not provide implementations of any OMA services.

There are as yet no defined test suites for assessing CORBA compliance. Users must evaluate vendor claims on face value, and assess the likelihood of vendor compliance based upon a variety of imponderables, such as the role played by the vendor in the OMG; vendor market share; and press releases and testimonials. Hands-on evaluation of ORB products is an absolute necessity. However, given the lack of a predefined compliance test suite, the complexity of the CORBA specification (see next topic), and the variability of vendor implementation choices, even this will be inadequate to fully assess "compliance."

Although not concerned with compliance testing in a formal sense, one organization has developed an operational testbed for demonstrating ORB interoperability [CORBANet 96]. It is conceivable that other similar centers may be developed that address different aspects of CORBA (e.g., real time, security), or that do formal compliance testing. However, no such centers exist at the time of this writing.

Complexity. CORBA is a complex specification, and considerable effort may be required to develop expertise in its use. A number of factors compound the inherent complexity of the CORBA specification.

- While CORBA defines a standard, there is great latitude in many of the implementation details- ORBs developed by different vendors may have significantly different features and capabilities. Thus, users must learn a specification, the way vendors implement the specification, and their value-added features (which are often necessary to make a CORBA

product usable).

- While CORBA makes the development of distributed applications easier than with previous technologies, this ease of use may be deceptive: The difficult issues involved in designing robust distributed systems still remain (e.g., performance prediction and analysis, failure mode analysis, consistency and caching, and security).
- Facility with CORBA may require deep expertise in related technologies, such as distributed systems design, distributed and multi-threaded programming and debugging; inter-networking; object-oriented design, analysis, and programming. In particular, expertise in object-oriented technology may require a substantial change in engineering practice, with all the technology transition issues that implies (see The Technology Adoption Challenge).

Stability. CORBA (and the OMA) represent a classical model of distributed computing, despite the addition of object-oriented abstraction. Recent advances in distributed computing have altered the landscape CORBA occupies. Specifically, the recent emergence of mobile objects via Java (see Java), and the connection of Java with "web browser" technologies has muddied the waters concerning the role of CORBA in future distributed systems. CORBA vendors are responding by supporting the development of "ORBlets", i.e., Java applets that invoke the services of remote CORBA objects. However, recent additions to Java support remote object invocation directly in a native Java form. The upshot is that, at the time of this writing, there is great instability in the distributed object technology marketplace.

Industry standards such as CORBA have the advantage of flexibility in response to changes in market conditions and technology advances (in comparison, formal standards bodies move much more slowly). On the other hand, changes to the CORBA specifications- while technically justified- have resulted in unstable ORB implementations. For example, CORBA v2.0, released in July 1995 with revisions in July 1996, introduced features to support interoperation among different vendor ORBs. These features are not yet universally available in all CORBA ORBs, and those ORBs that implement these features do so in uneven ways. Although the situation regarding interoperation among CORBA ORBs is improving, instability of implementations is the price paid for flexibility and evolvability of specification.

The OMA is also evolving, and different aspects are at different maturity levels. For instance, CORBAFacilities defines more of a framework for desired services than a specification suitable for implementation. The more fundamental CORBAServices, while better defined, are not rigorously defined; a potential consequence is that different vendor implementations of these services may differ widely both in performance and in semantics. The consequence is particularly troubling in light of the new interoperability features; prior to inter-ORB interoperability the lack of uniformity among CORBAServices implementations would not have been an issue.

Maturity

A large and growing number of implementations of CORBA are available in the marketplace, including implementations from most major computer

manufacturers and independent software vendors. See Object Request Broker for a listing of available CORBA-compliant ORBs. CORBA ORBs are also being developed by university research and development projects, for example Stanford's Fresco, XeroxPARC's ILU, Cornell's Electra, and others.

At the same time, it must be noted that not all CORBA ORBs are equally mature, nor has the OMA sufficiently matured to support the vision that lies behind CORBA (see Purpose and Origin). While CORBA and OMA products are maturing and are being used in increasingly complex and demanding situations, the specifications and product implementations are not entirely stable. This is in no small way a result of the dynamism of distributed object technology and middleware in general and is no particular fault of the OMG. Fortunately techniques exist for evaluating technology in the face of such dynamism [Wallace 96, Brown 96].

Costs and Limitations

Costs and limitations include the following:

- *Real time.* CORBA v2.0 does not address real-time issues.
- *Programming language support.* IDL is a "least-common denominator" language. It does not fully exploit the capabilities of programming languages to which it is mapped, especially where the definition of abstract types is concerned.
- *Pricing and licensing*. The price of ORBs varies greatly, from a few hundred to several thousand dollars. Licensing schemes also vary.
- *Training.* Training is essential for the already experienced programmer: five days of hands-on training for CORBA programming fundamentals is suggested [Mowbray 93].
- *Security.* CORBA specifies only a minimal range of security mechanisms; more ambitious and comprehensive mechanisms have not yet been adopted by the OMG. Deng discusses the potential integration of security into CORBA-based systems [Deng 95].

Dependencies

Dependencies include the following:

- TCP/IP is needed to support the CORBA-defined inter-ORB interoperability protocol (IIOP).
- Most commercial CORBA ORBs rely on C++ as the principal client and server programming environment. Java-specific ORBs are also emerging.

Alternatives

Alternatives include the following:

- The Open Group's Distributed Computing Environment (DCE) is sometimes cited as an alternative "open" specification for distributed

computing (see Distributed Computing Environment).

- Where openness is not a concern and PC platforms are dominant, Microsoft's COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities) may be suitable alternatives.
- Other middleware technologies may be appropriate in different settings (e.g., message-oriented middleware (see Message-Oriented Middleware)).

Complementary Technologies

Complementary technologies include the following:

- Java and/or web browsers can be used in conjunction with CORBA, although precise usage patterns have not yet emerged and are still highly volatile.
- Object-oriented database management systems (OODBMS) vendors are developing object adaptors to support more robust three-tier architecture (see Three Tier Software Architectures) development using CORBA.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Common Object Request Broker Architecture
Application category	Client/Server (AP.2.1.2.1), Client/Server Communication (AP.2.2.1)
Quality measures category	Maintainability (QM.3.1), Interoperability (QM.4.1), Portability (QM.4.2), Scalability (QM.4.3), Reusability (QM.4.4)
Computing reviews category	Distributed Systems (C.2.4), Object-Oriented Programming (D.1.5)

References and Information Sources

[Baker 94]	Baker, S. "CORBA Implementation Issues." *IEEE Colloquium on Distributed Object Management Digest 1994 7* (January 1994): 24-25.
[Brando 96]	Brando, T. "Comparing CORBA & DCE." *Object Magazine 6*, 1 (March 1996): 52-7.
[Brown 96]	Brown, A. & Wallnau, K. "A Framework for Evaluating Software Technology." *IEEE Software 13*, 5 (September 1996): 39-49.

[CORBA 96] *The Common Object Request Broker: Architecture and Specification*, Version 2.0. Framingham, MA: Object Management Group, 1996. Also available [online] WWW <URL: http://www.omg.org> (1996).

[CORBANet 96] Distributed Software Technology Center Home Page [online]. Available WWW <URL: http://corbanet.dstc.edu.au> (1996).

[Deng 95] Deng, R.H., et al. "Integrating Security in CORBA-Based Object Architectures," 50-61. *Proceedings of the 1995 IEEE Symposium on Security and Privacy*. Oakland, CA, May 8-10, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Foody 96] Foody, M.A. "OLE and COM vs. CORBA." *UNIX Review 14*, 4. (April 1996): 43-45.

[Jell 95] Jell, T. & Stal, M. "Comparing, Contrasting, and Interweaving CORBA and OLE," 140-144. *Object Expo Europe 1995*. London, UK, September 25-29, 1995. Newdigate, UK: SIGS Conferences, 1995.

[Kain 94] Kain, J.B. "An Overview of OMG's CORBA," 131-134. *Proceedings of OBJECT EXPO `94*. New York, NY, June 6-10, 1994. New York, NY: SIGS Publications, 1994.

[Mowbray 93] Mowbray, T.J. & Brando, T. "Interoperability and CORBA-Based Open Systems." *Object Magazine 3*, 3 (September/ October 1993): 50-4.

[OMG 96] Object Management Group home page [online]. Available WWW <URL: http://www.omg.org> (1996).

[Roy 95] Roy, Mark & Ewald, Alan. "Distributed Object Interoperability." *Object Magazine 5*, 1 (March/April 1995): 18.

[Steinke 95] Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

[Tibbets 95] Tibbets, Fred. "CORBA: A Common Touch for Distributed Applications." *Data Comm Magazine 24*, 7 (May 1995): 71-75.

[Wallace 96] Wallnau, Kurt & Wallace, Evan. "A Situated Evaluation of the Object Management Group's (OMG) Object Management Architecture (OMA)," 168-178. *Proceedings of the OOPSLA'96*. San Jose, CA, October 6-10, 1996. New York, NY: ACM, 1996. Presentation available [online] FTP. <URL: ftp://ftp.sei.cmu.edu/pub/corba/OOPSLA/present> (1996).

[Watson 96] Watson, A. "The OMG After CORBA 2." *Object Magazine 6*, 1 (March 1996): 58-60.

Current Author/Maintainer

Kurt Wallnau, SEI

External Reviewers

Dave Carney, SEI
Ed Morris, SEI

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/corba_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Component-Based Software Development / COTS Integration

Software Technology Roadmap

Status

Advanced

Note

We recommend COTS and Open Systems--An Overview as prerequisite reading for this technology description.

Purpose and Origin

Component-based software development (CBSD) focuses on building large software systems by integrating previously-existing software components. By enhancing the *flexibility* and *maintainability* of systems, this approach can potentially be used to reduce software development costs, assemble systems rapidly, and reduce the spiraling maintenance burden associated with the support and upgrade of large systems. At the foundation of this approach is the assumption that certain parts of large software systems reappear with sufficient regularity that common parts should be written once, rather than many times, and that common systems should be assembled through reuse rather than rewritten over and over. CBSD embodies the "buy, don't build" philosophy espoused by Fred Brooks [Brooks 87]. CBSD is also referred to as component-based software engineering (CBSE) [Brown 96a, Brown 96b].

Component-based systems encompass both commercial-off-the-shelf (COTS) products and components acquired through other means, such as nondevelopmental items (NDIs).[1] Developing component-based systems is becoming feasible due to the following:

- the increase in the quality and variety of COTS products
- economic pressures to reduce system development and maintenance costs
- the emergence of component integration technology (see Object Request Broker)
- the increasing amount of existing software in organizations that can be reused in new systems

CBSD shifts the development emphasis from programming software to composing software systems [Clements 95].

Technical Detail

In CBSD, the notion of building a system by writing code has been replaced with building a system by assembling and integrating existing software components. In contrast to traditional development, where system integration is often the tail end of an implementation effort, component integration is the centerpiece of the approach; thus, implementation has given way to integration as the focus of system construction. Because of this, integrability is a key consideration in the decision whether to acquire, reuse, or build the components.

As shown in Figure 4, four major activities characterize the component-based development approach; these have been adapted from Brown [Brown 96b]:

- component qualification (sometimes referred to as suitability testing)
- component adaptation
- assembling components into systems
- system evolution

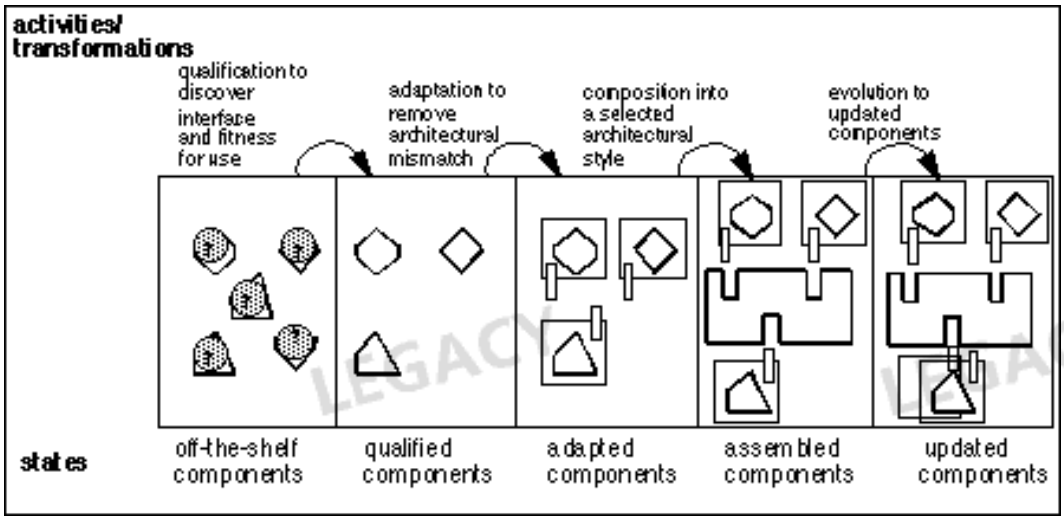


Figure 4: Activities of the Component-Based Development Approach

Each activity is discussed in more detail in the following paragraphs.

Component qualification. Component qualification is a process of determining "fitness for use" of previously-developed components that are being applied in a new system context. Component qualification is also a process for selecting components when a marketplace of competing products exists. Qualification of a component can also extend to include qualification of the development process used to create and maintain it (for example, ensuring algorithms have been validated, and that rigorous code inspections have taken place). This is most obvious in safety-critical applications, but can also reduce some of the attraction

of using preexisting components.

There are two phases of component qualification: *discovery* and *evaluation*. In the discovery phase, the properties of a component are identified. Such priorities include component functionality (what services are provided) and other aspects of a component's interface (such as the use of standards). These properties also include quality aspects that are more difficult to isolate, such as component reliability, predictability, and usability. In some circumstances, it is also reasonable to discover "non-technical" component properties, such as the vendor's market share, past business performance, and process maturity of the component developer's organization. Discovery is a difficult and ill-defined process, with much of the needed information being difficult to quantify and, in some cases, difficult to obtain.

There are some relatively mature evaluation techniques for selecting from among a group of peer products. For example, the International Standards Organization (ISO) describes general criteria for product evaluation [ISO 91] while others describe techniques that take into account the needs of particular application domains [IEEE 93, Poston 92]. These evaluation approaches typically involve a combination of paper-based studies of the components, discussion with other users of those components, and hands-on benchmarking and prototyping.

One recent trend is toward a "product line" approach that is based on a reusable set of components that appear in a range of software products. This approach assumes that similar systems (e.g., most radar systems) have a similar software architecture and that a majority of the required functionality is the same from one product to the next. (See Domain Engineering and Domain Analysis for further details on techniques to help determine similarity). The common functionality can therefore be provided by the same set of components, thus simplifying the development and maintenance life cycle. Results of implementing this approach can be seen in two different efforts [Lettes 96, STARSSCAI 95].

Component adaptation. Because individual components are written to meet different requirements, and are based on differing assumptions about their context, components often must be adapted when used in a new system. Components must be adapted based on rules that ensure conflicts among components are minimized. The degree to which a component's internal structure is accessible suggests different approaches to adaptation [Valetto 95]:

- *white box,* where access to source code allows a component to be significantly rewritten to operate with other components
- *grey box,* where source code of a component is not modified but the component provides its own extension language or application programming interface (API) (see Application Programming Interface)
- *black box,* where only a binary executable form of the component is available and there is no extension language or API

Each of these adaptation approaches has its own positives and negatives;

however, white box approaches, because they modify source code, can result in serious maintenance and evolution concerns in the long term. Wrapping, bridging, and mediating are specific programming techniques used to adapt grey- and black-box components.

Assembling components into systems. Components must be integrated through some well-defined infrastructure. This infrastructure provides the binding that forms a system from the disparate components. For example, in developing systems from COTS components, several architectural styles are possible:

- *database*, in which centralized control of all operational data is the key to all information sharing among components in the system
- *blackboard*, in which data sharing among components is opportunistic, involving reduced levels of system overhead
- *message bus*, in which components have separate data stores coordinated through messages announcing changes among components
- *object request broker (ORB) mediated*, in which the ORB technology (see Object Request Broker) provides mechanisms for language-independent interface definition and object location and activation

Each style has its own particular strengths and weaknesses. Currently, most active research and product development is taking place in object request brokers (ORBs) conforming to the Common Object Request Broker Architecture (CORBA).[2]

System evolution. At first glance, component-based systems may seem relatively easy to evolve and upgrade since components are the unit of change. To repair an error, an updated component is swapped for its defective equivalent, treating components as plug-replaceable units. Similarly, when additional functionality is required, it is embodied in a new component that is added to the system.

However, this is a highly simplistic (and optimistic) view of system evolution. Replacement of one component with another is often a time-consuming and arduous task since the new component will never be identical to its predecessor and must be thoroughly tested, both in isolation and in combination with the rest of the system. Wrappers must typically be rewritten, and side-effects from changes must be found and assessed. One possible approach to remedying this problem is Simplex (see Simplex Architecture).

Usage Considerations

Several items need to be considered when implementing component-based systems:

Short-term considerations

- *Development process*. An organization's software development process and philosophy may need to change. System integration can no longer be at the end of the implementation phase, but must be planned early and be continually managed throughout the development process. It is also recommended that as tradeoffs are being made among components during the development process, the rationale used in making the tradeoff decisions should be recorded and then evaluated in the final product [Brown 96b].
- *Planning*. Many of the problems encountered when integrating COTS components cannot be determined before integration begins. Thus, estimating development schedules and resource requirements is extremely difficult [Vigder 96].
- *Requirements*. When using a preexisting component, the component has been written to a preexisting, and possibly unknown, set of requirements. In the best case, these requirements will be very general, and the system to be built will have requirements that either conform or can be made to conform to the preexisting general requirements. In the worst case, the component will have been written to requirements that conflict in some critical manner with those of the new system, and the system designer must choose whether using the existing component is viable at all.
- *Architecture*. The selection of standards and components needs to have a sound architectural foundation, as this becomes the foundation for system evolution. This is especially important when migrating from a legacy system to a component-based system.
- *Standards*. If an organization chooses to use the component-based system development approach and it also has the goal of making a system open, then interface standards need to come into play as criteria for component qualification. The degree to which a software component meets certain standards can greatly influence the interoperability and portability of a system. Reference the COTS and Open Systems--An Overview description for further discussion.
- *Reuse of existing components*. Component-based system development spotlights reusable components. However, even though organizations have increasing amounts of existing software that can be reused, most often some amount of reengineering must be accomplished on those components before they can be adapted to new systems.
- *Component qualification.* While there are several efforts focusing on component qualification, there is little agreement on which quality attributes or measures of a component are critical to its use in a component-based system. A useful work that begins to address this issue is "SAAM: A Method for Analyzing the Properties of Software Architecture" [Abowd 94]. Another technique addresses the complexity of component selection and provides a decision framework that supports multi-variable component selection analysis [Kontio 96]. Other approaches, such as the qualification process defined by the US Air Force PRISM program, emphasize "fitness for use" within specific application domains, as well as the primacy of integrability of components [PRISM 96]. Another effort is Product Line Asset Support [PLAS 96].

Long-term considerations

- *External dependencies/vendor-driven upgrade problem.* An organization loses a certain amount of autonomy and acquires additional dependencies when integrating COTS components. COTS component producers frequently upgrade their components based on error reports, perceived market needs and competition, and product aesthetics. DoD systems typically change at a much slower rate and have very long lifetimes. An organization must juggle its new functionality requirements to accommodate the direction in which a COTS product may be going. New component releases require a decision from the component-based system developer/integrator on whether to include the new component in the system. To answer "yes" implies facing an undetermined amount of rewriting of wrapper code and system testing. To answer "no" implies relying on older versions of components that may be behind the current state-of-the-art and may not be adequately supported by the COTS supplier. This is why the component-based system approach is sometimes considered a risk transfer and not a risk reduction approach.
- *System evolution/technology insertion.* System evolution is not a simple plug-and-play approach. Replacing one component often has rippling affects throughout the system, especially when many of the components in the system are black box components; the system's integrator does not know the details of how a component is built or will react in an interdependent environment. Further complicating the situation is that new versions of a component often require enhanced versions of other components, or in some cases may be incompatible with existing components.

  Over the long-term life of a system, additional challenges arise, including inserting COTS components that correspond to new functionality (for example, changing to a completely new communications approach) and "consolidation engineering" wherein several components may be replaced by one "integrated" component. In such situations, maintaining external interface compatibility is very important, but internal data flows that previously existed must also be analyzed to determine if they are still needed.

Maturity

To date, the commercial components available and reliable enough for operational systems, and whose interfaces are well-enough understood, have primarily been operating systems, databases, email and messaging systems, office automation software (e.g., calendars, word processors, spreadsheets), and Graphical User Interface Builders. The number of available components continues to grow and quality and applicability continue to improve. As such, most successful applications have been in the AIS/MIS and C3I areas, with rather limited success in applications having real-time performance, safety, and security requirements. Indeed, in spite of the possible savings, using COTS components to build safety-critical systems where reliability, availability, predictability, and security are essential is frequently too risky [Brown 96b]. An organization will typically not have complete understanding or control of the COTS components and their development.

Examples of apparently successful integration of COTS into operational systems include the following

- Deep Space Network Program at the NASA Jet Propulsion Laboratory [NASA 96a]
- Lewis Mission at NASA's Goddard Space Center [NASA 96b]
- Boeing's new 777 aircraft with 4 million lines of COTS software [Vidger 96]
- Air Force Space and Missile System Center's telemetry, tracking, and control (TT&C) system called the Center for Research Support (CERES) [Monfort 96]

In addition to the increasing availability of components applicable to certain domains, understanding of the issues and technologies required to expand CBSD practice is also growing, although significant work remains. Various new technical developments and products, including Common Object Request Broker Architecture and Component Object Model (COM), DCOM, and Related Capabilities [Vidger 96] and changes in acquisition and business practices should further stimulate the move to CBSD.

Costs and Limitations

It is widely assumed that the component-based software development approach, particularly in the sense of using COTS components, will be significantly less costly (i.e., shorter development cycles and lower development costs) than the traditional method of building systems "from scratch." In the case of using such components as databases and operating systems, this is almost certainly true. However, there is little data available concerning the relative costs of using the component-based approach and, as indicated in Usage Considerations, there are a number of new issues that must be considered.

In addition, if integrating COTS components, an additional system development and maintenance cost will be to negotiate, manage, and track licenses to ensure uninterrupted operation of the system. For example, a license expiring in the middle of a mission might have disastrous consequences.

Dependencies

Adapting preexisting components to a system requires techniques such as Application Programming Interface, wrapping, bridging, or mediating, as well as an increased understanding of architectural interactions and components' properties.

Alternatives

The alternatives include using preexisting components or creating the entire system as a new item.

Complementary Technologies

The advantages of using the CBSD/COTS integration approach can be greatly enhanced by coupling the approach with open systems (see COTS and Open Systems--An Overview).

Domain Engineering and Domain Analysis aid in identifying common functions and data among a domain of systems which in turn identifies possible reusable components.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Component-Based Software Development/ COTS Integration
Application category	System Allocation (AP.1.2.1), Select or Develop Algorithms (AP.1.3.4), Plan and Perform Integration (AP.1.4.4), Reengineering (AP.1.9.5)
Quality measures category	Maintainability (QM.3.1)
Computing reviews category	Software Engineering Design (D.2.10), Software Engineering Miscellaneous (D.2.m)

References and Information Sources

[Abowd 94] Abowd, G., et al. "SAAM: A Method for Analyzing the Properties of Software Architecture," 81-90. *Proceedings of the 16th International Conference on Software Engineering.* Sorrento, Italy, May 16-21, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[Brooks 87] Brooks, F. P. Jr. "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer 20,* 4 (April 1987): 10-9.

[Brown 96a] Brown, Alan W. "Preface: Foundations for Component-Based Software Engineering," vii-x. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute.* Los Alamitos, CA: IEEE Computer Society Press, 1996.

[Brown 96b] Brown, Alan W. & Wallnau, Kurt C. "Engineering of Component-Based Systems," 7-15. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute.* Los Alamitos, CA: IEEE Computer Society Press, 1996.

[Clements 95] Clements, Paul C. "From Subroutines to Subsystems: Component-Based Software Development," 3-6. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute.* Los Alamitos, CA: IEEE Computer Society Press, 1996.

[IEEE 93] *IEEE Recommended Practice on the Selection and Evaluation of CASE Tools* (IEEE Std. 1209-1992). New York, NY: Institute of Electrical and Electronics Engineers, 1993.

[ISO 91] *Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use*. Geneve, Switzerland: International Standards Organization/International Electrochemical Commission, 1991.

[Kontio 96] Kontio, J. "A Case Study in Applying a Systematic Method for COTS Selection," 201-209. *Proceedings of the 18th International Conference on Software Engineering.* Berlin, Germany, March 25-30, 1996. Los Alamitos, CA: IEEE Computer Society Press, 1996.

[Lettes 96] Lettes, Judith A. & Wilson, John. *Army STARS Demonstration Project Experience Report* (STARS-VC-A011/003/02). Manassas, VA: Loral Defense Systems-East, 1996.

[Monfort 96] Monfort, Lt. Col. Ralph D. "Lessons Learned in the Development and Integration of a COTS-Based Satellite TT&C System." *33rd Space Congress*. Cocoa Beach, FL, April 23-26, 1996.

[NASA 96a] *COTS Based Development* [online]. Available WWW <URL: http://www-isds.jpl.nasa.gov/cwo/cwo_23/handbook/Dsnswdhb.htm> (1996).

[NASA 96b] *Create Mechanisms/Incentives for Reuse and COTS Use* [online]. Available WWW <URL: http://bolero.gsfc.nasa.gov/c600/workshops/sswssp4b.htm> (1996).

[PLAS 96] PLAS [online]. Available WWW <URL: http://www.cards.com/plas> (1996).

[Poston 92] Poston R.M. & Sexton M.P. "Evaluating and Selecting Testing Tools." *IEEE Software 9,* 3 (May 1992): 33-42.

[PRISM 96] Portable, Reusable, Integrated Software Modules (PRISM) Program [online]. Available WWW <URL: http://www.cards.com/PRISM/prism_ov.html>(1996).

[STARSSCAI 95] Air Force/STARS Demonstration Project Home Page [online]. Available WWW <URL: http://www.asset.com/stars/afdemo/home.html> (1995).

[Thomas 92] Thomas, I. & Nejmeh. B. "Definitions of Tool Integration for Environments." *IEEE Software 9,* 3 (March 1992): 29-35.

[Valetto 95] Valetto, G. & Kaiser, G.E. "Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments," 40-48. *Proceedings of 7th IEEE International Workshop on CASE.* Toronto, Ontario, Canada, July 10-14, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Vidger 96] Vidger, M.R.; Gentleman, W.M.; & Dean, J. *COTS Software Integration: State-of-the-Art* [online]. Available WWW <URL: http://wwwsel.iit.nrc.ca/abstracts/NRC39198.abs> (1996).

Current Author/Maintainer

Capt Gary Haines, AFMC SSSG
David Carney, SEI
John Foreman, SEI

External Reviewers

Paul Kogut, Lockheed Martin, Paoli, PA
Ed Morris, SEI
Tricia Oberndorf, SEI
Kurt Wallnau, SEI

Modifications

7 Oct 97: minor edits
20 Jun 97: updated URL for [NASA 96a]
10 Jan 97 (original)

Footnotes

[1] See the definition of NDI in COTS and Open Systems - An Overview.

[2] From Wallnau, K. & Wallace, E. *A Robust Evaluation of the Object Management Architecture: A Focused Case Study in Legacy Systems Migration.*

Submitted for publication to OOPLSA'96.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/cbsd_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Component Object Model (COM), DCOM, and Related Capabilities

Software Technology Roadmap

Status

Advanced

Note

We recommend Object Request Broker, Remote Procedure Call, *and* Component-Based Software Development/COTS Integration, *as prerequisite readings for this technology description.*

Purpose and Origin

COM [COM 95] refers to both a specification and implementation developed by Microsoft Corporation which provides a framework for integrating components. This framework supports *interoperability* and *reusability* of distributed objects by allowing developers to build systems by assembling reusable components from different vendors which communicate via COM. By applying COM to build systems of preexisting components, developers hope to reap benefits of *maintainability* and *adaptability*.

COM defines an application programming interface (API) to allow for the creation of components for use in integrating custom applications or to allow diverse components to interact. However, in order to interact, components must adhere to a binary structure specified by Microsoft. As long as components adhere to this binary structure, components written in different languages can interoperate.

Distributed COM [DCOM 97] is an extension to COM that allows network-based component interaction. While COM processes can run on the same machine but in different address spaces, the DCOM extension allows processes to be spread across a network. With DCOM, components operating on a variety of platforms can interact, as long as DCOM is available within the environment.

It is best to consider COM and DCOM as a single technology that provides a range of services for component interaction, from services promoting component integration on a single platform, to component interaction across heterogeneous networks. In fact, COM and its DCOM extensions are merged into a single runtime. This single runtime provides both local and remote access.

While COM and DCOM represent "low-level" technology that allows components to interact, OLE [Brockschmidt 95], ActiveX [Active 97] and MTS [Harmon 99] represent higher-level application services that are built on top of COM and DCOM. OLE builds on COM to provide services such as object "linking" and "embedding" that are used in the creation of compound documents (documents generated from multiple tool sources). ActiveX extends the basic capabilities to allow components to be embedded in Web sites. MTS expands COM capabilities with enterprise services such as transaction and security to allow Enterprise Information Systems (EIS) to be built using COM components. COM+ is the evolution of COM.

COM+ integrates MTS services and message queuing into COM, and makes COM programming easier through a closer integration with Microsoft languages as Visual Basic, Visual C++, and J++. COM+ will not only add MTS-like quality of service into every COM+ object, but it will hide some of the complexities in COM coding.

The distinctions among various Microsoft technologies and products are sometimes blurred. Thus, one might read about "OLE technologies" which encompass COM, or "Active Platform" as a full web solution. In this technology description, we focus on the underlying technology represented by COM, DCOM, and COM+.

Technical Detail

COM is a binary compatibility specification and associated implementation that allows clients to invoke services provided by COM-compliant components (COM objects). As shown in Figure 5, services implemented by COM objects are exposed through a set of interfaces that represent the only point of contact between clients and the object.

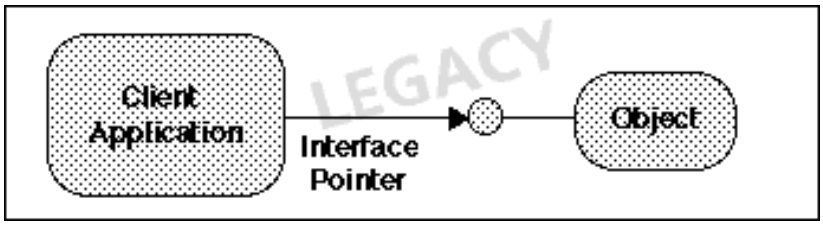


Figure 5: Client Using COM Object Through an Interface Pointer [COM 95]

COM defines a binary structure for the interface between the client and the object. This binary structure provides the basis for interoperability between software components written in arbitrary languages. As long as a compiler can reduce language structures down to this binary representation, the implementation language for clients and COM objects does not matter - the point of contact is the run-time binary representation. Thus, COM objects and clients can be coded in any language that supports Microsoft's COM binary structure.

A COM object can support any number of interfaces. An interface provides a grouped collection of related methods. For example, Figure 6 depicts a COM

object that emulates a clock. IClock, IAlarm and ITimer are the interfaces of the clock object. The IClock interface can provide the appropriate methods (not shown) to allow setting and reading the current time. The IAlarm and ITimer interfaces can supply alarm and stopwatch methods.

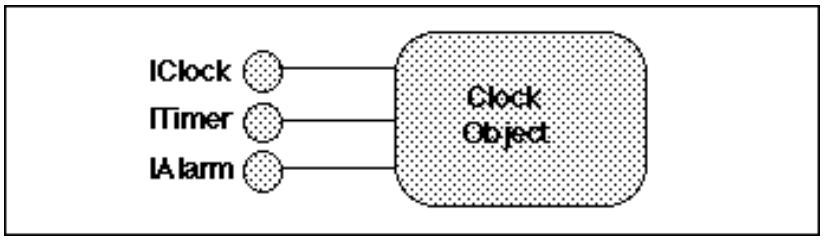


Figure 6: Clock COM object

COM objects and interfaces are specified using Microsoft Interface Definition Language (IDL), an extension of the DCE Interface Definition Language standard (see Distributed Computing Environment). To avoid name collisions, each object and interface must have a unique identifier.

Interfaces are considered logically immutable. Once an interface is defined, it should not be changed-new methods should not be added and existing methods should not be modified. This restriction on the interfaces is not enforced, but it is a rule that component developers should follow. Adhering to this restriction removes the potential for version incompatibility-if an interface never changes, then clients depending on the interface can rely on a consistent set of services. If new functionality has to be added to a component, it can be exposed through a different interface. For our clock example, we can design an enhanced clock COM object supporting the IClock2 interface that inherits from IClock. IClock2 may expose new functionality.

Every COM object runs inside of a server. A single server can support multiple COM objects. As shown in Figure 7, there are three ways in which a client can access COM objects provided by a server:

1. In-process server: The client can link directly to a library containing the server. The client and server execute in the same process. Communication is accomplished through function calls.
2. Local Object Proxy: The client can access a server running in a different process but on the same machine through an inter-process communication mechanism. This mechanism is actually a lightweight Remote Procedure Call (RPC).
3. Remote Object Proxy: The client can access a remote server running on another machine. The network communication between client and server is accomplished through DCE RPC. The mechanism supporting access to remote servers is called DCOM.

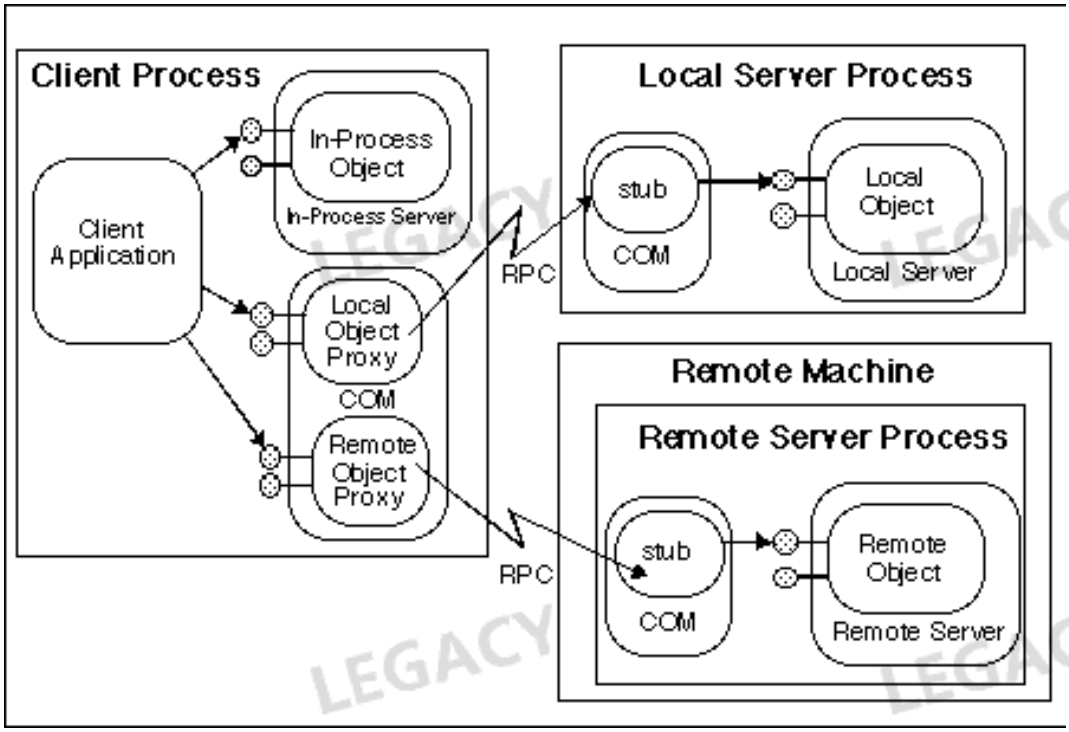


Figure 7: Three Methods for Accessing COM Objects [COM 95]

If the client and server are in the same process, the sharing of data between the two is simple. However, when the server process is separate from the client process, as in a local server or remote server, COM must format and bundle the data in order to share it. This process of preparing the data is called marshalling. Marshalling is accomplished through a "proxy" object and a "stub" object that handle the cross-process communication details for any particular interface (depicted in Figure 8). COM creates the "stub" in the object's server process and has the stub manage the real interface pointer. COM then creates the "proxy" in the client's process, and connects it to the stub. The proxy then supplies the interface pointer to the client.

The client calls the interfaces of the server through the proxy, which marshals the parameters and passes them to the server stub. The stub unmarshals the parameters and makes the actual call inside the server object. When the call completes, the stub marshals return values and passes them to the proxy, which in turn returns them to the client. The same proxy/stub mechanism is used when the client and server are on different machines. However, the internal implementation of marshalling and unmarshalling differs depending on whether the client and server operate on the same machine (COM) or on different machines (DCOM). Given an IDL file, the Microsoft IDL compiler can create default proxy and stub code that performs all necessary marshalling and unmarshalling.

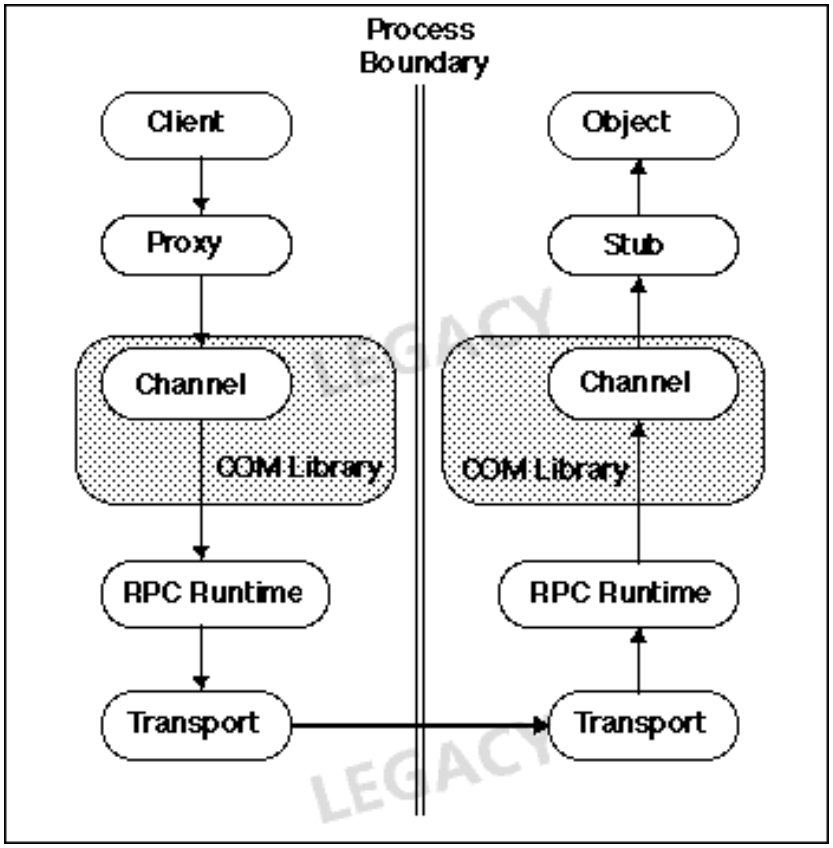


Figure 8: Cross-process communication in COM [COM 95]

All COM objects are registered with a component database. As shown in Figure 9, when a client wishes to create and use a COM object:

1. It invokes the COM API to instantiate a new COM object.
2. COM locates the object implementation and initiates a server process for the object.
3. The server process creates the object, and returns an interface pointer at the object.
4. The client can then interact with the newly instantiated COM object through the interface pointer.

An important aspect in COM is that objects have no identity, i.e. a client can ask for a COM object of some type, but not for a particular object. Every time that COM is asked for a COM object, a new instance is returned. The main advantage of this policy is that COM implementations can pool COM objects and return these pooled objects to requesting clients. Whenever a client has finished using an object the instance is returned to the pool. However, there are mechanisms to simulate identity in COM such as monikers (reviewed later).

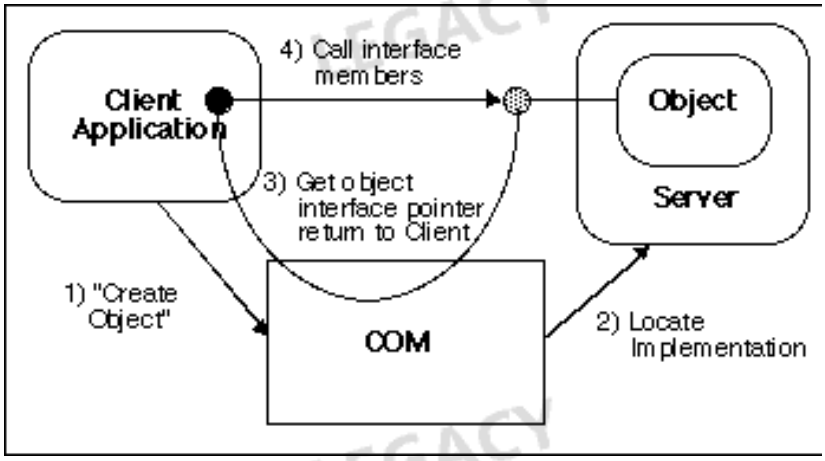


Figure 9: Creating a COM object pointer [COM 95]

COM includes interfaces and API functions that expose operating system services, as well as other mechanisms necessary for a distributed environment (naming, events, etc.). These are sometimes referred to as COM technologies (or services), and are shown in Table 3.

Table 3: COM Technologies

Service	Explanation
Type Information	Some clients need runtime access to type information about COM objects. This type information is generated by the Microsoft IDL compiler and is stored in a type library. COM provides interfaces to navigate the type library.
Structured Storage and Persistence	COM objects need a way to store their data when they are not running. The process of saving data for an object is called making an object persistent. COM supports object persistence through "Structured Storage", which creates an analog of a file system within a file. Individual COM objects can store data within the file, thus providing persistence.
Monikers	Clients often require a way to allow them to connect to the exact same object instance with the exact same state at a later point in time. This support is provided via "monikers". A moniker is a COM object that knows how to create and initialize the content of a single COM object instance. A moniker can be asked to bind to the COM object it represents, such as a COM object residing on specific machine on the network, or a group of cells inside a spreadsheet.

Uniform Data Transfer	COM objects often need to pass data amongst themselves. Uniform Data Transfer provides for data transfers and notifications of data changes between a source called the data object, and something that uses the data, called the consumer object.
Connectable Objects	Some objects require a way to notify clients that an event that has occurred. COM allows such objects to define outgoing interfaces to clients as well as incoming interfaces. The object defines an interface it would like to use (e.g., a notification interface) and the client implements the interface. This enables two-way communication between the client and the component.

COM has enjoyed great industrial support with thousands of ISVs developing COM components and applications. However, COM suffers from some weaknesses that have been recognized by Microsoft and addressed in Component Object Model+, which is the ongoing upgrade of COM.

1. COM is hard to use. Reference counting, Microsoft IDL, Global Unique Identifiers (GUID), etc. require deep knowledge of COM specification from developers.
2. COM is not robust enough for enterprise deployments. Services such as security, transactions, reliable communications, and load balancing are not integrated in COM.

Both issues were partially mitigated by add-ons of COM, complexity by integrated development environments and robustness by MTS. However, to further address those problems, the company is working to turn COM+ and the MTS (Microsoft Transaction Server) into one programming model that will simplifying the lives of developers building distributed, enterprise-wide COM applications. COM+ integrates seamlessly with all COM-aware languages (basically Microsoft languages). Users write components in their favorite language. The tool chosen and the COM+ runtime take care of turning these classes into COM components [Kirtland 97].

Usage Considerations

A number of issues must be evaluated when considering COM, DCOM, and COM+. They include

- *Platform support.* COM and DCOM are best supported on Windows 95 and NT platforms. However, Microsoft has released a version of COM/DCOM for MacOS that supports OLE-style compound documents and the creation of ActiveX controls. Software AG, a Microsoft partner, has released DCOM for some UNIX operating systems, concretely OS/390,

HP-UX 11.0, SUN Solaris, AIX 4.2, 4.3, Tru64 Unix 4.0 and Linux. However, DCOM over non-Windows platforms has few supporters. Until DCOM for alternate platforms has solidified, the technology is best applied in environments that are primarily Windows-based.

- *Platform specificity of COM/DCOM components.* Because COM and DCOM are based on a native binary format, components written to these specifications are not platform independent. Thus, either they must be recompiled for a specific platform, or an interpreter for the binary format must become available. Depending on your perspective, the use of a binary format may be either an advantage (faster execution, better use of native platform capabilities) or a disadvantage (ActiveX controls, unlike Java applets, are NOT machine independent). See Java for more information.
- *Security.* Because COM/DCOM components have access to a version of the Microsoft Windows API, "bad actors" can potentially damage the user's computing environment. In order to address this problem, Microsoft employs "Authenticode" [Microsoft 96] which uses public key encryption to digitally sign components. Independent certification authorities such as VeriSign issue digital certificates to verify the identity of the source of the component [VeriSign 97]. However, even certified code can contain instructions that accidentally, or even maliciously, compromise the user's environment.
- *Support for distributed objects.* COM/DCOM provides basic support for distributed objects. There is currently no support for situations requiring real time processing, high reliability, or other such specialized component interaction.
- *Stability of APIs.* In October of 1996 Microsoft turned over COM/DCOM, parts of OLE, and ActiveX to the Open Group (a merger of Open Software Foundation and X/Open). The Open Group has formed the Active Group to oversee the transformation of the technology into an open standard. The aim of the Active Group is to promote the technology's compatibility across systems (Windows, UNIX, and MacOS) and to oversee future extension by creating working groups dedicated to specific functions. However, it is unclear how much control Microsoft will relinquish over the direction of the technology. Certainly, as the inventor and primary advocate of COM and DCOM, Microsoft is expected to have strong influence on the overall direction of the technology and underlying APIs.
- *Long-term system maintainability.* Microsoft is actively supporting COM and DCOM technology and pushing it in distributed and Web-based directions. Microsoft is also trying to preserve existing investments in COM technology while introducing incremental changes. Microsoft, for example, has ensured backward compatibility of COM+. Although this affirmation is in general true, COM objects that access local information in the registry or in system folders may require modification. In general, the PC community has not been faced with the concern of very long-lived systems, and vendors often provide support only for recent releases.

Maturity

COM has its roots in OLE version 1, which was created in 1991 and was a proprietary document integration and management framework for the Microsoft

Office suite. Microsoft later realized that document integration is just a special case of component integration. OLE version 2, released in 1995 was a major enhancement over its predecessor. The foundation of OLE version 2, now called COM, provided a general-purpose mechanism for component integration on Windows platforms [Brockschmidt 95]. While this early version of COM included some notions of distributed components, more complete support for distribution became available with the DCOM specifications and implementations for Windows95 and Windows NT released in 1996. Beta versions of DCOM for Mac, Solaris and other operating systems followed shortly after.

There are many PC-based applications that take advantage of COM and DCOM technology. The basic approach has proven sound, and as previously mentioned, a large component industry has sprung up to take advantage of opportunities created by the Microsoft technology. On the other hand, DCOM has just arrived on non-Windows platforms, and there is little experience with it. DCOM for non-Windows platforms is mainly used to communicate COM based programs with legacy applications in Mainframes and Unix workstations.

COM+ is much younger than COM, it was announced in Sept. 23, 1997 and shipped with windows 2000 (a.k.a. Windows NT 5.0). COM+ can be considered the next release of COM. We are unaware of any large-scale distributed applications relying on COM+ support.

The computing paradigm for distributed applications is in flux, due to the relative immaturity of the technology and recent advances in web-based computing. The Web-centered computing industry has begun to align itself into two technology camps-with one camp centered around Microsoft's COM/DCOM/COM+, Internet Explorer, and ActiveX capabilities, and the other camp championing Netscape, CORBA, and Java/J2EE solutions. Both sides argue vociferously about the relative merits of their approach, but at this time there is no clear technology winner. Fortunately, both camps are working on mechanisms to support interplay between the technology bases. Thus, a COM/DCOM to CORBA mapping is supported by CORBA vendors [Foody 96], and Microsoft has incorporated Java into an Internet strategy. However, work on interconnection between the competing approaches is not complete, and each camp would shed few tears if the other side folded.

Costs and Limitations

Low cost development tools from Microsoft (such as Visual C++ or Visual Basic), as well as tools from other vendors provide the ability to build and access COM components for Windows platforms. Construction of clients and servers is straightforward on these platforms. In addition, the initial purchase price for COM and DCOM is low on Windows platforms. For other platforms the prices are considerably more expensive. DCOM for mainframes, for example, costs around two hundred thousand dollars by December 1999.

Beyond basic costs to procure the technology, any serious software development using COM/DCOM/COM+ requires substantial programmer expertise-the complexities of building distributed applications are not eliminated. It would be a serious mistake to assume that the advent of distributed object

technologies like COM/DCOM/COM+ reduces the need for expertise in areas like distributed systems design, multi-threaded applications, and networking.

However, Microsoft has a strong support organization to assist individuals developing COM/DCOM clients and objects: many sample components, books and guides on the subject of COM/DCOM development are available. Unfortunately, information on COM+ is limited at this time.

Dependencies

Dependencies include Remote Procedure Call and Distributed Computing Environment.

Alternatives

COM/DCOM/COM+ represents one of a number of alternate technologies that support distributed computing. Some technologies, such as remote procedure call, offer "low level" distribution support. Other technologies, such as message oriented middleware and transaction processing monitors, offer distribution support paradigms outside the realm of objects. The Common Object Request Broker Architecture (CORBA) and Java 2 Enterprise Edition (J2EE) can be considered direct competitors to COM/DCOM. Information about technologies supporting distributed computing is available in the following places:

- Distributed Computing Environment
- Remote Procedure Call
- Message-Oriented Middleware
- Transaction Processing Monitor Technology
- Common Object Request Broker Architecture
- Two Tier Software Architectures
- Java

Complementary Technologies

One commonly hears of COM and DCOM in conjunction with OLE, ActiveX, MTS and COM+. Indeed, these and other technologies constitute Microsoft's distributed and web-oriented strategy. This strategy is globally referred as Distributed interNet Architecture(tm) (DNA) and it comprises a full set of products and specifications to implement net-centric applications.

Technologies championed by other vendors can also be used in conjunction with COM. For example, COM objects can be created and manipulated from Java code. Tools are provided to create Java classes from COM type library information-these classes can be included in Java code. Using Internet Explorer, Java programs can also expose functionality as COM services. In general, Microsoft's approach for Java support involves tying it very closely to its existing Internet strategy (Internet Explorer, COM/DCOM, ActiveX); i.e., to provide a mechanism for interfacing to the wide range of components that already adhere to Microsoft's strategy and specifications.

COM+ is a good candidate to implement the middle layer of multitier architectures. The distribution support and quality of service provided by COM+ can help to overcome some of the complexities involved in these architectures.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Component Object Model (COM), DCOM, and Related Capabilities
Application category	Software Architecture Models (AP.2.1.1) Client/Server (AP.2.1.2.1) Client/Server Communications (AP.2.2.1)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1) Reusability (QM.4.4)
Computing reviews category	Distributed Systems (C.2.4) Object-Oriented Programming (D.1.5)

References and Information Sources

[Active 97] Active Group home page [online]. Available WWW <URL: http://www.activex.org/> (1997).

[Brockschmidt 95] Brockschmidt, Kraig. Inside OLE, 2nd edition, Microsoft Press, 1995

[Chappell 96] Chappell, David. *DCE and Objects* [online]. Available WWW <URL: http://www.opengroup.org/dce/info/dce_objects.htm> (1996).

[COM 95] Microsoft Corporation. *The Component Object Model Specification*, Version 0.9, October 24, 1995 [online]. Available WWW <URL: http://www.microsoft.com/com/resources/comdocs.asp>(1995).

[DCOM 97] Microsoft Corporation. *Distributed Component Object Model Protocol*-DCOM/1.0, draft, November 1996 [online]. Available WWW <URL: http://www.globecom.net/ietf/draft/draft-brown-dcom-v1-spec-03.html> (1996).

[Foody 96] Foody, M.A. "OLE and COM vs. CORBA." *UNIX Review 14*, 4. (April 1996): 43-45.

[Harmon 99] Harmon, Paul. *Microsoft transaction Server*. Component development Strategies Vol IX No 3. Available WWW <URL: http://www.cutter.com/cds/1999toc.htm#mar > 1999

[Kirtland 97] Kirtland, Mary. "The COM+ Programming Model Makes it Easy to Write Components in Any Language". Microsoft System Journal. Dec, 1997.

[Microsoft 96] Microsoft Corporation. *Microsoft Authenticode Technology* [online]. Available WWW <URL: http://www.microsoft.com/security/tech/misf8.htm> (1996).

[MSCOM 97] Microsoft home page [online]. The site provides information about COM, DCOM and OLE. Available WWW <URL: http://www.microsoft.com/> (1997).

[OMG 97] Object Management Group home page [online]. The site provides information comparing DCOM (ActiveX) to CORBA. Available WWW <URL: http://www.omg.org/> (1997).

[VeriSign 97] Verisign home page [online]. Available WWW <URL: http://www.verisign.com> (1997).

Current Author/Maintainer

Santiago Comella-Dorda, SEI

External Reviewers

Modifications

13 Mar 2001: Update with new developments of COM

23 June 1997: Total replacement text
10 Jan 1997: Original

URL: http://www.sei.cmu.edu/str/descriptions/com_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Computer System Security--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

C4I systems include networks of computers that provide real-time situation data for military decision makers and a means of directing response to a situation. These networks collect data from sensors and subordinate commands. That data is fused with the existing situation status data and presented by the C4I system to decision makers through display devices. C4I networks today may incorporate two general types of networks: networks of Multi-level Secure (MLS) Systems, and Intranets of single level systems. Figure 5 shows the relevant major security components of a C4I computer system network.

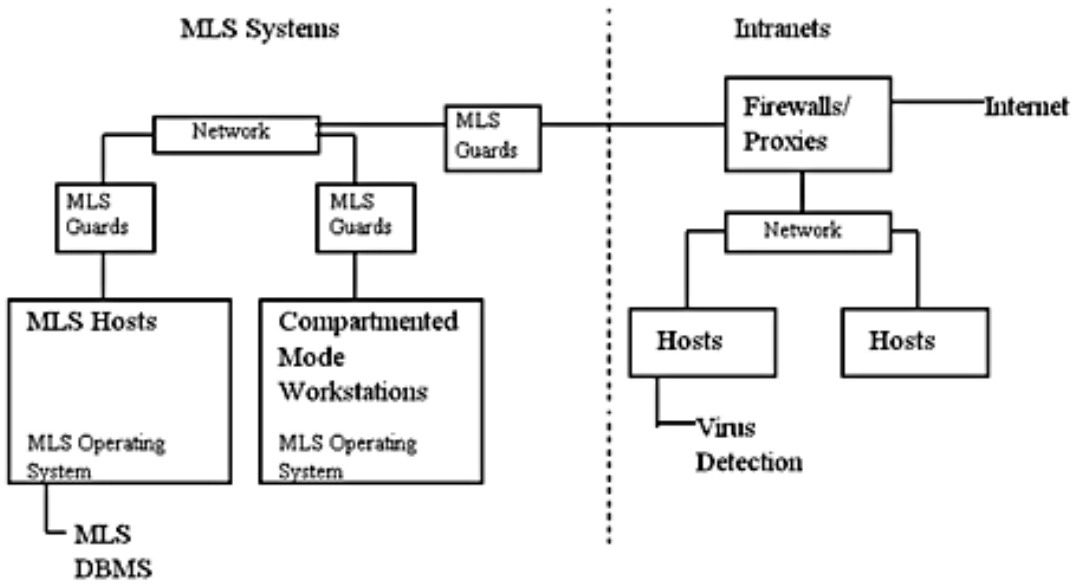


Figure 5: Computer System Security in C4I Systems

This technology description is tutorial in nature. It provides a general overview of key concepts and introduces key technologies. Detailed discussions of the individual technologies can be found in the referenced technology descriptions.

Technical Detail

Some computers in the network are hosts that collect and process data. A host can be a mainframe, a server, a workstation, or a PC. It may perform the function of an application processor, a communication processor, a database processor, a display processor, or a combination. The security mode for the host may be single-level or multi-level. A single-level host processes all data as though it was one security level. A multi-level host can process data at different security levels, identify and isolate data in the appropriate levels or categories, and distribute data only to the appropriately cleared users.

C4I systems benefit from multi-level security implementations because C4I systems fuse data from sources with a wide range of security levels and provide status, warning data, or direction to war fighting systems that may be at lesser security levels. An MLS operating system (see Multi-Level Secure One Way Guard with Random Acknowledgment) provides the software that makes a host MLS. A particular kind of MLS host is the Compartmented Mode Workstation (CMW). A CMW is a MLS host that has been evaluated to satisfy the Defense Intelligence Agency CMW requirements [Woodward 87] in addition to the Trusted Computer System Evaluation Criteria [DoD 85]. A MLS host may use a MLS DBMS (see Multi-Level Secure Database Management Schemes) to store and retrieve data at multiple security levels. A MLS guard provides a secure interface across a security boundary between systems operating at different security levels or modes.

MLS guards may allow data across the interface automatically or may require manual review of data and approval of transfer on an attached terminal. They also may control data transfer across the interface in both directions or be limited to allowing data to be transferred one way, usually from the low security level side of a security boundary to the high security level side. One-way guards are usually the easiest to implement and accredit for use. Data integrity is an issue with one-way guards because an acknowledgment message can not be used. Recent research in one-way guards has addressed allowing an acknowledgment message (see Multi-Level Secure One Way Guard with Random Acknowledgment).

Intranets use the same kind of networking software (e.g., TCP/IP, Telnet, Netnews, DNS, browsers, home pages) that is used on the Internet, but Intranets use them on a private dedicated network. They are in essence a private Internet. They are used in a growing number of ways in many military and corporate networks including mission performance, off-line processing of raw data, administrative support, and mail networks. They may be incorporated into C4I systems using firewalls or proxies (see Firewalls and Proxies) and MLS guards. Firewalls or proxies may be used to provide a security interface to the Internet. If the Intranets are to be connected to MLS systems, they must be connected through MLS guards. In an environment with Intranet hosts, a major concern is Virus Detection and Intrusion Detection. PCs on a network are particularly susceptible to virus attacks from other hosts on the network or the Internet. PCs are also vulnerable to viruses carried on floppy disks. Since PCs are now in most homes, transfer of files from home to work via floppy disk

provides the risk of introducing a virus into the Intranet. PCs are more vulnerable to viruses than UNIX-based workstations or mainframes because the PC has no memory protection hardware and the operating system (DOS and Windows) allows a program to access any part of memory or disk.

Security across the networks in a C4I system is crucial. Traditionally this security is provided by physically protecting the equipment and cables in the network for localized networks. When that is not possible, the network connections are encrypted using encryption hardware in the communications paths. End-to-end encryption is an alternative that encrypts the data using software before it is put on the network and decrypts it after it has been taken off of the network. Then non-encrypted circuits can be used for communications.

Any encryption system involves the distribution of keys used by the encryption algorithm for the encryption/decryption of messages and data. Encryption keys must be replaced periodically to enhance security or when the key has been compromised or lost. Traditionally these keys have been distributed through couriers or encrypted circuits. Public key cryptography provides a means of electronic encryption key distribution that can lower the security risk and administrative workload associated with encryption.

Data integrity is another issue associated with the networks used in C4I systems. Public Key Digital Signatures and providing for Nonrepudiation in Network Communications are two means to enhance data integrity. Public key digital signatures, which make use of public key encryption and message authentication codes, are a means to authenticate that data came from the person identified as the sender and that the data has not been modified. The nonrepudiation process uses a digital signature and a trusted arbitrator process to assure that a particular message has been sent and received and to establish the time when this occurred.

Usage Considerations

MLS systems require specialized knowledge to build, accredit, and maintain. The cost of MLS systems can be high. The system development overhead and operational performance overhead associated with MLS systems are substantial. They are difficult to implement in an "open" configuration because open requirements sometimes conflict with MLS requirements. On the other hand, using MLS techniques may be the only allowable way to construct some C4I systems. Operational security vulnerabilities may be unacceptable without MLS implementations. Procedural security approaches may be too slow for an operational C4I system as a non-MLS approach. A single-level system approach may be too restrictive. For example, a secret single-level system that contains unclassified, confidential, and secret data will not release confidential data to a user who is cleared for confidential and needs the data. That is because the system cannot determine what data is confidential rather than secret. Further usage discussions are addressed in individual technology descriptions.

The National Security Agency (NSA) Multilevel Information Systems Security Initiative (MISSI) is an evolutionary effort intended to provide better MLS capability in a cost-effective manner [MISSI 96]. This effort was initiated after the

Gulf War when it was recognized that war fighting commanders needed MLS systems in order to incorporate intelligence and other highly classified data into their planning and operations in a timely manner. The MISSI effort is developing a set of building block products that can be obtained commercially to construct an MLS system. The initial products include the FORTEZZA crypto cards and associated FORTEZZA ready workstation applications to control access to and protect data on a workstation in a network environment. Other products include high-assurance guards and firewalls to provide access control and encryption services between the local security boundary and external networks. MISSI will also include secure computing products that provide high-trust operating systems and application programs for MLS hosts, and network encryption and security management products. These products can be incorporated into developing MLS systems as the products become available.

Maturity

See individual technologies.

Costs and Limitations

See individual technologies.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Computer System Security - an Overview
Application category	Information Security (AP.2.4)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating Systems Security & Protection (D.4.6), Security & Protection (K.6.5), Computer-Communications Networks Security and Protection (C.2.0)

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Woodward 87] Woodward, John. *Security Requirements for High and Compartmented Mode Workstations* (MTR 9992, DDS 2600-5502-87). Washington, DC: Defense Intelligence Agency, 1987.

[DoD 85] *Department of Defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC)* (DoD 5200.28-STD 1985). Fort Meade, MD: Department of Defense, 1985. Also available WWW <URL: http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html> (1985).

[MISSI 96] MISSI Web site [online]. Available WWW <URL: http://beta.missilab.com> (1996).

[Russel 91] Russel, Deborah & Gangemi, G.T. Sr. *Computer Security Basics.* Sebastopol, CA: O'Reilly & Associates, Inc., 1991.

[White 96] White, Gregory B.; Fisch, Eric A.; & Pooch, Udo W. *Computer System and Network Security*. Boca Raton, FL: CRC Press, 1996.

Current Author/Maintainer

Tom Mills, Lockheed Martin

External Reviewers

Brian Gallagher, SEI

Modifications

8 July 97: added reference to MLS One-Way Guard with Random Ack.
20 June 97: updated URL for [MISSI 96]
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/security_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

COTS and Open Systems--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

One of the latest trends in systems development is to make greater use of commercial-off-the-shelf (COTS) products. While this change has been encouraged for many years for all kinds of systems development, especially in the Department of Defense (DoD), it is only in the early 1990s that the practice has been mandated by everyone from industry executives to Congress.

At the same time, an open systems approach to develop systems has been gaining popularity, with visions of open systems that are "plug-and-play" compatible, where components from one supplier can be easily replaced by those from another supplier. Advocates of open systems often confuse them with the use of COTS products, making it difficult for the average engineer to know just what (s)he should be doing to develop (and maintain) systems more effectively.

These two concepts- the use of COTS products and the creation of open systems- are closely related and complementary, although definitely *not* synonymous. The purpose of this technology description is to

- define/clarify what each is
- explain the differences between them
- examine the benefits each brings to the development, maintenance, and evolution of systems

A brief summary of the key points in this technology description follows:

- COTS products hold the potential for cost-effective acquisition of components and advancing technology, but they are *not* necessarily open.
- Open systems emphasize (1) the use of interface standards and (2) the use of implementations that conform to those standard interfaces. Open

systems provide stability and a framework for the effective use of COTS products and other non-developmental items (NDI) through the use of interface standards. *Well-chosen interface standards can outlast any particular product, vendor, or technology.*

- It is possible to use COTS products without creating an open system; similarly, it is possible to create an open system without significant use of COTS products or NDI.
- COTS products and an open systems approach are both means to important system goals of improving the quality and performance of our systems, developing them more quickly, and sustaining them more cost-effectively. The greatest advantage can be gained from using these two approaches together.

For further detail on COTS, open systems, and component-based software development approaches, see Component-Based Software Development/COTS Integration.

Technical Detail

COTS. The term "COTS" is meant to refer to things that one can buy, *ready-made,* from some manufacturer's virtual store shelf (e.g., through a catalogue or from a price list). It carries with it a sense of getting, at a reasonable cost, something that already does the job. It replaces the nightmares of developing unique system components with the promises of fast, efficient acquisition of cheap (or at least cheaper) component implementations.

Because of the need for precision in procurement, the federal government has defined the term "commercial item." The full text of this definition can be found in the Federal Acquisition Regulations (FARs); the following is a summary [FAR 96]:

A *commercial item* is

1. property[1] customarily used for nongovernmental purposes and has been sold, leased, or licensed (or offered for sale, lease or license) to the general public;
2. any item evolved from an item in (1) through advances in technology and is not yet available commercially but will be available in time to satisfy the requirement;
3. any item that would satisfy (1) or (2) but for modifications customarily available in the commercial marketplace or minor modifications made to meet Federal Government requirements;
4. any combination of items meeting (1) - (3) above;
5. services for installation, maintenance, repair, training, etc. if such services are procured for support of an item in (1), (2), or (3) above, as offered to the public or provided by the same work force as supports the general public, or other services sold competitively in the commercial marketplace;
6. a nondevelopmental item developed exclusively at private expense and sold competitively to multiple state and local governments.

Most people would agree that these ideas approximate the meaning of "commercial-off the-shelf" (COTS) products, although the inclusion of "services" as "COTS" is not often included. The salient characteristics of a COTS product are

- it exists *a priori*
- it is available to the general public
- it can be bought (or leased or licensed)

Non-developmental item. A closely-related term that is often heard in government (especially DoD) circles is *nondevelopmental item* (NDI). In summary, a *nondevelopmental item* is [FAR 96]:

1. any previously developed item used exclusively for government purposes by a federal, state, or local agency or government or by a foreign government that has a mutual defense agreement with the U.S.;
2. any item described in (1) above that requires only minor modification or modifications normally available in the commercial marketplace to meet requirements;
3. any item being produced that does not meet (1) or (2) above only because it is not yet in use.

The key point here is that NDI refers to something that was developed by someone else. It might have been developed by a commercial interest, but typically it will have been for some other government, department, or agency. A large-scale example of an NDI would be a fighter aircraft that was developed by some other nation. A more meaningful example in the current context would be a navigation software subsystem developed for one aircraft that is available for use in another aircraft. The salient characteristics of a nondevelopmental item are the following:

- it exists, although not necessarily off some vendor's "shelf."
- it is available, although not necessarily to the general public.
- it can be obtained for use, although more likely off an existing contract than off a published price list.

While there are certain reasons for using caution in applying the definitions of COTS and NDI (e.g., how safe is a "minor modification," and what if it just *looks* like a vendor has a product, whereas it is in reality just vaporware?), they do fairly characterize the features that are of interest to those who believe that "buying COTS" is desirable and beneficial. However, although closely related, there are differences between NDI and COTS items:

- COTS products would most likely be found in some sort of catalogue or price list, whereas it may be more difficult to discover the existence of NDI.
- The range of possibilities opened up by NDI is broader than what COTS products alone can offer, but NDI could lack the commercial leverage that is sought in the use of COTS products.

Open systems. The basic tenet of open systems is the use of interface standards in the engineering and design of systems, coupled with the use of implementations (preferably, but not necessarily, COTS and non-developmental items (NDI)) that conform to those interface standards. This provides a stable basis on which to make decisions about the system, particularly with regard to its evolution**.** An open systems approach has the potential to reduce the developmental *cost* and schedule of systems while maintaining or even improving performance. The dependence on stable interface standards makes open systems more adaptable to advances in technology and changes in the marketplace.

When people use the phrase open systems, they most often have in mind a system that is flexible and adaptive, one that is "open" to inclusion of many products from many sources. The phrase open systems often carries with it an image of easy "plug-and-play" between components and products that were *not* necessarily originally designed to work together. Open systems also hold out the promise of being able to take immediate advantage of new technology as it emerges, because it should be easier to plug in new technology, either in place of an old component(s) or as a new extension of the system.

Many different definitions of open system have been offered in the past. To find a truly workable one, we must look more closely at what it takes to make this vision a reality. For the purposes of this technology description, open systems is defined as follows [Meyers 97]:

An *open system* is

a collection of interacting software, hardware, and human components, designed to satisfy stated needs, with the interface specification of components

- fully defined
- available to the public
- maintained according to group consensus, and

in which the implementations of components are conformant to the specification. One key part of the definition addresses a set of criteria for the interface specifications/standards. Not only must they be fully defined, but they must also be available to the public. This implies that cost and public access may not be prohibitive constraining factors; that is, the specification cannot be available only to a selected group of people who have some special interest. Anyone is free to obtain a copy of the specification (perhaps at the cost of duplication and distribution, perhaps even at the cost of a small license fee) and they are also free to produce and sell implementations of that specification. It is also very important that the specification is of interest to a wide range of parties and is not exclusively under the control of any single vendor. To this end, the definition includes the idea that maintenance of the specification is by group consensus.

Taken together, these criteria come very close to requiring that the interface specification be a "standard."

The main benefit of this definition of open system is that it is *operational*. That is, it can be applied to a single system at a given point in time. In contrast, most other popular definitions identify desirable system qualities that open systems are expected to display, such as portability, interoperability, and scalability. Unfortunately, there is no way to measure a system with respect to these qualities at a single point in time (e.g., "Portable" to what platforms? And how many? "Interoperable" with what other systems or components? And how many? "Scalable" for what use? To what size?). Each of these qualities implies a relationship, either between the subject system and some other unspecified one (s) or between the subject system and itself over time.

This definition also supports the vision of what people hope to achieve with open systems. The very phrase "plug-and-play" brings to mind children's toys like Tinker Toys™ and Legos™. The key to them is a small set of well-defined, consistently-enforced interfaces. It also invokes the images of hardware components that can be plugged together because, for example, the pins and configuration of the female connector are perfectly complementary to those of the male connector. All these schemes have interface standards in common.

Most of the interface standards used in computer-based systems are for application program interfaces (APIs), data formats, or protocols. For all of these kinds of interface standards, one can find fully-defined specifications; without such clear definition in the specifications, wide variation quickly emerges among implementations, and this undermines the intended compatibility. Interface standards are made widely available to the public to generate a thriving market for components that can be plugged together. They are maintained using many forms of group consensus; this precludes one vendor or group from making arbitrary changes to the interface standard that will limit competition and availability of alternative products.

Finally, for many of these interface standards it is possible to tell whether or not a given implementation really matches the specification; this is called *conformance*. If the implementations all match the specification/standard closely enough, then one kind of incompatibility between components can be reduced if not eliminated, and it *may* be possible to "plug" them into a system and get them to "play" with the other components.[2] On the other hand, if implementations only loosely implement the standard or if incompatible interpretations cannot be detected before trying to integrate a component into the system, then it is less likely that the envisioned flexibility and adaptability can be realized.

It is important to realize that it is possible to create an open system, based on interface standards, in which no COTS products or NDI are used. This might be necessary in a situation where, for example, no COTS product conforming to the interface standard also meet other system requirements, such as for real-time performance or security. Although one would not gain the economic and schedule advantages of using a component implementation that already existed and was shared and supported by a number of users, the interface standards would still provide the framework for future evolution of the system (provided vendors do eventually pick up the standard and produce conformant products).

Potentially some future product may emerge that does meet all the requirements. In the mean time, the system enjoys the clarity and stability of a well-defined specification.

Usage Considerations

There currently is a very strong push within the federal government, particularly DoD, to make more use of COTS products and NDI.[3] In addition to action by DoD leaders, the Federal Acquisition Streamlining Acts of 1994 and 1995 directed the increased use of commercial items, coupled with several adjustments to the federal procurement regulations to encourage the new approach. Carney outlines current government trends toward using commercial-off-the-shelf (COTS) products [Carney 97a, Carney 97b].

The reasoning behind these directives and laws is that government organizations typically spend far too much effort on defining to the lowest level of detail the desired characteristics of systems and how the contractors are to build those systems to achieve those characteristics. Thus a lot of resources are expended developing systems and components that often already exist- or exist in "good enough" form with nearly the same capabilities- elsewhere. The prevailing, and time-consuming, approach is always to develop from the ground up; this approach results in unique systems each time. The result is systems that are

- very expensive, with only one customer to bear the development and maintenance costs over the life of the component or system
- inflexible and unable to easily capitalize on advances in technology
- historically fielding technology that is in excess of ten years old

Shifting to a paradigm in which systems are built primarily of components that are available commercially offers the opportunity to lower costs by sharing them with other users, thus amortizing them over a larger population, while taking advantage of the investments that industry is putting into the development of new technologies.

Open systems can have a positive impact either on new systems development or in the context of legacy systems. Although there is generally more decision-making freedom in the case of a new development, open systems can nevertheless help shape an evolutionary path for a legacy system that will help turn it into a more flexible and maintainable system.

Many initiatives are under way, both in the DoD and in individual services, agencies, and companies, that are designed to promote the use of an open systems approach and to secure even greater benefits than can be realized from the use of COTS products alone. These initiatives are occurring because projects have been learning the hard way that "just buying COTS" does not necessarily secure all of the benefits desired. There are other problems and sources of risk introduced by the use of COTS products.

COTS products are not necessarily open. That is, they do not necessarily conform to any recognized interface standards. Thus it is possible (in fact, likely) that using a COTS product commits the user to proprietary interfaces and solutions that are not common with any other product, component, or system. If the sole objective is the ability to capture new technology more cheaply, then the use of COTS products that are not open will do. But when one considers the future of such a system, the disadvantages of this approach become apparent. Many DoD systems have a 30- to 50-year lifetime, while the average COTS component is upgraded every 6 to 12 months and new technology appears on the scene about every 18 to 24 months. Thus any money that is saved by procuring a COTS product with proprietary interfaces will quickly be lost in maintenance as products and interfaces change- the ability to migrate cost-effectively to other products and other technologies in the future will have been lost.

Even if the expected lifetime of a system is only 5 to 10 years, the fluctuations in COTS products and technology result in a state of constant change for any system employing them. Interface standards provide a source of stability in the midst of all this. Without such standards every change in the marketplace can impose an unanticipated and unpredictable change to systems that use products found in the marketplace. This situation is particularly painful when the vendor stops supporting the product or goes out of business altogether, thus forcing a change to a different product or vendor.

Program managers and lead engineers should also know that the depth of understanding and technical and management skills required on a project team is not necessarily diminished or decreased because of the use of COTS or open systems. Arguably, the skills and understanding needed increase because of the potential complexity of integration issues, the need to seriously consider longer term system evolution as part of initial development, and the need to make informed decisions about which products and standards are best.

Paradoxically, given the desire to produce systems more quickly, the emphasis on standards can actually be something of an inhibitor. Some standards efforts, in their desire to achieve maximum consensus, have very long cycle times (five or more years), which certainly do not fit well with product development and release cycles. This conflict is of concern and is being addressed by some standards bodies, but it has led some projects to become involved with consortia standards and also with de facto industry standards. While these are often practical alternatives, they do have attendant risks; the de facto standards may be proprietary, for example, and this limits long-term evolution. The key is to weigh the risks of straying from the three basic criteria (fully-defined, available to the public, and maintained according to group consensus) against what is gained over the long term.

Maturity

The open systems concept has been at least partially introduced into C3I systems, but it has been difficult to move into the realm of real-time embedded systems, particularly weapon systems, where it is much more difficult to find standards that meet a system's requirements. Examples might include cases of extreme real-time performance or security concerns.

There is limited documented experience with the open systems approach. An example of successful use in the DoD is the Intelligence and Electronics Warfare Common Sensor (IEWCS) program [IEWCS 96]. A survey of the awareness, understanding, and implementation of open system concepts within the DoD is available from the Open Systems Joint Task Force (OSJTF) [OSJTF 96].

There is more experience with the use of COTS items, but often this experience is with COTS hardware. The concerns, problems, and solutions for COTS-based software systems are somewhat different and not as well understood.

Costs and Limitations

An open systems approach requires investments in the following areas early in a program's life cycle and on an ongoing basis:

- market surveys to determine the availability of standards
- standards selection
- standards profiling- the coordination and tailoring of standards to work together
- selection of standards-compliant implementations

These costs/activities are the necessary foundation for creating systems that serve current needs and yet can grow and advance as technology advances and the marketplace changes.

A separate cost is the continued willingness of the government to invest in standards development and maturation activities. While these activities are most often handled at high government levels concerned with standards development and usage (for example, Defense Information Systems Agency (DISA) in the DoD), it is likewise important for individual programs (especially the larger programs) to stay informed in this area. For example, individual programs should be concerned about the following issues:

- When are revisions to specific standards coming out?
- What changes are proposed in the new revision?
- When are ballots on the revisions going to occur?
- Where are the implementations headed?

A COTS-based systems approach also requires new and different investments:

- market research on available and emerging products and technologies
- COTS product evaluation and selection
- "black box" integration of COTS components

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	COTS and Open Systems
Application category	Interfaces Design (AP.1.3.3) Software Architecture (AP.2.1)
Quality measures category	Openness (QM.4.1.2) Interoperability (QM.4.1) Maintainability (QM.3.1)
Computing reviews category	Software Engineering Design (D.2.10) Software Engineering Miscellaneous (D.2.m)

References and Information Sources

[Carney 97a] Carney, D, & Oberndorf, P. "The Commandments of COTS: Still Searching for the Promised Land." *Crosstalk 10*, 5 (May 1997): 25-30. Also available online at <URL: http://www.stsc.hill.af.mil/Crosstalk/frames.asp?uri=1997/05/commandments.asp>.

[Carney 97b] Carney, D. *Assembling Large Systems from COTS Components: Opportunities, Cautions, and Complexities* [online]. Available WWW <URL: /cbs/papers/paper13a.html>.

[FAR 96] *Federal Acquisition Regulations.* Washington, DC: General Services Administration, 1996.

[IEWCS 96] *Open Systems Joint Task Force Case Study of U.S. Army Intelligence and Electronic Warfare Common Sensor* (IEWCS) [online]. Available WWW <URL: http://www.acq.osd.mil/osjtf/how_to_do_os/program_apps/index_1b.html> (1996).

[Meyers 97] Meyers, Craig & Oberndorf, Tricia. *Open Systems: The Promises and the Pitfalls*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997.

[OSJTF 96] *Open Systems Joint Task Force Baseline Study* [online]. Available WWW <URL: http://www.acq.osd.mil/osjtf/current_activities/studies_and_projects/baseline.doc> (1996).

Current Author/Maintainer

Tricia Oberndorf, SEI

Modifications

7 August 97: updated [Carney 97a] reference
2 July 97: incorporated new definitions for COTS and NDI from the FARs
Minor updates in Maturity section and Costs and Limitations
9 April 97: minor edits; no content changes
10 Jan 97 (original); co-author for this version: John Foreman, SEI

Footnotes

[1] "Property" in this definition explicitly excludes real property.

[2] It should be noted that interface specifications are in general not sufficient to ensure full "plug-and-play" operation. In practice, the real interface between two components of a system consists of all the assumptions that each makes about the other. APIs, data formats, and protocols address a large number of these assumptions, but by no means all of them. It remains for further investigations to determine the full set of interface knowledge that must be standardized to ever get really close to an ideal "plug-and-play" system creation process.

[3] In June 1994 Secretary of Defense William Perry directed that DoD acquisitions should make maximum use of performance specifications and commercial standards. In November 1994 Undersecretary of Defense (Acquisition and Technology) Paul Kaminski directed "that `open systems' specifications and standards be used for acquisition of weapon systems electronics to the greatest extent practical."

URL: http://www.sei.cmu.edu/str/descriptions/cots_body.html
Last Modified: 24 July 2008

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Cyclomatic Complexity

Software Technology Roadmap

Status

Advanced

Note

We recommend reading Maintenance of Operational Systems--An Overview before reading this description; it offers a view of the life cycle of software from development through reengineering. We also recommend concurrent reading of Maintainability Index Technique for Measuring Program Maintainability, which illustrates a specific application of cyclomatic complexity to quantify the maintainability of software. These descriptions provide a framework for assessing the applicability of cyclomatic complexity and other technologies to a specific environment.

Purpose and Origin

Cyclomatic *complexity* is the most widely used member of a class of static software metrics. Cyclomatic complexity may be considered a broad measure of *soundness* and *confidence* for a program. Introduced by Thomas McCabe in 1976, it measures the number of linearly-independent paths through a program module. This measure provides a single ordinal number that can be compared to the complexity of other programs. Cyclomatic complexity is often referred to simply as program complexity, or as McCabe's complexity. It is often used in concert with other software metrics. As one of the more widely-accepted software metrics, it is intended to be independent of language and language format [McCabe 94].

Cyclomatic complexity has also been extended to encompass the design and structural complexity of a system [McCabe 89].

Technical Detail

The cyclomatic complexity of a software module is calculated from a connected graph of the module (that shows the topology of control flow within the program):

Cyclomatic complexity (CC) = E - N + p

where E = the number of edges of the graph

N = the number of nodes of the graph

p = the number of connected components

To actually count these elements requires establishing a counting convention (tools to count cyclomatic complexity contain these conventions). The complexity number is generally considered to provide a stronger measure of a program's structural complexity than is provided by counting lines of code. Figure 6 is a connected graph of a simple program with a cyclomatic complexity of seven. Nodes are the numbered locations, which correspond to logic branch points; edges are the lines between the nodes.

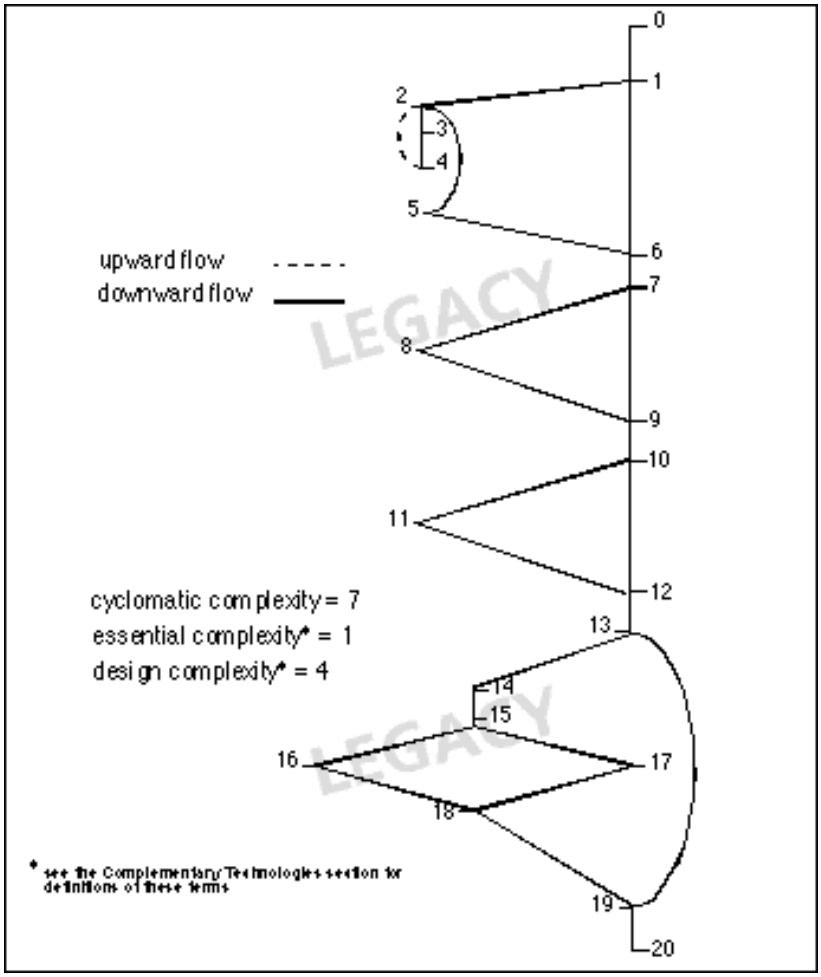


Figure 6: Connected Graph of a Simple Program

A large number of programs have been measured, and ranges of complexity have been established that help the software engineer determine a program's inherent risk and stability. The resulting calibrated measure can be used in development, maintenance, and reengineering situations to develop estimates of risk, cost, or program stability. Studies show a correlation between a program's cyclomatic complexity and its error frequency. A low cyclomatic complexity contributes to a program's *understandability* and indicates it is amenable to *modification* at lower risk than a more complex program. A module's cyclomatic

complexity is also a strong indicator of its *testability* (see Test planning under Usage Considerations).

A common application of cyclomatic complexity is to compare it against a set of threshold values. One such threshold set is in Table 4:

Table 4: Cyclomatic Complexity

Cyclomatic Complexity	Risk Evaluation
1-10	a simple program, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk program
greater than 50	untestable program (very high risk)

Usage Considerations

Cyclomatic complexity can be applied in several areas, including

- *Code development risk analysis*. While code is under development, it can be measured for complexity to assess inherent risk or risk buildup.
- *Change risk analysis in maintenance*. Code complexity tends to increase as it is maintained over time. By measuring the complexity before and after a proposed change, this buildup can be monitored and used to help decide how to minimize the risk of the change.
- *Test Planning*. Mathematical analysis has shown that cyclomatic complexity gives the exact number of tests needed to test every decision point in a program for each outcome. Thus, the analysis can be used for test planning. An excessively complex module will require a prohibitive number of test steps; that number can be reduced to a practical size by breaking the module into smaller, less-complex sub-modules.
- *Reengineering*. Cyclomatic complexity analysis provides knowledge of the structure of the operational code of a system. The risk involved in reengineering a piece of code is related to its complexity. Therefore, cost and risk analysis can benefit from proper application of such an analysis.

Cyclomatic complexity can be calculated manually for small program suites, but automated tools are preferable for most operational environments. For automated graphing and complexity calculation, the technology is language-sensitive; there must be a front-end source parser for each language, with

variants for dialectic differences.

Cyclomatic complexity is usually only moderately sensitive to program change. Other measures (see Complementary Technologies) may be very sensitive. It is common to use several metrics together, either as checks against each other or as part of a calculation set (see Maintainability Index Technique for Measuring Program Maintainability).

Maturity

Cyclomatic complexity measurement, an established but evolving technology, was introduced in 1976. Since that time it has been applied to tens of millions of lines of code in both Department of Defense (DoD) and commercial applications. The resulting base of empirical knowledge has allowed software developers to calibrate measurements of their own software and arrive at some understanding of its complexity. Code graphing and complexity calculation tools are available as part (or as options) of several commercial software environments.

Costs and Limitations

Cyclomatic complexity measurement tools are typically bundled inside commercially-available CASE toolsets. It is usually one of several metrics offered. Application of complexity measurements requires a small amount of training. The fact that a code module has high cyclomatic complexity does not, by itself, mean that it represents excess risk, or that it can or should be redesigned to make it simpler; more must be known about the specific application.

Alternatives

Cyclomatic complexity is one measure of structural complexity. Other metrics bring out other facets of complexity, including both structural and computational complexity, as shown in Table 5.

Table 5: Other Facets of Complexity

Complexity Measurement	Primary Measure of
Halstead Complexity Measures	Algorithmic complexity, measured by counting operators and operands
Henry and Kafura metrics	Coupling between modules (parameters, global variables, calls)
Bowles metrics	Module and system complexity; coupling via parameters and global variables

Troy and Zweben metrics	Modularity or coupling; complexity of structure (maximum depth of structure chart); calls-to and called-by
Ligier metrics	Modularity of the structure chart

Marciniak offers a more complete description of complexity measures and the complexity factors they measure [Marciniak 94].

Complementary Technologies

The following three metrics are specialized measures that are used in specific situations:

1. *Essential complexity*. This measures how much unstructured logic exists in a module (e.g., a loop with an exiting GOTO statement).
2. The program in Figure 6 has no such unstructured logic, so its essential complexity value is one.
3. *Design complexity*. This measures interaction between decision logic and subroutine or function calls.
4. The program in Figure 6 has a design complexity value of 4, which is well within the range of desirability.
5. *Data complexity*. This measures interaction between data references and decision logic.

Other metrics that are "related" to Cyclomatic complexity in general intent are also available in some CASE toolsets.

The metrics listed in Alternatives are also complementary; each metric highlights a different facet of the source code.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Cyclomatic Complexity
Application category	Test (AP.1.4.3) Reapply Software Lifecyle (AP.1.9.3) Reverse Engineering (AP.1.9.4) Reengineering (AP.1.9.5)

Quality measures category	Maintainability (QM.3.1) Testability (QM.1.4.1) Complexity (QM.3.2.1) Structuredness (QM.3.2.3)
Computing reviews category	Software Engineering Metrics (D.2.8) Complexity Classes (F.1.3) Tradeoffs Among Complexity Measures (F.2.3)

References and Information Sources

[Marciniak 94] Marciniak, John J., ed. *Encyclopedia of Software Engineering*, 131-165. New York, NY: John Wiley & Sons, 1994.

[McCabe 89] McCabe, Thomas J. & Butler, Charles W. "Design Complexity Measurement and Testing." *Communications of the ACM 32*, 12 (December 1989): 1415-1425.

[McCabe 94] McCabe, Thomas J. & Watson, Arthur H. "Software Complexity." *Crosstalk, Journal of Defense Software Engineering 7*, 12 (December 1994): 5-9.

[Perry 88] Perry, William E. *A Structured Approach to Systems Testing.* Wellesley, MA: QED Information Sciences, 1988.

[Watson 96] Watson, Arthur H. & McCabe, Thomas J. "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric." [online]. Available WWW, <URL: http://hissa.ncsl.nist.gov/HHRFdata/Artifacts/ITLdoc/235/mccabe.html> (1996).

Current Author/Maintainer

Edmond VanDoren, Kaman Sciences, Colorado Springs

Modifications

12 Jul 2000: Updated reference list

10 Jan 1997 (original)

URL: http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Database Two Phase Commit

Software Technology Roadmap

Status

Advanced

Note

We recommend Three Tier Software Architectures *as prerequisite reading for this technology description.*

Purpose and Origin

Since the 1980s, two phase commit technology has been used to automatically control and monitor commit and/or rollback activities for transactions in a distributed database system. Two phase commit technology is used when data updates need to occur simultaneously at multiple databases within a distributed system. Two phase commits are done to maintain data integrity and *accuracy* within the distributed databases through synchronized locking of all pieces of a transaction. Two phase commit is a proven solution when data integrity in a distributed system is a requirement. Two phase commit technology is used for hotel and airline reservations, stock market transactions, banking applications, and credit card systems. For more details on two phase commit see the *ORACLE7 Server Concept Manual* and *The Performance of Two-Phase Commit Protocols in the Presence of Site Failures* [ORACLE7 92, UCSB 94].

Technical Detail

As shown in Figure 7, applying two phase commit protocols ensures that execution of data transactions are synchronized, either all committed or all rolled back (not committed) to each of the distributed databases.

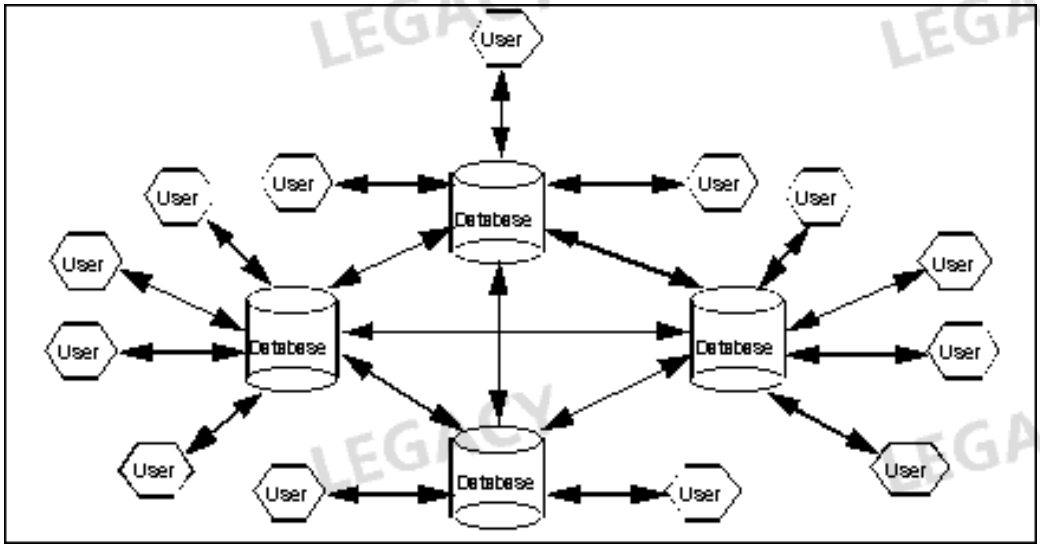


Figure 7: Distributed Databases When Two Phase Commit Happens Simultaneously Through the Network

When dealing with distributed databases, such as in the client/server architecture, distributed transactions need to be coordinated throughout the network to ensure data integrity for the users. Distributed databases using the two phase commit technique update all participating databases simultaneously.

Unlike non-distributed databases (see Figure 8), where a single change is or is not made locally, all participating databases must all commit or all rollback in distributed databases, even if there is a system or network failure at any node. This is how the two phase commit process maintains system data integrity.

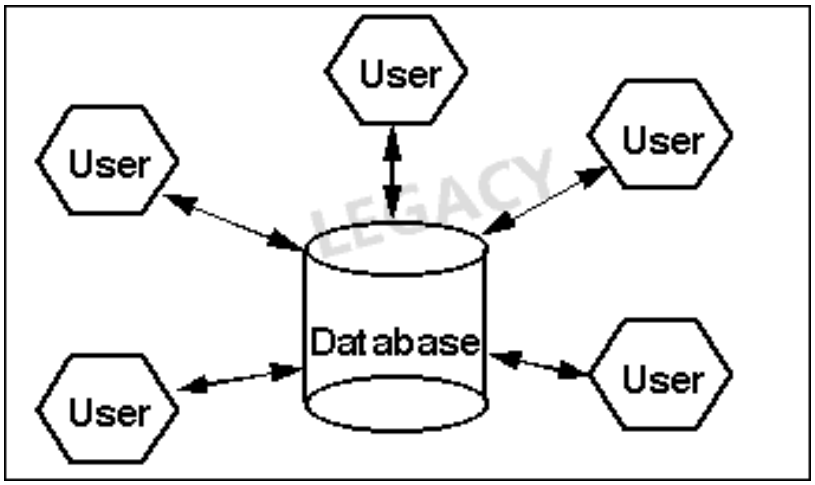


Figure 8: Non-Distributed Databases Make Only Local Updates

Two phase commit has two distinct processes that are accomplished in less than a fraction of a second:

1. The Prepare Phase, where the global coordinator (initiating database) requests that all participants (distributed databases) will promise to commit or rollback the transaction. (Note: Any database could serve as the global coordinator, depending on the transaction.)

2. The Commit Phase, where all participants respond to the coordinator that they are prepared, then the coordinator asks all nodes to commit the transaction. If all participants cannot prepare or there is a system component failure, the coordinator asks all databases to roll back the transaction.

Should there be a machine, network, or software failure during the two phase commit process, the two phase commit protocols will automatically and transparently complete the recovery with no work from the database administrator. This is done through use of pending transaction tables in each database where information about distributed transaction is maintained as they proceed through the two phase commit. Information in the pending transaction table is used by the recovery process to resolve any transaction of questionable status. This information can also be used by the database administrator to override automated recovery procedures by forcing a commit or a rollback to available participating databases.

Usage Considerations

Two phase commit protocols are offered in all modern distributed database products. However, the methods for implementing two phase commits may vary in the degree of automation provided. Some vendors provide a two phase commit implementation that is transparent to the application. Other vendors require specific programming of the calls into an application, and additional programming would be needed should rollback be a requirement; this situation would most likely result in an increase to program cost and schedule.

Maturity

The two phase commit protocol has been used successfully since the 1980s for hotel and airline reservations, stock market transactions, banking applications and credit card systems [Citron 93].

Costs and Limitations

There have been two performance issues with two phase commit:

1. If one database server is unavailable, none of the servers gets the updates. This is correctable if the software administrator forces the commit to the available participants, but if this is a recurring problem the administrator may not be able to keep up, thus causing system and network performance will deteriorate.
2. There is significant demand in network resources as the number of database servers to which data must be distributed increases. This is correctable through network tuning and correctly building the data distribution through database optimization techniques.

Currently, two phase commit procedures are vendor proprietary. There are no standards on how they should be implemented. X/Open has developed a standard that is being implemented in several transaction processing monitors

(see Transaction Processing Monitor Technology), but it has not been adopted by the database vendors [X/Open 96]. Two phase commit proprietary protocols have been published by several vendors.

Alternatives

An alternative to updating distributed databases with a two phase commit mechanism is to update multiple servers using a transaction queuing approach where transactions are distributed sequentially. Distributing transactions sequentially raises the problem of users working with different version of the data. In military usage, this could result in planning sorties for targets that have already been eliminated.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Database Two Phase Commit
Application category	Client-Server (AP.2.1.2.1) Data Management (AP.2.6.1)
Quality measures category	Accuracy (QM.2.1.2.1)
Computing reviews category	Distributed Systems (C.2.4)

References and Information Sources

[Citron 93]	Citron, A., et al. "Two-Phase Commit Optimization and Tradeoffs in the Commercial Environment," 520-529. *Proceedings of the Ninth International Conference on Data Engineering*. Vienna, Austria, April 19-23, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.
[ORACLE7 92]	"Two-Phase Commit," 22-1-22-21. *ORACLE7 Server Concept Manual* (6693-70-1292). Redwood City, CA: Oracle, 1992.
[Schussel 96]	Schussel, G. *Replication, The Next Generation of Distributed Database Technology* [online]. Available WWW <URL: http://www.dciexpo.com/geos/replica.htm> (1996).
[UCSB 94]	*The Performance of Two-Phase Commit Protocols in the Presence of Site Failures* (TRCS94-09). Santa Barbara, CA: University of California, Computer Science Department, April 1994.

[X/Open 96] *X/Open Web Site* [online]. Available WWW <URL: http://www.rdg.opengroup.org/> (1996).

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

David Altieri, GTE

Modifications

20 June 97: updated URLs for [Schussel 96] and [X/Open 96]; changed label for [UCSB 94]
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/dtpc_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Defense Information Infrastructure Common Operating Environment (DII COE)

Software Technology Roadmap

Status

Advanced

Note

We recommend Reference Models, Architectures, Implementations--An Overview as prerequisite reading for this technology description.

Purpose and Origin

The Defense Information Infrastructure (DII) Common Operating Environment (COE) was developed in late 1993. DII COE was designed to eliminate duplication of development (in areas such as mapping, track management, and communication interfaces) and eliminate design incompatibility among Department of Defense (DoD) systems. Conceptually, the COE is designed to reduce program cost and risk through reusing proven solutions and sharing common functionality, rather than developing systems from "scratch" every time. The purpose of DII COE is to field systems with increasing *interoperability*, *reusability*, *portability*, and operational capability, while reducing development time, technical obsolescence, training requirements, and life-cycle cost.

DII COE reuses proven software components contributed by services and programs to provide common Command, Control, Communication, Computer and Intelligence (C4I) functions. For more details on DII COE see the *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification and the DII COE Style Guide* [DII COE 96a, DII COE 96b].

Technical Detail

DII COE technically is

- an architecture (including a set of guidelines and standards)
- a runtime environment
- software (including reusable components)
- a definition for acceptable application programming interfaces

The four major areas are described in further detail below:

1. *Architecture*. The DII COE architecture is fully compliant with the Department of Defense's Technical Architecture for Information Management (TAFIM Reference Model). The DII COE

architecture, presented in Figure 9, is a "plug and play," client/server architecture (implemented and running) that defines COE interfaces and how system components will fit together and interact.

2. *Runtime environment*. A runtime operating environment that includes a standard user system interface, operating system, and windowing environment. The DII COE architecture facilitates a developer in establishing the environment such that there is no conflict with other developers' products.

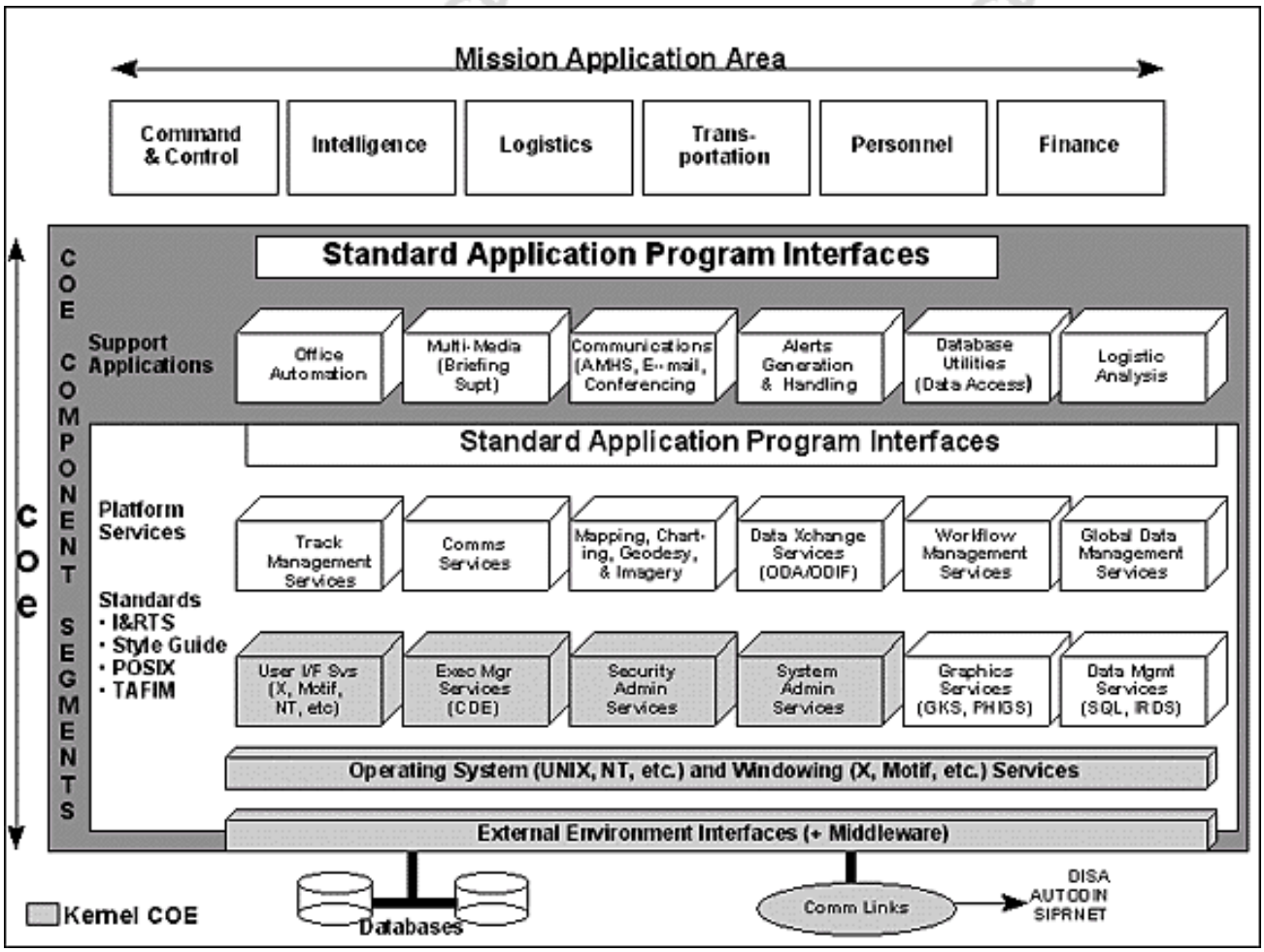


Figure 9: Defense Information Infrastructure Common Operating Environment [DII COE 96]

1. *Software*. A defined set of reusable functions that are already built (available commercially or as government products). Software (with the exception of the operating system and basic windowing software) is packaged in self-contained, manageable units called segments. Segments are the DII COE building block for constructing COE systems. Segments (mission applications and components) may consist of one or more Computer Software Configuration Items (CSCIs). Segments that are part of the reusable (by many mission applications) COE are referred to as COE component segments. Segments are named according to their meaning to operators, rather than internal software structures. Structuring the software into segments allows functionality to be easily added or removed from the target system to meet specific mission and site needs. DII COE databases are divided among segments (as are mission applications) according to the data they contain and the mission applications they support.
2. The kernel COE (light gray shading in Figure 9) is the minimal set of software that is required on every workstation. It includes operating system, windowing services, and external environment interfaces. There are normally five other services also included in the COE kernel: system administration, security administration, executive manager, and two

templates, one for creating privileged operator accounts, and one for creating non-privileged operator accounts. A subset of the kernel COE (defined as Bootstrap COE) is used during initial installation of COE. DII COE is hardware-independent and will run on any open system platform with a standards-based operating system, such as POSIX-compliant UNIX and Windows NT.

3. *APIs*. Two types of Application Programming Interfaces (APIs) are defined for accessing COE segments:
    - public APIs (COE interfaces that will be supported for the COE life cycle)
    - private APIs (interfaces that are supported for a short period of time to allow legacy systems to migrate to full COE compliance)
4. Newly-developed software (segments) must use public APIs to be COE compliant. The incremental implementation strategy for DII COE is to protect legacy system functionality while migrating to fully-compliant COE design by evolving from private APIs to public APIs.

Usage Considerations

There is only one COE available for use by other systems. This COE is currently being used by GCCS (Global Command and Control System) and GCSS (Global Combat Support System). Any system built to the COE infrastructure must access the services using the COE APIs. This improves interoperability between systems because the integration approach, the tool sets, and the segments (software components, not just algorithms) are used by each system [DII COE 96a].

Conceptually, compliance to COE standards ensures that software that is developed or modified for use within COE meets the intended requirements and goals and will evolve with the COE system. Another perspective is that compliance measures the degree to which "plug and play" is possible [Perry 96]. Owners of legacy systems should be familiar with COE compliance requirements to ensure that scoping and planning for future legacy enhancement includes COE requirements and goals.

There are a number of tradeoffs an organization must address when determining evolution of a legacy system to a system that meets COE compliance.

- What are the goals of the legacy system, and will migrating to COE compliance support achievement of the long range goals?
- What level of COE compliance will best and most cost effectively achieve the legacy system's long range goals?
- What is the current state of the legacy system- how compliant is it today?
- Given the current state of the legacy system, what resources are available to begin and follow through on the migration of the code to COE compliance?
- Does the organization want/need to control the legacy system code, and if not, when in the migration to COE is turning it over to DISA desirable?

Based on this analysis, the appropriate level and strategy for compliance can be determined. The four DII COE compliance categories are described in Table 6:

Table 6: DII COE Compliance Categories

Category	Name	Description

1	Runtime Environment	Measures compliance of the proposed segment's fit within the COE executing environment, the amount it reuses COE segments, whether it will run on a COE platform, and whether it will interfere with other segments. This can be done by prototyping within the COE.
2	Style Guide	Measures compliance of the proposed segment's user interface to the Style Guide [DII COE 96b]. This is to ensure that proposed segment will appear consistent with the rest of the COE-based system to minimize training and maintenance cost. Style Guide compliance can be done via a checklist based on the Style Guides requirements.
3	Architectural Compatibility	Measures compliance of the proposed segment's fit within the COE architecture, and the segment's potential life cycle as COE evolves. This can be done by evaluating the segment's use of TAFIM and COE standards and guidelines, and it's internal software structures.
4	Software Quality	Assesses a proposed segment's program risk and software maturity through the use of traditional software metrics. This can be done using measurements such as lines of code and McCabe complexity metrics (see Cyclomatic Complexity).

Category 1 (Runtime) compliance progresses through eight (8) levels of integration from a state of coexistence (agreement on a set of standards and ensure non-interference) with other COE segments, to federated (non-interference when on the same workstation), to fully integrated (share the same software and data). For a segment to be COE compliant, it must be qualified with a category name and compliance level. The following summarizes Category 1's eight levels of compliance; Appendix B of [DII COE 96a] provides a compliance *checklist* for each of the eight levels. Checklists are the current means of assessing progress toward compliance.

- *Standards Compliance Level One* - A proposed segment shares only a common set of standards with the rest of the COE environment, data sharing is undisciplined, and software reuse is minimal other than use of Commercial-Off-The Shelf (COTS) software products. Level 1 allows simultaneous execution of two systems.
- *Network Compliance Level Two* - Two segments will coexist on the same Local Area Network (LAN), but on different CPUs. There is limited data sharing and there may be common user interface "look and feel" if common user interface standards are applied.
- *Workstation Compliance Level Three* - Two applications can reside on the same LAN, share data, and coexist on the same workstation (environmental conflict have been resolved). The kernel COE, or its functional equivalent, resides on the workstation. Some COE components may be reused, but segmenting may not be done. Segments may not interoperate, and do not use the COE services.
- *Bootstrap Compliance Level Four* - Segment formatting is used in all applications. Segments share the bootstrap COE. Some segment conflicts can be automatically checked by the COE system. COE services are not being used. To switch between segments, users may still require separate login accounts. To submit a prototype to DISA for consideration of use, Bootstrap Compliance is required, although these segments will not be fielded or put in the

DISA maintained online library.

- *Minimal COE Compliance Level Five* - All segments share the same kernel COE (equivalent functionality is not acceptable at Level Five). Functionality is available through the COE Executive Manager. Segments may be successfully installed and removed through COE installation tools. Segment descriptor files describe boot, background, and local processes. Segments are registered and available through the online library. Applications appear integrated to the user, but there may be duplication of functionality. Interoperability is not guaranteed. DISA may allow Minimal COE Compliance segments to be installed and used as prototypes at a few sites for evaluation. They can be placed in the library. Currently, Level 5 appears to be the level many legacy systems are targeting.
- *Intermediate COE Compliance Level Six* - Segments use existing account groups, and reuse one or more COE segments. Minor differences may exist between the Style Guide [DII COE 96b] and the segment's graphical user interface implementation.
- *Interoperability Compliance Level Seven* - To ensure interoperability, proposed segments must reuse COE segments, including communication interfaces, message parsers, database tables, track data elements, and logistic services. Public APIs provide access with very few, if any, private APIs. There is no duplicate functionality in the COE segments. DISA requires Interoperability Compliance, for fieldable products and a migration strategy to full COE Compliance (Level 8). A migration strategy is not needed if the proposed segment will be phased out in the near term.
- *Full COE Compliance Level Eight* - All proposed new segments use COE services to the maximum extent possible. New segments are available through the Executive Manager and are completely integrated into the system. All segments fully comply with the Style Guide. [DII COE 96b]. All segments use only public APIs. There is no duplication of functionality any where in the system (as COE or as a mission application).

Two important resources for COE developers and operational sites are the online COE Software Repository System (CSRS) that is used to disseminate and manage software, and the COE Information Server (CINFO) that is used for documentation, meeting notices and general COE information. [DII COE 96a]

Maturity

COE initial proof of concept was created and installed in 1994 with Global Command and Control System (GCCS) Version 1.0. GCCS version 1.1 was used to monitor events during the 1994 Haiti crisis. In 1995, GCCS version 2.0 began fielding to a number of operational sites. There are two systems currently using DII COE: GCCS (developed in 1994 for a near term replacement for World-Wide Military Command and Control System) and GCSS (already fielded at a number of operational CINCs). It is expected that DII COE will be enhanced to include more functionality in such areas as Electronic Commerce/Electronic Data Interchange (EC/EDI), transportation, base support, personnel, health affairs, and finance. [DII COE 96a]

Costs and Limitations

DII COE is relatively new; actual cost, benefit, and risk information is still being collected.

Dependencies

DII COE is dependent of the evolution of TAFIM to ensure compatibility. (see TAFIM Reference Model). An additional dependency could be the Joint Technical Architecture (JTA). The JTA is now being mandated as a set of standards and guidelines for C4I systems, specifically in the area of interoperability, to supersede TAFIM Volume 7, which did not appear to go far enough to ensure

interoperability [JTA 96].

Alternatives

Under conditions where the TAFIM reference model and DII COE compliance is not required, an alternative model would be the Reference Model for Frameworks of Software Engineering Environments (known as the ECMA reference model [ECMA 93]) that is promoted in Europe, and used commercially and world-wide. Commercially-available Hewlett-Packard products use this model [HP 96]. Another alternative would be the Common Object Request Broker Architecture (CORBA) if the design called for object-oriented infrastructure (see Common Object Request Broker Architecture).

Complementary Technologies

Open systems (see COTS and Open Systems--An Overview) would be a complementary technology to DII COE because work done in open system supports the COE goal of achieving interoperable systems.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Defense Information Infrastructure Common Operating Environment
Application category	Software Architecture Models (AP.2.1.1)
Quality measures category	Interoperability (QM.4.1) Reusability (QM.4.4) Portability (QM.4.2)
Computing reviews category	not available

References and Information Sources

[DII COE 96a] *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification (I&RTS)* [online]. Available WWW <URL: http://spider.osfl.disa.mil/dii> (1996).

[DII COE 96b] *DII COE Style Guide*, Version 2.0 [online]. Available WWW <URL: http://spider.osfl.disa.mil/dii> (1996).

[ECMA 93] *Reference Model for Frameworks of Software Engineering Environments*, 3rd Edition (NIST Special Publication 500-211/Technical Report ECMA TR/55). Prepared jointly by NIST and the European Computer Manufacturers Association (ECMA). Washington, DC: U.S. Government Printing Office, 1993.

[HP 96] *Integrated Solutions Catalog for the SoftBench Product Family*. Palo Alto, CA: Hewlett-Packard, 1996.

[JTA 96] U.S. Department of Defense. *Joint Technical Architecture (JTA)* [online]. Available WWW <URL: http://www-jta.itsi.disa.mil/>(1996).

[Perry 96] Perry, Frank. *Defense Information Infrastructure Common Operating Environment* (briefing). April 17, 1996. Arlington, VA: Defense Information Systems Agency.

Current Author/Maintainer

Darleen Sadoski, GTE

Modifications

10 Jan 97 (original)

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Multi-Level Secure One Way Guard with Random Acknowledgment

Software Technology Roadmap

Status

Draft

Note

We recommend Computer System Security--An Overview *as prerequisite reading for this technology description.*

Purpose and Origin

Multi-level secure (MLS) systems are composed of low systems and high systems. Low systems can transmit data to a high system, but high systems cannot transmit data to a low system. That is called *write down* and it is not allowed by multi-level security models, not even to acknowledge (ACK) receipt of data from the low system. This rule exists to prevent a covert timing channel from the high system to the low system. If data integrity and reliable communications are to occur in a system, then messages must be acknowledged. MLS one way guard with random ACK is a form of information flow controls to be imbedded in operational systems that provides a means of acknowledging data without providing a covert path. This technology was first developed (theoretically) in 1993 as an interface between one source and one destination. In 1995 the concept was expanded to address a network of several source low and destination high systems.

Technical Detail

This technology employs a one way guard that buffers a message from a low system and passes it on to the high system. When the high system ACKs the message, the one way guard holds the ACK for a bounded random length of time until passing the ACK to the low system. This destroys any possible covert timing channel as the high system has no control of the timing to the low system. The algorithm to determine the length of time to delay the ACK considers the effect on throughput of delaying multiple sources of data for each destination and the combined throughput to the destination. The algorithm therefore becomes more complex as more sources and destinations are considered. There will be a small negative performance influence on individual messages that could require upgraded interfaces if they are close to capacity. A benefit of

this technology is that it allows reliable transmission over an MLS network because messages that are not ACKed are recognized as not received and can then be retransmitted by the sending system.

Usage Considerations

Sending processes using this technology must account for the maximum possible delay in an ACK before retransmitting a message. Increased buffer space must be provided in the one way guard to hold messages until they can be ACKed. The amount of time and amount of buffer space required are a function of the number of sources and destinations involved and the size and rate of messages. Using this technology in a network of mixed security systems provides for no lost messages and no duplication of messages.

Maturity

This technology is new but is an incremental development of one way security guards that have been in use since the 1960s. This technology has been modeled and prototyped but has not been used in an operational system.

Costs and Limitations

Using this technology will require knowledge of security architectures, the recognition of covert timing channels and means to eliminate them, and Designated Approving Authority (DAA) requirements for assurance.[1]

Dependencies

Successful use of this technology in a system requires that an ACK protocol be employed by the nodes that sends another message only after the last transmitted message has been ACKed.

Alternatives

Other approaches to transferring data through a one way guard to enhance reliability involve multiple transmissions of a message without acknowledging receipt or manual accounting of messages and requests for transmission. These alternatives lead to increased traffic over the network because of duplicate messages or increased operator interaction.

Complementary Technologies

A complimentary technology is covert channel analysis in MLS systems

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Multi-Level Secure One Way Guard with Random Acknowledgment
Application category	System Security (AP.2.4.3)
Quality measures category	Vulnerability (QM.2.1.4.1) Security (QM.2.1.5)
Computing reviews category	Computer-Communications Networks Security and Protection (C.2.0) Security and Protection (K.6.5)

References and Information Sources

[IEEE 95]	*Proceedings of the 1995 IEEE Symposium on Security and Privacy*. Oakland, CA, May 8-10, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

Modifications

10 Jan 97 (original)

Footnotes

[1] The DAA is the security official with the authority to say a system is secure and is permitted to be used.

URL: http://www.sei.cmu.edu/str/descriptions/mlsone_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Network Management--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

In the early 1980s computer networks began to grow and be interconnected. As the size of these networks grew, they became harder to manage and maintain, thus the need for network management was realized. One of the oldest forms of network management is the use of the remote login to monitor or configure a network device; however, today more sophisticated network management tools are available. Network management is a requirement for anyone who wants to control and monitor their networks.

Technical Detail

Functional Areas of Network Management. Network management is the ability to control and monitor a computer network from a central location. The International Organization for Standardization (ISO)[1] defined a conceptual model for describing the key functional areas of network management which are described below [X.700 96]:

Note: In general, network management systems available from vendors today do *not* support all the key functional areas, and in a supported functional area, the coverage may be incomplete even though support is claimed.

- Fault Management: Provides facilities that allow network managers to discover faults in managed devices,[2] the network, and network operation, to determine their cause and to take remedial action. To enable this, fault management provides mechanisms to:
  - Report the occurrence of faults
  - Log reports
  - Perform diagnostic tests
  - Correct faults (possibly automatically)
- Configuration Management: Monitors network configuration information so that the effects of specific hardware and software can be managed and tracked. It may provide the ability to initialize, reconfigure, operate and shut down managed devices.
- Accounting: Measures network utilization of individual users or groups to:
  - Provide billing information

- Regulate users or groups
- Help keep network performance at an acceptable level

- Performance Management: Measures various aspects of network performance including the gathering and analysis of statistical data about the system so that it may be maintained at an acceptable level. Performance management provides the ability to:
  - Obtain the utilization and error rates of network devices
  - Provide a consistent level of performance by ensuring that devices have a sufficient capacity.
- Security Management: Controls access to network resources so that information can not be obtained without authorization by:
  - Limiting access to network resources
  - Providing notification of security breaches and attempts

Network Management Architecture. In general, network management systems have the same basic architecture, as shown in Figure 27.

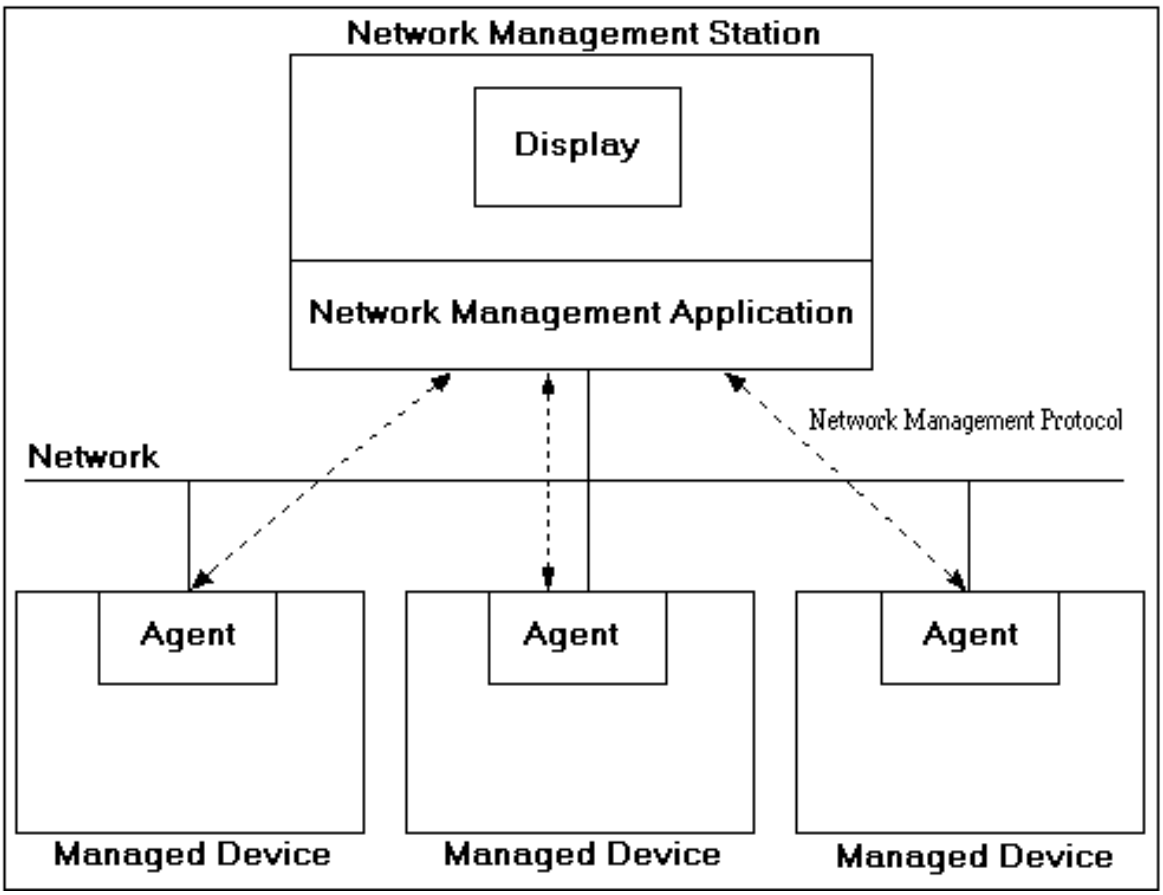


Figure 27: Typical Network Management Architecture [Cisco 96]

The architecture consists of the following elements:

- Network Management Station(s): The network management station[3] runs the network management application[4] that gathers information about managed devices from the management agent[5] which resides within a managed device. The network management application typically must process large amounts of data, react to events, and prepare relevant information for display. It usually has a control console with a GUI interface which allows the operator to view a

graphical representation of the network, control managed devices on the network and program the network management application. Some network management applications can be programmed to react to information collected from management agents and/or set thresholds with the following actions:
    - Perform tests and automatic corrective actions (reconfiguration, shutdown of a managed device)
    - Logging network events
    - Present status information and alerts to operator
- Managed Devices: A managed device can be any type of node residing on a network, such as a computer, printer or router. Managed devices contain a management agent.
- Management agents: Provides information about the managed device to the network management application(s) and may also accept control information.
- Network management protocol: Protocol used by the network management application(s) and the management agent to exchange management information.
- Management Information: The information that is exchanged between the network management application(s) and the management agents that allows the monitoring and control of a managed device.

Network management software (network management applications and agents) is usually based upon a particular network management protocol and the network management capabilities provided with the software are usually based upon the functionality supported by the network management protocol. Most systems use open protocols; however, some network management software is based upon vendor specific proprietary protocols. The selection of network management software is driven by the following factors:

- Network environment (scope and nature of the network)
- Network management requirements
- Cost
- Operating systems involved

The two most common network management protocols are the

- Simple Network Management Protocol
- Common Management Information Protocol

SNMP is by far the most widely used network management protocol and use is widespread in LAN environments. CMIP is used extensively in telecommunication environments, where networks tend to be large and complex.

Usage Considerations

A considerable amount of time is usually required to effectively deploy and learn to use network management software. This is because network managers must be extremely familiar with the network management protocol and the data structures associated with the network management information. Network management protocols and the data structures associated with the network management information are typically complex.

Many network management implementations do not provide support for network

devices which use vendor specific protocols.

A network management system for a small isolated network may not be cost effective or needed. This of course depends on functionality, reliability and performance requirements of the network and attached systems.

Maturity

Network management software often lacks the functionality needed to effectively manage a network. Some of this can be attributed to the deficiencies in the network management protocols.

Numerous network management packages are available from a wide variety of vendors. Some packages are simple and provide network management facilities for a single network, others can be complex and handle multiple types of networks. New products and enhancements to existing network management packages are announced frequently.

Costs and Limitations

Network management systems can be quite expensive, and are often complex. Personnel with specialized training are often required to effectively configure, maintain and operate the network management system.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Network Management
Application category	Protocols (AP.2.2.3) Network Management (AP.2.2.2)
Quality measures category	Openness (QM.4.1.2) Interoperability (QM.4.1) Maintainability (QM.3.1) Scalability (QM.4.3) Security (QM.2.1.5)
Computing reviews category	Network Operations (C.2.3) Distributed Systems (C.2.4)

References and Information Sources

[Cisco 96] *Internetworking Technology Overview / Network Management Basics* [online]. Available WWW <URL: http://cio.cisco.com/univercd/data/doc/cintrnet/ito/55018.htm> (1996).

[Stallings 93] Stallings, William. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards.* Reading, MA: Addison-Wesley, 1993.

[Vallillee 96] Vallillee, Tyler. *SNMP & CMIP: An Introduction To Network Management* [online]. Available WWW <URL: http://www.inforamp.net/~kjvallil/t/snmp.html> (1996).

[X.700 96] *X.700 and Other Network Management Services* [online]. Available WWW <URL: http://ganges.cs.tcd.ie/4ba2/x700/index.html> (1996).

Current Author/Maintainer

Dan Plakosh, SEI

Modifications

9 February 98: Minor modifications

19 June 97 (original)

Footnotes

[1] A voluntary, non-treaty organization founded in 1946 which is responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of the 89 member countries, including ANSI for the U.S.

[2] A managed device is any type of node residing on a network, such as a computer, printer or routers that contain a management agent.

[3] The network management station is the system that hosts the network management application.

[4] The network management application is the application that provides the ability to monitor and control the network.

[5] The network management agent is the software that resides in a managed device that allows the device to be monitored and/or controlled by a network management application.

---

sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/network_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Nonrepudiation in Network Communications

Software Technology Roadmap

Status

Draft

Note

We recommend Computer System Security--An Overview *as prerequisite reading for this technology description.*

Purpose and Origin

The goal of nonrepudiation is to prove that a message has been sent and received. This is extremely important in C4I networks where commands and status must be issued and responded to, in banking networks where financial transactions must be verifiably completed, and in legal networks where signed contracts are transmitted. The Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria (the Red Book) defines the requirement for the military environment. Current technology to accomplish this involves a central authority that verifies and time stamps digital signatures. The technologies for digital signatures have existed since the development of Public Key Cryptography in the late 1970s.

Technical Detail

Three parties are involved in current nonrepudiation schemes: the message sender, the message arbitrator, and the message receiver. The sender creates a message and creates and appends a public key encryption based digital signature to the message. The sender appends identifying data to the message and signs it again. The sender then transmits the message over the network to the arbitrator. The arbitrator verifies the sender's signature and identifying data. The arbitrator then adds a time stamp to the message and signs it. The message is then sent to both the sender and the receiver. The receiver verifies the arbitrator's signature and the sender's signature. The sender verifies the message transmitted by the arbitrator as a copy of the one the sender originally sent. If it does not verify or the sender did not send an original message, the arbitrator is notified immediately. This prevents someone from pretending to be the sender and transmitting a message to the receiver. The arbitrator keeps a record of expired or compromised secret keys to use in the verification process. This whole technology process assures the receiver that the message came

from the indicated source and records the time that the message was sent from the sender to the receiver. The sender can not claim to not have sent the message nor that a lost cryptographic key was used. The message sender, arbitrator, and receiver can be implemented in software in different parts of the network.

Usage Considerations

This technology introduces considerable overhead in the processing of messages. Not only are there creation and verification additions at each end of the transmission but the third party arbitrator processing adds additional overhead and delay. The additional overhead should be considered in the design of the system that uses the technology. This technology may provide the only assured means to identify a source of a message on a network and associate it with a time. The same technology can be used to validate an acknowledgment message.

Maturity

The components of this technology are mature and are used in networks consisting of PCs, workstations, or mainframes.

Costs and Limitations

Using this technology requires knowledge of digital signature algorithms, public key encryption, one-way hashing algorithms and the means of protecting the related keys from inadvertent or malicious compromise.

Dependencies

Successful use of this technology requires the generation and distribution of public keys and the generation and protection of secret keys.

Alternatives

A less secure alternative is to use a time stamp in the senders signature without using a central arbitrator. This is less secure because the sender could claim that someone else sent the message with a stolen or lost key.

Complementary Technologies

Complementary technologies include one-way hashing, digital signatures, and public key cryptography.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Nonrepudiation in Network Communications
Application category	System Security (AP.2.4.3)
Quality measures category	Integrity (QM.2.1.4.1.1) Trustworthiness (QM.2.1.4)
Computing reviews category	Computer-Communications Networks Security and Protection (C.2.0) Security and Protection (K.6.5)

References and Information Sources

[Abrams 95]	Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.
[Schneier 96]	Schneier, Bruce. *Applied Cryptography*. New York, NY: John Wiley & Sons, 1996.
[White 96]	White, Gregory B.; Fisch, Eric A.; & Pooch, Udo W. *Computer System and Network Security.* Boca Raton, FL: CRC Press, 1996.

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/nonrep_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes

- Lists of related topics

Object-Oriented Analysis

Software Technology Roadmap

Status

In review

Purpose and Origin

Object-oriented analysis (OOA) is concerned with developing software engineering requirements and specifications that expressed as a system's object model (which is composed of a population of interacting objects), as opposed to the traditional data or functional views of systems. OOA can yield the following benefits: *maintainability* through simplified mapping to the real world, which provides for less analysis effort, less complexity in system design, and easier verification by the user; *reusability* of the analysis artifacts which saves time and costs; and depending on the analysis method and programming language, *productivity* gains through direct mapping to features of Object-Oriented Programming Languages [Baudoin 96].

Technical Detail

An object is a representation of a real-life entity or abstraction. For example, objects in a flight reservation system might include: an airplane, an airline flight, an icon on a screen, or even a full screen with which a travel agent interacts. OOA specifies the structure and the behavior of the object- these comprise the requirements of that object. Different types of models are required to specify the requirements of the objects. The information or object model contains the definition of objects in the system, which includes: the object name, the object attributes, and object relationships to other objects. The behavior or state model describes the behavior of the objects in terms of the states the object exists in, the transitions allowed between objects, and the events that cause objects to change states. These models can be created and maintained using CASE tools that support representation of objects and object behavior.

OOA views the world as objects with data structures and behaviors and events that trigger operations, or object behavior changes, that change the state of objects. The idea that a system can be viewed as a population of interacting objects, each of which is an atomic bundle of data and functionality, is the foundation of object technology and provides an attractive alternative for the development of complex systems. This is a radical departure from prior methods of requirements specification, such as functional decomposition and structured

analysis and design [Yourdon 79].

Usage Considerations

This technology works best when used in new development. The experiences of Hewlett-Packard in trying to recapture the requirements of legacy systems using OOA suggests that the process can be accomplished only when legacy systems are projected to be long-lived and frequently updated [Malan 95].

Maturity

Numerous OOA methods have been described since 1988. These OOA methods include: Shlaer-Mellor, Jacobson, Coad-Yourdon, and Rumbaugh [Baudoin 96]. The results of implementing these methods range from tremendous successes at AT&T Bell Labs [Kamath 93] to a mixture of successes and partial failures on other projects. AT&T Bell Labs realized benefits from OOA on a large project called the Call Attempt Data Collection System (CADCS). Additionally, they found during the development of two releases of the CADCS that use of the OOA techniques resulted in an 8% reduction in requirements specification time and a 30% reduction in requirements staff effort [Kamath 93]. Other OOA efforts have not been able to reproduce these successes for reasons such as the lack of completed pilot projects, and the lack of formal OOA training [Malan 95].

Costs and Limitations

The use of this technology requires a commitment to formal training in OOA methods. A method of training that has produced desired results is to initiate pilot projects, conduct formal classes, employ OOA mentors, and conduct team reviews to train properly both the analysis and development staff as well as the program management team [Kamath 93]. There are almost no reported successes using OOA methods on the first application without this type of training program [Kamath 93]. Projects with initial and continuing OOA training programs realize that the benefits of this technology depend upon the training and experience levels of their staffs. Purchase of CASE tools that support object-oriented methods may significantly enhance OOA- this is another cost to consider.

Alternatives

Alternative technologies that are used for developing software engineering requirements and specifications include functional decomposition, essential systems analysis, and structured analysis [Yourdon 79].

Complementary Technologies

There is a strong relationship between OOA and other object-oriented technologies (see Object-Oriented Database, Object-Oriented Design, and

Object-Oriented Programming Languages). This is especially true of object-oriented design- certain object-oriented methods combine particular analysis and design methods that work well together. In fact, the seamless use of objects throughout the analysis, design, and programming phases provides the greatest benefit. Use of OOA alone, without transition into OOD, would be a severely limited approach.

Combining object-oriented methods with Cleanroom (with its emphasis on rigor, formalisms, and reliability) can define a process capable of producing results that are reusable, predictable, and high-quality. Thus, object-oriented methods can be used for front-end domain analysis and design, and Cleanroom can be used for life-cycle application engineering [Ett 96].

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Object-Oriented Analysis
Application category	Define and Develop Requirements (AP.1.2.2.1) Analyze Functions (AP.1.2.1.1) Reengineering (AP.1.9.5)
Quality measures category	Maintainability (QM.3.1) Reusability (QM.4.4)
Computing reviews category	Software Engineering Requirements and Specifications (D.2.1) Software Engineering Tools and Techniques (D.2.2) Software Engineering Design (D.2.10)

References and Information Sources

[Baudoin 96] Baudoin, Claude & Hollowell, Glenn. *Realizing the Object-Oriented Lifecycle*. Upper Saddle River, NJ: Prentice Hall, 1996.

[Embley 95] Embley, David W.; Jackson, Robert B.; & Woodfield, Scott N. "OO Systems Analysis: Is it or Isn't it?" *IEEE Software 12*, 2 (July 1995): 19-33.

[Ett 96] Ett, William & Trammell, Carmen. *A Guide to Integration of Object-Oriented Methods and Cleanroom Software Engineering* [online]. Originally available WWW <URL: http://www.asset.com/stars/loral/cleanroom/oo/guidhome.htm> (1996).

[Kamath 93] Kamath, Y. H.; Smilan, R. E.; & Smith, J. G. "Reaping Benefits With Object-Oriented Technology." *AT&T Technical Journal 72*, 5 (September/October 1993): 14-24.

[Malan 95] Malan, R.; Coleman, D.; & Letsinger, R. "Lessons Learned from the Experiences of Leading-Edge Object Technology Projects in Hewlett-Packard," 33-46. *Proceedings of Tenth Annual Conference on Object-Oriented Programming Systems Languages and Applications*. Austin, TX, October 15-19, 1995. Palo Alto, CA: Hewlett-Packard, 1995.

[Yourdon 79] Yourdon, E. & Constantine, L. *Structured Design*. Englewood Cliffs, NJ: Prentice Hall, 1979.

Current Author/Maintainer

Mike Bray, Lockheed-Martin Ground Systems

Modifications

27 Oct 97: updated URL for [Ett 96]
10 Jan 97: original

URL: http://www.sei.cmu.edu/str/descriptions/ooanalysis_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes

- Lists of related topics

Object-Oriented Database

Software Technology Roadmap

Status

In review

Purpose and Origin

Object-oriented databases (OODBs) evolved from a need to support object-oriented programming and to reap the benefits, such as system *maintainability*, from applying object orientation to developing complex software systems. The first OODBs appeared in the late 1980s. Martin provides a complete list of these early OODBs [Martin 93]. OODBs are based on the object model and use the same conceptual models as Object-Oriented Analysis, Object-Oriented Design and Object-Oriented Programming Languages. Using the same conceptual model simplifies development; improves communication among users, analysts, and programmers; and lessens the likelihood of errors [Martin 93].

Technical Detail

OODBs are designed for the purpose of storing and sharing objects; they are a solution for persistent object handling. Persistent data are data that remain after a process is terminated.

There is no universally-acknowledged standard for OODBs. There is, however, some commonality in the architecture of the different OODBs because of three necessary components: object managers, object servers, and object stores. Applications interact with object managers, which work through object servers to gain access to object stores.

OODBs provide the following benefits:

- OODBs allow for the storage of complex data structures that can not be easily stored using conventional database technology.
- OODBs support all the persistence necessary when working with object-oriented languages.
- OODBs contain active object servers that support not only the distribution of data but also the distribution of work (in this context, relational database management systems (DBMS) have limited capabilities) [Vorwerk 94].

In addition, OODBs were designed to be well integrated with object-oriented programming languages such as C++ and Smalltalk. They use the same object model as these languages. With OODBs, the programmer deals with transient (temporary) and persistent (permanent) objects in a uniform manner. The persistent objects are in the OODB, and thus the conceptual walls between programming and database are removed. As stated earlier, the employment of a unified conceptual model greatly simplifies development [Tkach 94].

Usage Considerations

The type of database application should dictate the choice of database management technology. In general, database applications can be categorized into two different applications:

1. *Data collection applications* focus on entering data into a database and providing queries to obtain information about the data. Examples of these kinds of database applications are accounts payable, accounts receivable, order processing, and inventory control. Because these types of applications contain relatively simple data relationships and schema design, relational database management systems (RDBMs) are better suited for these applications.
2. *Information analysis applications* focus on providing the capability to navigate through and analyze large volumes of data. Examples of these applications are CAD/CAM/CAE, production planning, network planning, and financial engineering. These types of applications are very dynamic and their database schemas are very complex. This type of application requires a tightly-coupled language interface and the ability to handle the creation and evolution of schema of arbitrary complexity without a lot of programmer intervention. Object-oriented databases support these features to a great degree and are therefore better suited for the information analysis type of applications [Desanti 94].

OODBs are also used in applications handling BLOBs (binary large objects) such as images, sound, video, and unformatted text. OODBs support diverse data types rather than only the simple tables, columns and rows of relational databases.

Maturity

Claims have been made that OODBs are not used in mainstream applications, are not scalable, and represent an immature technology [Object 96]. Two examples to the contrary include the following:

- Northwest Natural Gas uses an OODB for a customer information system. The system stores service information on 400,000 customers and is accessed by 250 customer service representatives in seven district offices in the Pacific Northwest.
- Ameritech Advanced Data Services uses an OODB for a comprehensive management information system that currently includes accounting, order entry, pricing, and pre-sales support and is accessed by more than 200

people dispersed in a five state region.

Both of these applications are mainstream and represent databases well over a gigabyte in size; this highlights the fact that OODBs do work well with large numbers of users in large applications [Object 96].

Costs and Limitations

The costs of implementing OODB technology are dependent on the required platforms and numbers of licenses required. The costs of the actual OODB software are comparable to relational database management systems on similar platforms. The use of OODBs requires an educational change among the software developers and database maintainers and requires a corporate commitment to formal training in the proper use of the OODB features.

Alternatives

An alternative is relational database management systems (RDBMs).

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Object-Oriented Database
Application category	Database Design (AP.1.3.2) Database Administration (AP.1.9.1) Databases (AP.2.6)
Quality measures category	Maintainability (QM.3.1)
Computing reviews category	Database Management (H.2)

References and Information Sources

[Desanti 94] Desanti, Mike & Gomsi, Jeff. "A Comparison of Object and Relational Database Technologies." *Object Magazine 3*, 5 (January 1994): 51-57.

[Martin 93] Martin, James. *Principles of Object-Oriented Analysis and Design*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[Object 96] "Focus on ODBMS Debunking the Myths." *Object Magazine 5*, 9 (February 1996): 21-23.

[Tkach 94] Tkach, Daniel & Puttick, Richard. *Object Technology in Application Development*. Redwood City, CA: Benjamin/ Cummings Publishing Company, 1994.

[Vorwerk 94] Vorwerk, Raymond. "Towards a True OBBMS." *Object Magazine 3*, 5 (January 1994): 38-39.

Current Author/Maintainer

Mike Bray, Lockheed-Martin Ground Systems

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/oodatabase_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Object-Oriented Design

Software Technology Roadmap

Status

In review

Purpose and Origin

Object-oriented design (OOD) is concerned with developing an object-oriented model of a software system to implement the identified requirements. Many OOD methods have been described since the late 1980s. The most popular OOD methods include Booch, Buhr, Wasserman, and the HOOD method developed by the European Space Agency [Baudoin 96]. OOD can yield the following benefits: *maintainability* through simplified mapping to the problem domain, which provides for less analysis effort, less complexity in system design, and easier verification by the user; *reusability* of the design artifacts, which saves time and costs; and productivity gains through direct mapping to features of Object-Oriented Programming Languages [Baudoin 96].

Technical Detail

OOD builds on the products developed during Object-Oriented Analysis (OOA) by refining candidate objects into classes, defining message protocols for all objects, defining data structures and procedures, and mapping these into an object-oriented programming language (OOPL) (see Object-Oriented Programming Languages). Several OOD methods (Booch, Shlaer-Mellor, Buhr, Rumbaugh) describe these operations on objects, although none is an accepted industry standard. Analysis and design are closer to each other in the object-oriented approach than in structured analysis and design. For this reason, similar notations are often used during analysis and the early stages of design. However, OOD requires the specification of concepts nonexistent in analysis, such as the types of the attributes of a class, or the logic of its methods.

Design can be thought of in two phases. The first, called high-level design, deals with the decomposition of the system into large, complex objects. The second phase is called low-level design. In this phase, attributes and methods are specified at the level of individual objects. This is also where a project can realize most of the reuse of object-oriented products, since it is possible to guide the design so that lower-level objects correspond exactly to those in existing object libraries or to develop objects with reuse potential. As in OOA, the OOD artifacts are represented using CASE tools with object-oriented terminology.

Usage Considerations

OOD techniques are useful for development of large complex systems. AT&T Bell Labs used OOD and realized the benefits of reduced product development time and increased reuse of both code and analysis/design artifacts on a large project called the Call Attempt Data Collection System (CADCS). This large project consisted of over 350,000 lines of C++ code that ran on a central processor with over 100 remote systems distributed across the United States. During the development of two releases of the CADCS they found that use of the OOD techniques resulted in a 30% reduction in development time and a 20% reduction in development staff effort as compared to similarly sized projects using traditional software development techniques [Kamath 93]. However, these successes were realized only after thorough training and completion of three- to six-month pilot projects by their development staff [Kamath 93].

Experiences from other organizations show costly learning curves and few productivity improvements without thoroughly-trained designers and developers. Additionally, OOD methods must be adapted to the project since each method contains object models that may be too costly, or provide little value, for use on a specific project [Malan 95].

The maximum impact from OOD is achieved when used with the goal of designing reusable software systems. For objects without significant reuse potential, OOD techniques were more costly than traditional software development methodologies. This was because of the costs associated with developing objects and the software to implement these objects for a one-time use [Maring 96].

Maturity

Many OOD methods have been used in industry since the late 1980s. OOD has been used worldwide in many commercial, Department of Defense (DoD), and government applications. There exists a wealth of documentation and training courses for each of the various OOD methods, along with commercially-available CASE tools with object-oriented extensions that support these OOD methods.

Costs and Limitations

One reason for the mixed success reviews on OOD techniques is that the use of this technology requires a corporate commitment to formal training in the OOD methods and the purchase of CASE tools with capabilities that support these methods. The method of training that produces the best results is to initiate pilot projects, conduct formal classes, employ OOD mentors, and conduct team reviews to train properly both the analysis and development staff as well as the program management team [Kamath 93]. There are few, if any, reported successes using OOD methods on the first application without this type of training program [Maring 96]. Projects with initial and continuing OOD training programs realize that the benefits of this technology depend upon the training

and experience levels of their staffs.

Dependencies

The use of OOD technology requires the development of object requirements using OOA techniques, and CASE tools to support both the drawing of objects and the description of the relationships between objects. Also, the final steps of OOD, representing classes and objects in programming constructs, are dependent on the object-oriented programming language (OOPL) chosen. For example, if the OOPL is Ada 95, a package-based view of the implementation should be used; if C++ is the OOPL, then a class-based view should be used. These different views require different technical design decisions and implementation considerations.

Alternatives

An alternative technology that can be used for developing a model of a software system design to implement the identified requirements is a traditional design approach such as Yourdon and Constantine's Structured Design [Yourdon 79]. This method, used successfully for many different types of applications, is centered around design of the required functions of a system and does not lend itself to object orientation.

Complementary Technologies

Combining object-oriented methods with Cleanroom (with its emphasis on rigor, formalisms, and reliability) can define a process capable of producing results that are reusable, predictable, and high-quality. Thus, object-oriented methods can be used for front-end domain analysis and design, and Cleanroom can be used for life-cycle application engineering [Ett 96].

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Object-Oriented Design
Application category	Detailed Design (AP.1.3.5) Reengineering (AP.1.9.5)
Quality measures category	Maintainability (QM.3.1) Reusability (QM.4.4)

Computing reviews category	Object-Oriented Programming (D.1.5) Software Engineering Design (D.2.10)

References and Information Sources

[Baudoin 96] Baudoin, Claude & Hollowell, Glenn. *Realizing the Object-Oriented Lifecycle*. Upper Saddle River, NJ: Prentice Hall, 1996.

[Ett 96] Ett, William & Trammell, Carmen. *A Guide to Integration of Object-Oriented Methods and Cleanroom Software Engineering* [online]. Originally available WWW <URL: http://www.asset.com/stars/loral/cleanroom/oo/guidhome.htm> (1996).

[Kamath 93] Kamath, Y. H.; Smilan, R. E.; & Smith, J. G. "Reaping Benefits With Object-Oriented Technology." *AT&T Technical Journal 72*, 5 (September/October 1993): 14-24.

[Malan 95] Malan, R.; Coleman, D.; & Letsinger, R. "Lessons Learned from the Experiences of Leading-Edge Object Technology Projects in Hewlett-Packard," 33-46. *Proceedings of Tenth Annual Conference on Object-Oriented Programming Systems Languages and Applications*. Austin, TX, October 15-19, 1995. Palo Alto, CA: Hewlett-Packard, 1995.

[Maring 96] Maring, B. "Object-Oriented Development of Large Applications." *IEEE Software 13*, 3 (May 1996): 33-40.

[Yourdon 79] Yourdon, E. & Constantine, L. *Structured Design*. Englewood Cliffs, NJ: Prentice Hall, 1979.

Current Author/Maintainer

Mike Bray, Lockheed-Martin Ground Systems

Modifications

27 Oct 97: updated URL for [ETT 96]
10 Jan 97: original

URL: http://www.sei.cmu.edu/str/descriptions/oodesign_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Object-Oriented Programming Languages

Software Technology Roadmap

Status

In review

Purpose and Origin

Object-oriented programming languages (OOPLs) are the natural choice for implementation of an Object-Oriented Design because they directly support the object notions of classes, inheritance, information hiding, and dynamic binding. Because they support these object notions, OOPLs make an object-oriented design easier to implement [Baudoin 96]. An object-oriented system programmed with an OOPL results in less complexity in the system design and implementation, which can lead to an increase in *maintainability* [Baudoin 96]. The genesis of this technology dates back to the early 1960s with the work of Nygaard and Dahl in the development of the first object-oriented language called Simula 67. Research progressed through the 1970s with the development of Smalltalk at Xerox. Current OOPLs include C++, Objective C, Smalltalk, Eiffel, Common LISP Object System (CLOS), Object Pascal, Java, and Ada 95 [Baudoin 96].

Technical Detail

Object-oriented (OO) applications can be written in either conventional languages or OOPLs, but they are much easier to write in languages especially designed for OO programming. OO language experts divide OOPLs into two categories, hybrid languages and pure OO languages. Hybrid languages are based on some non-OO model that has been enhanced with OO concepts. C++ (a superset of C), Ada 95, and CLOS (an object-enhanced version of LISP) are hybrid languages. Pure OO languages are based entirely on OO principles; Smalltalk, Eiffel, Java, and Simula are pure OO languages.

In terms of numbers of applications, the most popular OO language in use is C++. One advantage of C++ for commercial use is its syntactical familiarity to C, which many programmers already know and use; this lowers training costs. Additionally, C++ implements all the concepts of object orientation, which include classes, inheritance, information hiding, polymorphism, and dynamic binding. One disadvantage of C++ is that it lacks the level of polymorphism and dynamics most OO programmers expect. Ada 95 is a reliable, standardized language well-suited for developing large, complex systems that are reliable [Tokar 96].

The major alternative to C++ or Ada 95 is Smalltalk. Its advantages are its consistency and flexibility. Its disadvantages are its unfamiliarity (causing an added training cost for developers), and its inability to work with existing systems (a major benefit of C++) [Tokar 96].

Usage Considerations

OOPLs are strongly recommended to complete the implementation of Object-Oriented Analysis (OOA) and Object-Oriented Design (OOD) technologies. AT&T Bell Labs used OOD and OOPLs and realized the benefits of reduced product development time and increased reuse of both code and analysis/design artifacts on a large project called Call Attempt Data Collection System (CADCS). This large project consisted of over 350,000 lines of C++ code that ran on a central processor with over 100 remote systems distributed across the United States. During the development of two releases of the CADCS, the use of the OOD techniques and subsequent implementation in OOPL resulted in an 30% reduction in development time and a 20% reduction in development staff effort as compared to similarly-sized projects using traditional software development techniques and languages [Kamath 93].

Organizations such as Bell Labs have found that through the introduction of OO programming techniques in pilots and training courses, the developers were able to learn properly and experiment with the OOPL constructs. This resulted in increased object-oriented expertise such that much of the CADCS software (objects) was reused on a similar project [Kamath 93].

OOPLs such as Ada 95 and C++ can also be used to develop traditional non-object-oriented software. These applications can be developed by avoiding the use of the object-oriented language features. There are many commercial, Department of Defense (DoD), and government applications of this type in existence today.

For applications where OOPL code is to be generated by a CASE tool, developers must decide which programming language to generate: C++, Ada 95, Smalltalk, Java, or CLOS. The choice of an OOPL can limit the choices of CASE tools because the tools may not support the chosen language. However, if language generation is not a consideration, then CASE tools can be chosen based on features and design capabilities without regard to the OOPL chosen for implementation.

Since different OOPLs support different levels of 'objectiveness' (e.g., inheritance), different OOD constructs may or may not map directly to OOPL constructs. Therefore, the choice of an OOPL is affected by a design captured using OOD techniques. Where OOD is not present, any OOPL can be used, depending upon the training of the developers.

Maturity

OOPLs have been used worldwide on many commercial, DoD, and government

applications/projects. There exists a wealth of documentation and training courses for each of the various OOPLs.

Costs and Limitations

The use of OOPL technology requires a corporate commitment to formal training in the proper use of the OOPL features and the purchase of the language compiler. The costs of completely training a development staff implies that the insertion of this technology should be undertaken only on new developments (instead of maintenance of legacy systems), and only after pilot project(s) are successfully completed [Malan 95].

Alternatives

Both object-oriented and non-object-oriented applications can be written in either traditional languages or OOPLs. To fully realize the benefits of an object orientation, it is much easier to write the implementations in languages especially designed for OO programming.

Complementary Technologies

Combining object-oriented methods with Cleanroom (with its emphasis on rigor, formalisms, and reliability) can define a process capable of producing results that are reusable, predictable, and high-quality. Thus, OOPLs can be used for implementation of an object-oriented design and Cleanroom can be used for life-cycle application engineering.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Object-Oriented Programming Languages
Application category	Programming Language (AP.1.4.2.1)
Quality measures category	Maintainability (QM.3.1)
Computing reviews category	Object-Oriented Programming (D.1.5) Programming Language Classifications (D.3.2)

References and Information Sources

[Baudoin 96]	Baudoin, Claude & Hollowell, Glenn. *Realizing the Object-Oriented Lifecycle*. Upper Saddle River, NJ: Prentice Hall, 1996.
[Kamath 93]	Kamath, Y. H.; Smilan, R. E.; & Smith, J. G. "Reaping Benefits With Object-Oriented Technology." *AT&T Technical Journal 72*, 5 (September/October 1993): 14-24.
[Malan 95]	Malan, R.; Coleman, D.; & Letsinger, R. "Lessons Learned from the Experiences of Leading-Edge Object Technology Projects in Hewlett-Packard," 33-46. *Proceedings of Tenth Annual Conference on Object-Oriented Programming Systems Languages and Applications*. Austin, TX, October 15-19, 1995. Palo Alto, CA: Hewlett-Packard, 1995.
[Tokar 96]	Tokar, Joyce L. "Ada 95: The Language for the 90's and Beyond." *Object Magazine 6*, 4 (June 1996): 53-56.

Current Author/Maintainer

Mike Bray, Lockheed-Martin Ground Systems

Modifications

10 Jan 97: (original)

URL: http://www.sei.cmu.edu/str/descriptions/oopl_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Object Request Broker

Software Technology Roadmap

Status

Complete

Note

We recommend Middleware, as prerequisite reading for this technology description.

Purpose and Origin

An object request broker (ORB) is a middleware technology that manages communication and data exchange between objects. ORBs promote *interoperability* of distributed object systems because they enable users to build systems by piecing together objects- from different vendors- that communicate with each other via the ORB [Wade 94]. The implementation details of the ORB are generally not important to developers building distributed systems. The developers are only concerned with the object interface details. This form of information hiding enhances system *maintainability* since the object communication details are hidden from the developers and isolated in the ORB [Cobb 95].

Technical Detail

ORB technology promotes the goal of object communication across machine, software, and vendor boundaries. The relevant functions of an ORB technology are

- interface definition
- location and possible activation of remote objects
- communication between clients and object

An object request broker acts as a kind of telephone exchange. It provides a directory of services and helps establish connections between clients and these services [CORBA 96, Steinke 95]. Figure 21 illustrates some of the key ideas.

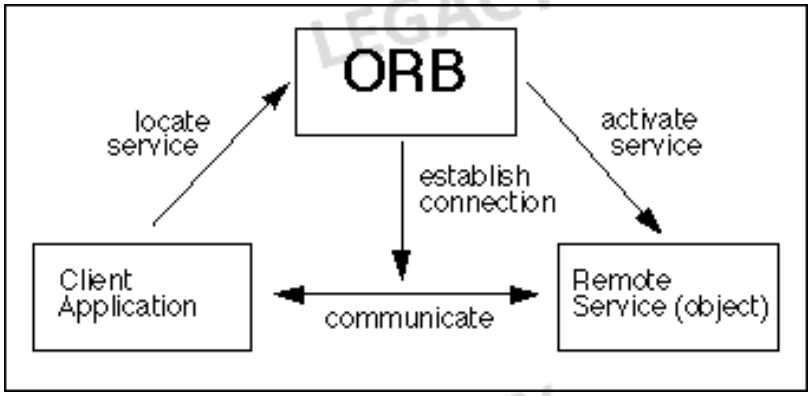


Figure 21: Object Request Broker

The ORB must support many functions in order to operate consistently and effectively, but many of these functions are hidden from the user of the ORB. It is the responsibility of the ORB to provide the illusion of locality, in other words, to make it appear as if the object is local to the client, while in reality it may reside in a different process or machine [Reddy 95]. Thus the ORB provides a framework for cross-system communication between objects. This is the first technical step toward interoperability of object systems.

The next technical step toward object system interoperability is the communication of objects across platforms. An ORB allows objects to hide their implementation details from clients. This can include programming language, operating system, host hardware, and object location. Each of these can be thought of as a "transparency,"[1] and different ORB technologies may choose to support different transparencies, thus extending the benefits of object orientation across platforms and communication channels.

There are many ways of implementing the basic ORB concept; for example, ORB functions can be compiled into clients, can be separate processes, or can be part of an operating system kernel. These basic design decisions might be fixed in a single product; or there might be a range of choices left to the ORB implementer.

There are two major ORB technologies:

- The Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) specification
- Microsoft's Component Object Model (see Component Object Model (COM), DCOM, and Related Capabilities)

An additional, newly-emerging ORB model is Remote Method Invocation (RMI); this is specified as part of the Java language/virtual machine. RMI allows Java objects to be executed remotely. This provides ORB-like capabilities as a native extension of Java [RMI 97].

A high-level comparison of ORB technologies is available in Table 8. Details are available in the referenced technology descriptions.

Usage Considerations

Successful adoption of ORB technology requires a careful analysis of the current and future software architectural needs of the target application and analysis of how a particular ORB will satisfy those needs [Abowd 96]. Among the many things to consider are platform availability, support for various programming languages, as well as implementation choices and product performance parameters. After performing this analysis, developers can make informed decisions in choosing the ORB best suited for their application's needs.

Table 8: Comparison of ORB Technologies

ORB	Platform Availability	Applicable to	Mechanism	Implementations
COM/ DCOM	originally PC platforms, but becoming available on other platforms	"PC-centric" distributed systems architecture	APIs to proprietary system[2]	one[3]
CORBA	platform-independent and interoperability among platforms	general distributed system architecture	specification of distributed object technology	many[4]
Java/ RMI	wherever Java virtual machine (VM) executes	general distributed system architecture and Web-based Intranets	implementation of distributed object technology	various[5]

Maturity

As shown in Table 8, there are a number of commercial ORB products available. ORB products that are not compliant with either CORBA or OLE also exist; however, these tend to be vendor-unique solutions that may affect system interoperability, portability, and maintainability.

Major developments in commercial ORB products are occurring, with life cycles seemingly lasting only four to six months. In addition, new ORB technology (Java/RMI) is emerging, and there are signs of potential "mergers" involving two of the major technologies. The continued trend toward Intranet- and Internet-based applications is another stimulant in the situation. Whether these

commercial directions are fully technically viable and will be accepted by the market is unknown.

Given the current situation and technical uncertainty, potential users of ORB technologies need to determine

- what new features ORB technologies add beyond technologies currently in use in their organizations
- the potential benefits from using these new features
- the key risks involved in adopting the technology as a whole
- how much risk is acceptable to them

One possible path would be to undertake a disciplined and "situated" technology evaluation. Such an evaluation, as described by Brown and Wallnau, focuses on evaluating so-called "innovative" technologies and can provide technical information for adoption that is relative to the current/existing approaches in use by an organization [Brown 96, Wallnau 96]. Such a technology evaluation could include pilot projects focusing on model problems pertinent to the individual organization.

Costs and Limitations

The license costs of the ORB products from the vendors listed above are dependent on the required operating systems and the types of platform. ORB products are available for all major computing platforms and operating systems.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Object Request Broker
Application category	Client/Server (AP.2.1.2.1), Client/Server Communication (AP.2.2.1)
Quality measures category	Interoperability (QM.4.1), Maintainability (QM.3.1)
Computing reviews category	Distributed systems (C.2.4), Object-Oriented programming (D.1.5)

References and Information Sources

[Abowd 96] Abowd, Gregory, et al. "Architectural Analysis of ORBs." *Object Magazine 6*, 1 (March 1996): 44-51.

[Brown 96] Brown, A. & Wallnau, K. "A Framework for Evaluating Software Technology." *IEEE Software 13*, 5 (September 1996): 39-49.

[Cobb 95] Cobb, Edward E. "TP Monitors and ORBs: A Superior Client/ Server Alternative." *Object Magazine 4*, 9 (February 1995): 57-61.

[CORBA 96] *The Common Object Request Broker: Architecture and Specification*, Version 2.0. Framingham, MA: Object Management Group, 1996. Also available [online] WWW <URL: http://www.omg.org> (1996).

[Reddy 95] Reddy, Madhu. "ORBs and ODBMSs: Two Complementary Ways to Distribute Objects." *Object Magazine 5*, 3 (June 1995): 24-30.

[RMI 97] Remote Method Invocation [online]. Available WWW <URL: http://java.sun.com/products/jdk/1.1/docs/guide/rmi> (1997).

[Steinke 95] Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

[Tkach 94] Tkach, Daniel & Puttick, Richard. *Object Technology in Application Development*. Redwood City, CA: Benjamin/ Cummings Publishing Company, 1994.

[Wade 94] Wade, Andrew E. "Distributed Client-Server Databases." *Object Magazine 4*, 1 (April 1994): 47-52.

[Wallnau 96] Wallnau, Kurt & Wallace, Evan. "A Situated Evaluation of the Object Management Group's (OMG) Object Management Architecture (OMA)," 168-178. *Proceedings of the OOPSLA'96*. San Jose, CA, October 6-10, 1996. New York, NY: ACM, 1996. Presentation available [online] FTP. <URL: ftp://ftp.sei.cmu.edu/pub/corba/OOPSLA/present> (1996).

Current Author/Maintainer

Kurt Wallnau, SEI
John Foreman, SEI

External Reviewers

Ed Morris, SEI
Richard Soley, VP, Chief Technical Officer, Object Management Group

Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM; added notes to Table 8
9 April 97: minor edits and reorganization; no meaningful content changes
10 Jan 97 (original): Mike Bray, Lockheed-Martin Ground Systems

Footnotes

[1] transparency: making something invisible to the client

[2] COM/DCOM specifications have been turned over to the Open Group, but the outcome of this standardization activity remains unclear.

[3] Microsoft maintains the only implementation of PC platforms, and is working closely with selected vendors to migrate technology to alternate platforms.

[4] Examples include ORBIX by IONA Technology, NEO by SunSoft, VisiBroker by VisiGenic, PowerBroker by Expersoft, SmallTalkBroker by DNS Technologies, Object Director by Fujitsu, DSOM by IBM, DAIS by ICL, SORBET by Siemens Nixdorf, and NonStop DOM by Tandem.

[5] Implementations of the Java VM have been ported to various platforms.

URL: http://www.sei.cmu.edu/str/descriptions/orb_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Organization Domain Modeling

Software Technology Roadmap

Status

Complete

Note

We recommend Domain Engineering and Domain Analysis *as prerequisite reading for this technology description.*

Purpose and Origin

Organization domain modeling (ODM) was developed to provide a formal, manageable, and repeatable approach to domain engineering. The ODM method evolved and was subsequently formalized by Mark Simos (Organon Motives, Inc.) with collaboration and sponsorship from Hewlett-Packard Company, Lockheed-Martin,[1] and the DARPA STARS[2] program [Simos 96]. ODM affects the *maintainability*, *understandability*, and *reusability* characteristics of a system or family of systems.

Technical Detail

ODM was developed and refined as part of the overall reuse/product line approaches developed under the STARS program. The STARS reuse approach decomposes reuse technologies into several levels or layers of abstraction, specifically: Concepts, Processes, Methods, and Tools. An example of a "concept" is the Conceptual Framework for Reuse Processes (CFRP), a conceptual foundation and framework for understanding domain-specific reuse in terms of the processes involved [STARS 93]. An example of a "process" is the Reuse-Oriented Software Evolution (ROSE) process model, which is based on the CFRP life-cycle process model; it partitions software development into domain engineering, asset management, and application engineering; and emphasizes the role of reuse in software evolution. ODM is an example of a "method" compatible with the CFRP framework.

The primary goal of ODM is the systematic transformation of artifacts (e.g., requirements, design, code, tests, and processes) from multiple legacy systems into assets that can be used in multiple systems. The method can also be applied to requirements for new systems; the key element is to ground domain

models empirically by explicit consideration of multiple exemplars, which determine the requisite range of variability that the models must encompass. ODM stresses the use of legacy artifacts and knowledge as a source of domain knowledge and potential resources for reengineering/reuse. However, one of its objectives is to avoid embedding hidden constraints that may exist in legacy systems into the domain models and assets.

Domain Engineering and Domain Analysis identifies three areas where domain analysis methods can be differentiated. Distinguishing features for ODM are:

Primary product of the analysis. The result of ODM is a knowledge representation framework populated with a domain architecture and a flexible asset base. It can be thought of as a reuse library designed to support systematic reuse in a prescribed context; however, the method supports the use of diverse implementation techniques such as generators in the asset base.

Focus of analysis

- ODM is structured in terms of a core domain engineering life cycle, which is distinct from and orthogonal to the system engineering life cycle. The ODM life cycle is divided into three phases:
  - plan domain: selecting, scoping, and defining target domains
  - model domain: modeling the range of variability that can exist within the scope of the domain
  - engineer asset base: engineering an asset base that satisfies some subset of the domain variability, based on the needs of specific target customers [Simos 96]
- *Iterative scoping*. The approach to systematic scoping involves structuring the ODM life cycle as a series of incremental scoping steps; each step builds upon and validates the previous step.
- *Stakeholder focus*. The ODM life cycle provides an up-front analysis of the organizational stakeholders and objectives. The stakeholder focus is carried throughout the life cycle with tasks to reconsider the strategic interests of stakeholders at critical points.
- *Exemplar-based modeling*. ODM works from a set of explicit examples, called exemplars, of the domain rather than a single, generalized example or speculation about a "general" solution.
- *Emphasis on descriptive modeling*. ODM places heavy emphasis on studying a set of example systems for the domain in order to derive the shape of the domain space.
- *Explicit modeling of variability.* ODM encourages modelers to maximize variability in the descriptive phase of modeling. This is to generate as much insight as possible about the potential range of variability in the domain.
- *Methods for context recovery*. ODM emphasizes identifying contextual information (e.g., language, values, assumptions, dependencies, history) embedded within an artifact to make them more dependable and predictable. (Note: This activity does not remove dependencies. This occurs during the engineering asset base phase.)
- *Prescriptive asset base engineering*. After descriptive modeling takes place, the prescriptive modeling phase begins. Initially, the range of functionality to be supported by the reusable assets are re-scoped and

commitments are made to a real set of customers. Prescriptive features are mapped onto the structure of the asset base and to sets of specifications for particular assets. Traceability from the features back to exemplar artifacts are maintained to enable the retrieval of additional information (e.g., existing prototypes, history).

Representation Techniques. Although ODM encompasses all of domain engineering, the core method focuses on activities that are unique to domain engineering. Other activities that fall within, but are not specific to domain engineering are supported through "supporting methods." This means that ODM can be integrated with a variety of existing methods (e.g., system and taxonomic modeling techniques) to support unique constraints or preferences of an organization or domain. Examples of supporting methods are the methods associated with the Reuse Library Framework (RLF) [STARS 96c], Canvas [STARS 96a], Domain Architecture-Based Generation for Ada Reuse (DAGAR) [Klinger 96, STARS 96b], and the Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationales (KAPTUR) Tool, which is described as a part of Argument-Based Design Rationale Capture Methods for Requirements Tracing.

Usage Considerations

ODM was developed primarily to support domain engineering projects for domains that are mature, reasonably stable, and economically viable. Although all of the criteria do not need to be met, ODM is most successful when all are present. ODM can be applied in reuse programs that are in their infancy or very mature. ODM does not assume application within an established reuse program, and in fact includes some risk-reduction steps (such as up-front stakeholder analysis) that enable the use of domain analysis as a first step in establishing such a program.

However, it is recommended that the first application of ODM be on a pilot project in a relatively small domain. ODM supports evolution to larger domains or a broader reuse program.

Maturity

ODM has been applied on small-scale and relatively large-scale projects. The following are examples:

- Hewlett-Packard developed a domain engineering workbook by tailoring aspects of an early version of the ODM process model to their organizational objectives. The workbook is being used on numerous internal domain engineering projects within their divisions [Cornwell 96, Simos 96].
- The Air Force CARDS Program applied ODM in several different areas: as a means of structuring a comparative study on architecture representation languages; on the automated message handling system (AMHS) domain analysis effort; and for product-line analysis as part of the Hanscom AFB Domain Scoping effort.

- ODM formed the basis for the domain engineering approach of the Army STARS Demonstration Project. ODM processes were integrated closely with the CFRP [STARS 93] as a higher level planning guide, and with RLF as a domain modeling representation technology [Lettes 96].

Costs and Limitations

Before incorporating ODM into the overall reuse plan, an organization should consider the following:

- The core ODM method does not directly address the ongoing management of domain models and assets, or the use of the assets by application development projects. These activities are part of a larger reuse program described in the CFRP [STARS 93].
- ODM does not encompass the overall reuse program planning including the establishment of producer-consumer relationships between domain engineering projects and other efforts, such as system reengineering projects or planned new projects.
- ODM may not be applicable within organizations that are not prepared to commit to, or at least experiment with, systematic reuse (i.e., reuse of assets that were developed using a software engineering process that is specifically structured for reuse).
- ODM requires that an organization adopt the level of modeling rigor, the modeling styles, or approaches recommended within ODM.
- The use of ODM necessitates a technology infrastructure and level of technical expertise sufficient to support ODM modeling needs [Simos 96].

Complementary Technologies

A complimentary technology is generation techniques.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Organization Domain Modeling
Application category	Domain Engineering (AP.1.2.4)
Quality measures category	Reusability (QM.4.4) Maintainability (QM.3.1) Understandability (QM.3.2)

Computing reviews category	Software Engineering Tools and Techniques (D.2.2)

References and Information Sources

[Cornwell 96]	Cornwell, Patricia Collins. "HP Domain Analysis: Producing Useful Models for Reusable Software." *HP Journal* (August 1996): 46-55.
[Klinger 96]	Klinger, Carol & Solderitsch, James. *DAGAR: A Process for Domain Architecture Definition and Asset Implementation* [online]. Available WWW <URL: http://source.asset.com/stars/darpa/Papers/ArchPapers.html> (1996).
[Lettes 96]	Lettes, Judith A. & Wilson, John. *Army STARS Demonstration Project Experience Report* (STARS-VC-A011/003/02). Manassas, VA: Loral Defense Systems-East, 1996.
[Simos 94]	Simos, M. "Juggling in Free Fall: Uncertainty Management Aspects of Domain Analysis Methods," 512-521. *Fifth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Paris, France, July 4-8, 1994. Berlin, Germany: Springer-Verlag, 1995.
[Simos 96]	Simos, M., et al. *Software Technology for Adaptable Reliable Systems (STARS) Organization Domain Modeling (ODM) Guidebook* Version 2.0 (STARS-VC-A025/001/00). Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996.
[STARS 93]	*Conceptual Framework for Reuse Processes* Volume I, Definition, Version 3.0 (STARS-VC-A018/001/00). Reston, VA: Software Technology for Adaptable Reliable Systems, 1993.
[STARS 96a]	*Canvas Knowledge Acquisition Guide Book* Version 1.0 (STARS-PA29-AC01/001/00) Reston, VA: Software Technology for Adaptable, Reliable Systems, 1996.
[STARS 96b]	*Domain Architecture-Based Generation for Ada Reuse (DAGAR) Guidebook* Version 1.0. Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996.
[STARS 96c]	*Open RLF* (STARS-PA31-AE08/001/00). Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996.

Current Author/Maintainer

Liz Kean, Air Force Rome Laboratory

External Reviewers

Dick Creps, Lockheed Martin, Manassas, VA
Teri Payton, Lockheed Martin, Manassas, VA
Mark Simos, Organon Motives, Inc., Belmont, MA

Modifications

10 Jan 97 (original)

Footnotes

[1] formerly Unisys Defense Systems, Reston, VA

[2] Defense Advanced Research Projects Agency (DARPA) Software Technology for Adaptable, Reliable Systems (STARS)

URL: http://www.sei.cmu.edu/str/descriptions/odm_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

People Capability Maturity Model® (People CMM®)

Software Technology Roadmap

Status

Complete

Purpose and Origin

The People Capability Maturity Model (People CMM) is an organizational change model designed on the premise that improved workforce practices will not survive unless an organization's behavior changes to support them. It was developed to guide systems and software organizations in attracting, motivating, and retaining talented technical staff. The practices in the model help an organization develop the workforce required to execute business strategies, characterize the maturity of workforce practices, set priorities for improving workforce capability, integrate improvements in process and workforce capability, and become an employer of choice.

Employing the process maturity framework of the CMM for Software (SW-CMM), the People CMM describes best practices for managing and developing an organization's entire workforce. It was developed as a companion to the SW-CMM when organizations attempting to conduct improvement efforts discovered that the workforce practices required for enabling software projects were organizational in scope and required specific attention in order to remove barriers to achieving higher levels of SW-CMM maturity.

Although it still supports the SW-CMM, the People CMM has adopted some of the advances made in the CMM Integration℠ (CMMI℠) and has tried to ensure that People CMM improvement programs can integrate with improvement programs guided by the CMMI models. Enhancing the focus on process abilities in workforce competencies at maturity level 3, and quantitative performance management practices at maturity level 4, makes integrating these various models much easier. Also because of its subject matter, the People CMM presents a more detailed framework for the development of workgroups (or teams) which support the use of CMMI models having Integrated Product and Process Development (IPPD) extensions.

People CMM is intended for executives and managers, systems and software professionals, those responsible for improving workforce management practices, human resource professionals, Software Engineering Process Group members who want to accelerate the achievement of higher CMM maturity levels, and those who aspire to be managers of technical professionals.

Technical Detail

The primary objective of the People CMM is to improve the capability of the entire workforce. This can be defined as the level of knowledge, skills, and process abilities available for performing an organization's current and future business activities.

The People CMM consists of five maturity levels. Each maturity level is an evolutionary plateau at which one or more domains of the organization's processes are transformed to achieve a new level of organizational capability. The five levels are as follows:

Maturity Level	Focus	Process areas
5 Optimizing	Continuously improve and align personal, workgroup, and organizational capability	• **Continuous Workforce Innovation** • **Organizational Performance Alignment** • **Continuous Capability Improvement**
4 Predictable	Empower and integrate workforce competencies and manage performance quantitatively.	• **Mentoring** • **Organizational Capability Management** • **Quantitative Performance Management** • **Competency-Based Assets** • **Empowered Workgroups** • **Competency Integration**
3 Defined	Develop workforce competencies and workgroups, and align with business strategy and objectives	• **Participatory Culture** • **Workgroup Development** • **Competency-Based Practices** • **Career Development** • **Competency Development** • **Workforce Planning** • **Competency Analysis**
2 Managed	Managers take responsibility for managing and developing their people.	• **Compensation** • **Training and Development** • **Performance Management** • **Work Environment** • **Communication and Coordination** • **Staffing**

1 Initial	Workforce practices applied inconsistently.	

Process Areas of the People CMM

At Level 1, an organization has no consistent way of performing workforce practices. Most workforce practices are applied without analysis of impact.

At Level 2, organizations establish a foundation on which they deploy common workforce practices across the organization. The goal of Level 2 is to have managers take responsibility for managing and developing their people. For example, the first benefit an organization experiences as it achieves Level 2 is a reduction in voluntary turnover. The turnover costs that are avoided by improved workforce retention more than pay for the improvement costs associated with achieving Level 2.

At Level 3, the organization identifies and develops workforce competencies and aligns workforce and workgroup competencies with business strategies and objectives. For example, the workforce practices that were implemented at Level 2 are now standardized and adapted to encourage and reward growth in the organization's workforce competencies.

At Level 4, the organization empowers and integrates workforce competencies and manages performance quantitatively. For example, the organization is able to predict its capability for performing work because it can quantify the capability of its workforce and of the competency-based processes they use in performing their assignments.

At Level 5, the organization continuously improves and aligns personal, workgroup, and organizational capability. For example, at Maturity Level 5, organizations treat continuous improvement as an orderly business process to be performed in an orderly way on a regular basis.

Usage Considerations

The People CMM was designed initially for knowledge-intense organizations and workforce management processes. However, it can be applied in almost any organizational setting, either as a guide in implementing workforce improvement activities or as a vehicle for assessing workforce practices.

The companion product suite for the People CMM includes:

- A three-day Introduction to the People CMM course
- The People CMM Assessment Method Description
- Two Maturity Questionnaires - one for managers and one for individual contributors

To ensure useful and credible results are obtained from People CMM

assessments, a certification and authorization process has been developed for People CMM Lead Assessors.

Several types of People CMM-based assessments can be performed. Each type of assessment method is most appropriate for distinct situations. Organizations select the type and class of assessment appropriate to their needs.

Assessment Type	People CMM-Based Assessment Method	Joint Assessment	Questionnaire-Based Assessment	Gap Analysis
Assessment Class	Class A	Class A	Class B	Class C
Usage Mode	1. Rigorous and in-depth investigation of workforce practices 2. Basis for improvement activities	1. Rigorous and in-depth investigation of practices, both for workforce practices and the process in the joint domain 2. Basis for improvement activities	1. Initial (first-time) 2. Incremental (partial) 3. Self-assessment	1. Initial (first-time) 2. Self-assessment
Advantages	Thorough coverage; strengths and weak-nesses for each PA investigated; robust-ness of method with consistent, repeatable results; provides objective view	Thorough coverage; strengths and weak-nesses for each PA investigated across multiple domains; robustness of method with consistent, repeatable results; provides objective view	Organization gains insight into own capability; focuses on areas that need most attention; pro-motes awareness and buy-in	Organization gains insight into own capability; provides a starting point to focus on areas that need most attention; promotes buy-in and ownership of results through participation in analysis and planning; typically inexpensive; short duration; rapid feed-back

Disadvantages	Demands significant resources	Demands significant resources	Does not emphasize depth of coverage and rigor and cannot be used for maturity level rating	Risk of participant biases influencing results; not enough depth to ensure completeness; does not emphasize rigor and cannot be used for maturity level rating
Sponsor	Executive management of the organization	Executive management of the organization	Any internal manage	Any internal manager sponsoring an improvement effort
Team Size	4-10 persons + assessment team leader	4-10 persons per domain + assessment team leader(s)	1-6 persons + assessment team leader	3-12 (recommended) + facilitator
Team Qualifications	Experienced	Experienced	Moderately experienced	Limited experience, except for the facilitator
Assessment Team Leader Requirements	Lead assessor	Lead Assessors	Lead assessor	Person trained in People CMM and method

Characteristics of People CMM Assessment Classes

From the perspective of the People CMM, an organization's maturity is derived from the workforce practices it routinely performs, and the extent to which these practices have been institutionalized. A maturity level is, therefore, an evolutionary plateau at which existing processes have been transformed to achieve a level of organizational ability. The transformation and implementation methods may be different at each maturity level, but moving to the next maturity level always requires capabilities established at earlier levels. Consequently, each maturity level establishes a foundation on which higher levels of maturity are built.

Maturity

Since its initial release in 1995, the People CMM has been used throughout the United States, Canada, Europe, Australia, and India to guide and conduct organizational improvement activities. Small and large commercial organizations, as well as government organizations are using People CMM.

Moreover, once executives identify an organization's strategic objectives, the People CMM provides guidance that improves the organization's ability to satisfy those objectives through a competent, capable workforce.

Costs and Limitations

There are several concerns or issues that should be addressed by anyone considering People CMM. When a company reaches Level 3, it has developed workforce competencies and workgroups that are aligned with its business strategies. Without constant updating and renewal, workforce competencies can become obsolete and no longer match business strategies. Therefore, it is imperative to maintain an active program of competency definition and development even if higher maturity levels are not attained.

Another issue is the temptation to skip maturity levels. Although tempting, experience indicates that this practice normally leads to a failed improvement program. In fact, it can actually damage the organization if the workforce builds expectations that are not met because foundational process areas have not been adequately addressed.

Finally, some organizations get "level fever." In these cases, attaining a particular maturity level becomes more important than achieving the business benefits attained through improved practices. Organizations must ensure that the practices implemented in pursuit of higher maturity levels create beneficial change. Otherwise, the organization is just adding a level of bureaucracy that will eventually have to be dismantled.

Complementary Technologies

Complimentary technologies of the People CMM include: CMMI, SW-CMM, Integrated Product Process Development (IPPD), Integrated Project Management (IPM), Integrated Teaming, Organizational Environment for Integration, and Total Quality Management (TQM).

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	People Capability Maturity Model® (P-CMM®)
Application category	Not Applicable
Quality measures category	Organizational Measures (QM.5)

Computing reviews category	Organizational Impacts (K.4.3) Project and People Management (K.6.1) The Computing Profession (K.7)

References and Information Sources

[CMMI 00] CMMI Product Development Team. CMMI[SM] for Systems Engineering/Software Engineering/Integrated Product and Process Development, Version 1.02, Staged Representation (CMU/SEI-2000-TR-030). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/publications/documents/00.reports/00tr030.html

[Curtis 95] Curtis, B.; Hefley, W.E.; & Miller S. People Capability Maturity Model (CMU/SEI-95-MM-002), Pittsburgh, PA: Software Engineering Institute. Carnegie Mellon University, Sept. 1995. http://www.sei.cmu.edu/publications/documents/95.reports/95.mm.002.html

[Curtis 01] Curtis, Bill; Hefley, W.E; & Miller, SA. The People Capability Maturity Model: Guidelines for Improving the Workforce, Reading, MA: Addison-Wesley, 2001.

[Hefley 98] Hefley, W. E. & Curtis, B. People CMM-Based Assessment Method Description (CMM/SEI-98-TR-012 ADA354685). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998. http://www.sei.cmu.edu/publications/documents/98.reports/98tr012/98tr012abstract.html

[Paulk 95] Paulk, M.C.; Weber, C.; Curtis, B.; & Chrissis, M.B. The Capability Maturity Model: Guidelines for Improving the Software Process. Reading, MA: Addison-Wesley, 1995.

Current Author/Maintainer

For more information about the People Capability Maturity Model contact

Sally Miller
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
E-mail: sal@sei.cmu.edu

URL: http://www.sei.cmu.edu/str/descriptions/pcmm_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Personal Software Process for Module-Level Development

Software Technology Roadmap

Status

Complete

Purpose and Origin

The Personal Software Process (PSP)[1] is a set of advanced process and quality techniques to help software engineers improve their performance in their organizations through a step-by-step, disciplined approach to measuring and analyzing their work. Software engineers that use the PSP can substantially improve their ability to estimate and plan their work and significantly improve the quality, i.e., reduce the defects, in the code they develop. PSP is a result of research by Watts Humphrey into applying process principles to the work of individual software engineers and small software teams [Humphrey 95]. The objective was to transfer the quality concepts of the Capability Maturity Model (CMM)[2] for Software [Paulk 95] to the individual and team processes. The PSP provides the foundational skills necessary for individuals to participate on teams using the Team Software Process (TSP)[3].

Technical Detail

The foundations of PSP are the advanced process and quality methods that have been used in manufacturing to improve all forms of production. These concepts include the following:

- definition of the processes
- use of the defined processes
- measurement of the effects of the processes on product
- analysis of the effects on the product
- continuous refinement of the processes based on analysis

Some of the engineering methods used in PSP are data gathering, size and resource estimating, defect management, yield management, and cost of quality and productivity analysis. The basic data gathered in PSP are

- the time the engineer spends in each process phase
- the defects introduced and found in each phase
- the size of the developed product

All PSP process quality measures are derived from this basic set of data. Size and resource estimating is done using a proxy-based estimating method, PROBE [Humphrey 96b], that uses the engineer's personal data and statistical techniques to calculate a new item's most likely size and development time, and the likely range of these estimates. A key PSP tenet is that defect management is an engineer's personal responsibility. By analyzing the data gathered on the defects they injected, engineers refine their personal process to minimize injecting defects, and devise tailored checklists and use personal reviews to remove as many defects as possible. Yield, the principal PSP quality measure, is used to measure the effectiveness of review phases. Yield is defined as the percentage of the defects in the product at the start of the review that were found during the review phase. The basic cost-of-quality measure in PSP is the appraisal-to-failure-ratio (A/FR), which is the ratio of the cost of the review and evaluation phases to the cost of the diagnosing and repair phases. PSP-trained engineers learn to relate productivity and quality, i.e., that defect-free (or nearly so) products require much less time in diagnosing and repair phases, so their projects will likely be more productive.

The PSP (and the courses based on it) concentrates on applying PSP to the design, code, and Unit Test phases, i.e. module-level development phases of software development [Humphrey 95]. As such, an instance of PSP for module-level development is created. In PSP for module-level development, the individual process phases are planning, design, design review, code, code review, compile, test, and postmortem. Size is measured in lines of code and size/resource estimating is done using functions or objects as the proxies for the PROBE method. Engineers build individually tailored checklists for design review and code review based on the history of the defects they inject most frequently. Yield is the percentage of defects found before the first compile, engineers are taught to strive for a 100% yield and can routinely achieve yields in the range of 70%. A/FR is the ratio of the time spent in design review and code review phases to the time spent in compile and test phases, and engineers are encouraged to plan for an A/FR of 2 or higher, which has been shown to correlate well with high yield.

The PSP substantially improves engineering performance on estimating accuracy, defect injection, and defect removal. Class data from 104 engineers that have taken the PSP course shows reductions of 58% in the average number of defects injected (and found in development) per 1,000 lines of code (KLOC), and reductions of 72% in the average number of defects per KLOC found in test. Estimating and planning accuracy also improved with an average 26% reduction in size estimating error and an average 46% reduction in time estimating error. Average productivity of the group went up slightly [Humphrey 96a].

Usage Considerations

The PSP is applicable to new development and enhancement work on whole systems or major subunits. The PSP has been demonstrated and used in organizations at all levels of the CMM. Because PSP emphasizes early defect removal, i.e., spending time in the design through code review phases to prevent and remove as many defects as possible before the first compile, PSP introduction will likely be difficult in organizations that are concerned primarily with schedule and not with product quality.

PSP is an individual development process; however, it can be used by small teams if all members are PSP-trained and team conventions are established for code counting and defect types. PSP does not require sophisticated tools or software development environments; however, simple spreadsheet-based tools can significantly help individual engineers reduce the effort needed to track and analyze their personal data. For teams to apply these principles, the TSP should be applied.

The principles of PSP can be applied to other areas such as developing documentation, handling maintenance, conducting testing, and doing requirements analysis. Humphrey describes how the PSP can be adapted to these and other areas [Humphrey 95].

Maturity

PSP is a relatively new technology. It is already being taught in a number of universities, but industrial introduction has just begun and only limited results are available. Early industrial results are similar to the PSP course results, showing reduced defects and improved quality. Work on industrial transition methods, support tools, and operational practices is ongoing.

Costs and Limitations

PSP training (class room instruction, programming exercises, and personal data analysis reports) requires approximately 10 days of instruction on the part of each engineer. Through the use of 10 programming exercises and 5 reports, engineers are led through a progressive sequence of software processes, taking them from their current process to the full-up PSP for module-level development process. By doing the exercises, engineers learn to use the methods and by analyzing their own data, engineers see how the methods work for them.

Based on demonstrated benefits from course data, it is projected that the costs of training will be recovered by an organization within one year through reduced defects, reduced testing time, improved cycle time, and improved product quality.

Attempts by engineers to learn PSP methods by reading the book and then trying to apply the techniques on real projects have generally not worked. Until they have practiced PSP methods and have become convinced of their effectiveness, engineers are not likely to apply them on the job.

PSP introduction requires strong management commitment because of the significant effort required to learn PSP. It is also recommended that you follow TSP into strategy. Management must provide engineers the time to learn PSP and track their training progress to ensure the training is completed.

Complementary Technologies

The TSP is a follow-up approach to Humphrey's work on PSP. The TSP was designed to provide both a strategy and a set of operational procedures for using

disciplined software process methods at the individual and team levels. The TSP has been used with software-only teams and with mixed teams composed of hardware, software, systems, and test professionals. The TSP can be used on teams that typically range in size from 2 to about 150 individuals. The TSP has been used for both new development and enhancement, and on applications ranging from commercial software to embedded real-time systems. It is also applicable in maintenance and support environments. The TSP is being developed for a wider range of project applications, including large multi-teams, geographically distributed teams, and functional teams [McAndrews 00].

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Personal Software Process for Module-Level Development
Application category	Detailed Design (AP.1.3.5) Code (AP.1.4.2) Unit Testing (AP.1.4.3.4) Component Testing (AP.1.4.3.5) Reapply Software Life Cycle (AP.1.9.3) Reengineering (AP.1.9.5)
Quality measures category	Reliability (QM.2.1.2) Availability (QM.2.1.1) Maintenance Control (QM.5.1.2.3) Productivity (QM.5.2)
Computing reviews category	Management (D.2.9)

References and Information Sources

[Humphrey 95] Humphrey, Watts. *A Discipline for Software Engineering*. Reading, MA: Addison-Wesley Publishing Company, 1995.

[Humphrey 96a] Humphrey, Watts. "Using a Defined and Measured Personal Software Process." *IEEE Software 13*, 3 (May 1996): 77-88.

[Humphrey 96b] Humphrey, Watts. "The PSP and Personal Project Estimating." *American Programmer 9*, 6 (June 1996): 2-15.

[Humphrey 00] Humphrey, Watts. *The Team Software Process℠ (TSP℠)* (CMU/SEI-2000-TR-023). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2000.

[McAndrews 00] McAndrews, Donald R. *The Team Software ProcessSM (TSP℠): An Overview and Preliminary Results of Using Disciplined Practices* (CMU/SEI-2000-TR-015). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2000.

[Paulk 95] Paulk, Mark C. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley Publishing Company, 1995.

Current Author/Maintainer

Dan Burton, SEI

External Reviewers

Archie Andrews, SEI
Watts Humphrey, SEI
Mark Paulk, SEI

Modifications

10 Jan 97 (original)
1 June 01 (updated)

Footnotes

[1] Personal Software Process and PSP are service marks of Carnegie Mellon University.

[2] Capability Maturity Model and CMM are service marks of Carnegie Mellon University.

[3] Team Software Process and TSP are service marks of Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/psp_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Public Key Cryptography

Software Technology Roadmap

Status

Draft

Purpose and Origin

Cryptography is an algorithmic process of converting a plain text (or clear text) message to a cipher text (or cipher) message based on an algorithm that both the sender and receiver know, so that the cipher text message can be returned to its original, plain text form. In its cipher form, a message cannot be read by anyone but the intended receiver. The act of converting a plain text message to its cipher text form is called enciphering. Reversing that act (i.e., cipher text form to plain text message) is deciphering. Enciphering and deciphering are more commonly referred to as *encryption* and *decryption*, respectively.

There are a number of algorithms for performing encryption and decryption, but comparatively few such algorithms have stood the test of time. The most successful algorithms use a *key*. A key is simply a parameter to the algorithm that allows the encryption and decryption process to occur. There are many modern key-based cryptographic techniques [Schneier 96]. These are divided into two classes: symmetric and asymmetric (also called public/private) key cryptography. In *symmetric key cryptography*, the same key is used for both encryption and decryption. In *asymmetric key cryptography*, one key is used for encryption and another, mathematically related key, is used for decryption.

Symmetric Key Cryptography

The most widely used symmetric key cryptographic method is the Data Encryption Standard (DES) [NIST 93]. Although originally published in 1977 by the National Bureau of Standards (reprinted in [Beker+ 82]), DES has not yet been replaced by any other symmetric-key approach. DES uses a fixed length, 56-bit key and an efficient algorithm to quickly encrypt and decrypt messages. DES can be easily implemented in hardware, making the encryption and decryption process even faster. In general, increasing the key size makes the system more secure. A variation of DES, called Triple-DES or DES-EDE (encrypt-decrypt-encrypt), uses three applications of DES and two independent DES keys to produce an effective key length of 168 bits [ANSI 85].

The International Data Encryption Algorithm (IDEA) was invented by James Massey and Xuejia Lai of ETH Zurich, Switzerland in 1991 and is patented and

registered by the Swiss Ascom Tech AG, Solothurn [Lai 92]. IDEA uses a fixed length, 128-bit key (larger than DES but smaller than Triple-DES). It is also faster than Triple-DES. In the early 1990s, Don Rivest of RSA Data Security, Inc., invented the algorithms RC2 and RC4. These use variable length keys and are claimed to be even faster than IDEA. However, implementations may be exported from the U.S. only if they use key lengths of 40 bits or fewer.

Although symmetric key cryptography works, it has a fundamental weak spot-key management. Since the same key is used for encryption and decryption, it must be kept secure. If an adversary knows the key, then the message can be decrypted. At the same time, the key must be available to the sender and the receiver and these two parties may be physically separated. Symmetric key cryptography transforms the problem of transmitting messages securely into that of transmitting keys securely. This is a step forward, because keys are much smaller than messages, and the keys can be generated beforehand. Nevertheless, ensuring that the sender and receiver are using the same key and that potential adversaries do not know this key remains a major stumbling block. This is referred to as the key management problem.

Public/Private Key Cryptography

Asymmetric key cryptography overcomes the key management problem by using different encryption and decryption key pairs. Having knowledge of one key, say the encryption key, is not sufficient enough to determine the other key - the decryption key. Therefore, the encryption key can be made public, provided the decryption key is held only by the party wishing to receive encrypted messages (hence the name public/private key cryptography). Anyone can use the public key to encrypt a message, but only the recipient can decrypt it.

James Ellis, Malcolm Williamson, and Clifford Cocks first investigated public/private key cryptography at the British Government Communications Headquarters (GCHQ) in the early 1970s [Ellis 87]. The first public discussion of public/private key cryptography was by Whitfield Diffie and Martin Hellman in 1976 [Diffie+ 76].

A widely used public/private key algorithm is RSA, named after the initials of its inventors, Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman [RSA 91]. RSA depends on the difficulty of factoring the product of two very large prime numbers. Although used for encrypting whole messages, RSA is much less efficient than symmetric key algorithms such as DES. ElGamal is another public/private key algorithm [El Gamal 85]. It uses a different arithmetic algorithm than RSA, called the discrete logarithm problem. An extensive discussion of public/private key cryptography, including much of the mathematical detail, can be found in the book, *Public Key Cryptography* [Salomaa 96].

Technical Detail

The mathematical relationship between the public/private key pair permits a general rule: any message encrypted with one key of the pair can be successfully decrypted only with that key's counterpart. To encrypt with the public key means you can decrypt only with the private key. The converse is also

true - to encrypt with the private key means you can decrypt only with the public key.

The decision as to which key is kept private and which is made public is not arbitrary. In the case of RSA, the public key uses exponents that are relatively small (in comparison to the private key) making the process of encryption and digital signature verification (discussed later) faster.

Figure 1 illustrates the proper and intended used of public/private key cryptography for sending confidential messages. In the illustration, a user, Bob, has a public/private key pair. The public portion of that key pair is placed in the public domain (for example in a Web server). The private portion is guarded in a private domain, for example, on a digital key card or in a password-protected file.

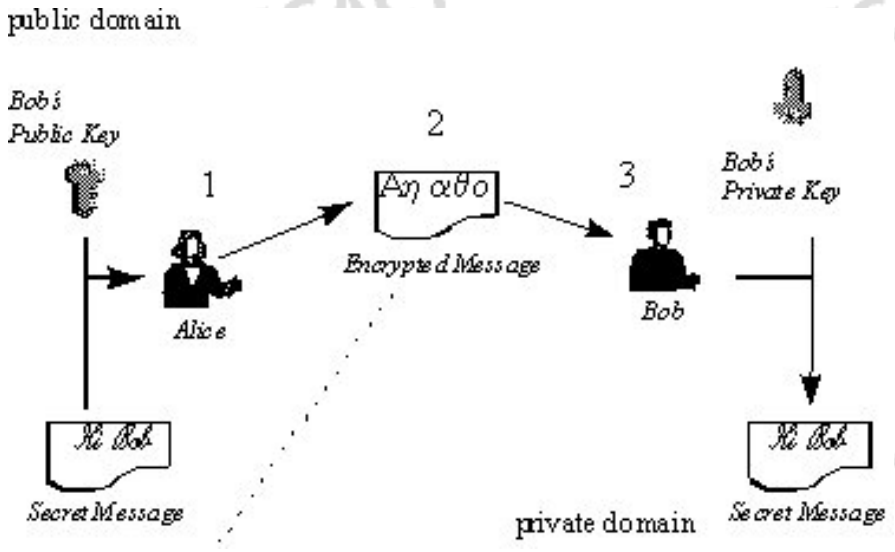


Figure 1: Proper Use of Public Key Cryptography

For Alice to send a secret message to Bob, the following process needs to be followed:

1. Alice passes the secret message and Bob's public key to the appropriate encryption algorithm to construct the encrypted message.
2. Alice transmits the encrypted message (perhaps via e-mail) to Bob.
3. Bob decrypts the transmitted, encrypted message with his private key and the appropriate decryption algorithm.

Bob can be assured that Alice's encrypted secret message was not seen by anyone else since only his private key is capable of decrypting the message.

Since we know that a private key can also be used to encrypt messages, Bob could technically respond in secret to Alice's original message by using the same public/private key pair as illustrated in Figure 2.

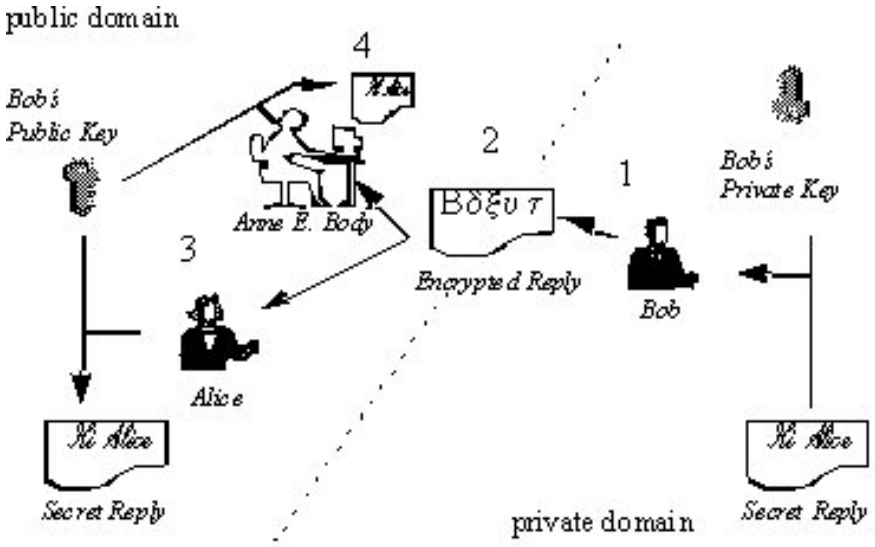


Figure 2: Improper Use of Public Key Cryptography

In this scenario:

1. Bob passes the secret reply and his private key to the encryption algorithm to construct the encrypted reply.
2. Bob transmits the encrypted reply to Alice.
3. Alice decrypts the transmitted, encrypted reply with Bob's public key and the decryption algorithm to read this reply.

Unfortunately, Bob's message will not be confidential because anyone with access to the encrypted reply and Bob's public key (which is in the public domain) can decrypt the reply and see the text of the message. However, if Alice had her own public/private key pair, then Bob and Alice could communicate confidentially. In this case, Bob would send messages encrypted with Alice's public key (which only Alice could decrypt by using her private key), and Alice would send messages to Bob encrypted with Bob's public key (which only he could decrypt using his private key).

Usage Considerations

Public key cryptography is especially useful in situations where there is a need for confidentiality, integrity, and non-repudiation. That is, in situations where the messages being passed are intended to only be shared by the sending and receiving parties. Further, public key cryptography is used in situations where the recipient of a message must have confidence that the message received was received as intended by the sender and has not been altered or forged in any manner.

Confidentiality assures that unintended third parties can not view information sent between two communicating parties. Encryption is the most widely used mechanism for providing confidentiality over an insecure medium.

Integrity is knowing that the message you receive was exactly what was sent and it was unaltered or damaged during transmission. Digital signatures are

used to seal a message as a means to warn if the integrity of a message has been compromised. Today, Web content that executes on local workstations is commonly downloaded. Knowing that the content has not been surreptitiously modified is critical if you are to trust the content. If the content is from a trusted source and it is unmodified, your confidence in that content is higher - because the content has integrity. If the content is from an unknown source or you cannot tell if it has been modified, the content cannot be trusted. Mechanisms such as digital signatures and certificates help maintain the integrity of exchanged products and services.

Non-repudiation is the inability to disavow an act. In other words, evidence exists that prevents a person from denying an act. For example, you log in to a computer system by presenting a user name and password. Most software applications consider this sufficient evidence to permit access, but could it be proved that it was really you that was logged in? You could argue that someone else obtained your password, possibly using snooping techniques. Now, suppose that a computer system requires a fingerprint or retinal image to gain access. Contesting the fact now becomes more difficult.

Finally, as opposed to symmetric key cryptography, public key cryptography is a useful means of getting around issues dealing with key distribution and management.

Maturity

Public key cryptography has been in use for more than 30 years. Secure Sockets Layer (SSL) defined by Netscape is a popular application of public key cryptography found in Web-enabled applications requiring secure communications and authentication. Pretty Good Privacy (or PGP) is another popular application of public key cryptography used to send confidential electronic mail and digitally signing electronic documents.

Further, a number of commercial companies have become third party providers of public key cryptography software including, but not limited to, RSA Security, Inc, Sun Microsystems, Microsoft, Entrust, Inc., and VeriSign, Inc.

Costs and Limitations

Cost to implement public key cryptography in a system vary according to size and scope. Characteristics that can determine costs include the number of pair-wise keys that need to be created for the purposes of confidentiality and integrity. For example, securing all corporate email will require that employers to issue public keys to all of its employees and enforce the use of those key when communicating corporate ideas and correspondence. Systems are available to support such wide use but come at a cost. A counter-example of this would be a corporate "portal" or web site available to the public from which the public may be asked to place orders. In such a case, the corporation may only be required to acquire public key cryptography for the one or more server(s) that will be used to interact with the public, this is typically a annual cost from security providers such as VeriSign, Inc.

Using this technology may require network management personnel with knowledge of public key cryptography and the use of software that implements public key cryptography and digital signature algorithms especially if an outside provider for public key infrastructures is NOT used. It also requires security personnel and software that can generate, distribute, and control encryption/ decryption keys and respond to the loss or compromise of keys.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Public Key Cryptography
Application category	Information Security (AP.2.4)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating Systems Security & Protection (D.4.6), Security & Protection (K.6.5), Computer-Communications Networks Security and Protection (C.2.0)

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Schneier 96] Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd editon by , John Wiley & Sons, ISBN 0471128457, 1996.

[NIST 93] Data Encryption Standard (DES) (FIPS PUB 46-2). Gaithersburg, Md.: National Institute of Standards and Technology, January, 1993. Available WWW: <URL: http://www.nist.gov/itl/div897/pubs/fip46-2.htm>.

[Beker+ 82] Beker, H. & Piper, F. Cipher Systems. London: Northwood Books, 1982.

[ANSI 85] ANSI X9.17-1985, American National Standard, Financial Institution Key Management (Wholesale), American Bankers Association, Section 7.2. New York: American National Standards Institute, 1985.

[Lai 92] Lai, X. ETH Series on Information Processing (J.L. Massey, ed.). Vol. 1, On the Design and Security of Block Ciphers. Konstanz, Switzerland: Hartung-Gorre Verlag, 1992.

[Ellis 87] Ellis, J.H. ìThe Story of Non-Secret Encryption.î Cheltenham, UK: Communications Electronics Security Group, 1987. Available WWW: <URL: http://www.cesg.gov.uk/about/nsecret/ellis.htm>

[Diffie+] Diffie, W. & Hellman, M.E. ìNew Directions in Cryptography.î IEEE Transactions on Information Theory, IT-22, Vol. 6, pp. 644-654, 1976.

[RSA 91] PKCS #1: RSA Encryption Standard, Version 1.4. San Mateo, Ca.: RSA Data Security, Inc., 1991.

[El Gamal 85] El Gamal, T. ìA Public Key Cryptosystem and Signature Scheme Based on Discrete Logarithms.î IEEE Transactions on Information Theory, IT-31, pp. 469-473, 1985.

[Salomaa 96] Salomaa, A. Public-Key Cryptography, 2nd edition. Berlin: Springer-Verlag, 1996.

Current Author/Maintainer

Scott A. Hissam, SEI

External Reviewers

Modifications

9 Dec 01 (original)

URL: http://www.sei.cmu.edu/str/descriptions/publickey_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Public Key Digital Signatures

Software Technology Roadmap

Status

Advanced

Note

We recommend Computer System Security- an Overview as prerequisite reading for this technology description.

Purpose and Origin

Public key digital signature techniques provide data integrity and source authentication capabilities to enhance data trustworthiness in computer networks. This technology uses a combination of a message authentication code (MAC) to guarantee the integrity of data and unique features of paired public and private keys associated with public key cryptography to uniquely authenticate the sender [Schneier 96, Abrams 95]. This technology was first defined in the early 1980s with the development of public key cryptography but has received renewed interest as an authentication mechanism on the Internet.

Technical Detail

Trustworthiness of data received by a computer from another computer is a function of the security capabilities of both computers and the communications between them. One of the fundamental objectives of computer security is data integrity [White 96]. Two aspects of data integrity are improved by public key digital signature techniques. These are sender authentication and data integrity verification. Positive authentication of the message source is provided by the unique relationship of the two encryption keys used in public key cryptography. Positive verification of message integrity is provided by the use of a message authentication code (sometimes called a manipulation detection code or a cryptographic checksum) that is produced by a message digest (sometimes called a data hashing) function. The use of a message authentication code and public key cryptography are combined in the public key digital signature techniques technology.

Sender authentication. Public key cryptography uses two paired keys. These are the public key and the private key (sometimes called the secret key), which are related to each other mathematically. The public key is distributed to anyone that needs to encrypt a message destined for the holder of the private key. The private key is not known to anyone but the holder of the private key. Because of the mathematical relationship of the keys, data encrypted with the public key can only be decrypted with the private key. Another feature of the paired key relationship is that if a message can be successfully decrypted with the public key then it must have been encrypted with the private key. Therefore, any message decrypted by a holder of the public key must have been sent by the holder of the private key. This is used to authenticate the source of a message. Public

key cryptography can use one of several algorithms but the most common one is the Revest, Shamir, and Adleman (RSA) algorithm. It is used to produce the paired keys and to encrypt or decrypt data using the appropriate key.

Data integrity verification. Message digest functions produce a single large number called the message authentication code (MAC) that is *unique*[1] to the total combination and position of characters in the message being digested. The message digest function distributed with RSA is called the MD5 message digest function. It produces a *unique* 128 bit number for each different message digested. If even one character is changed in the message, a dramatically-different 128 bit number is generated.

The overall process for using Public Key Digital Signatures to verify data integrity is shown in Figure 22.

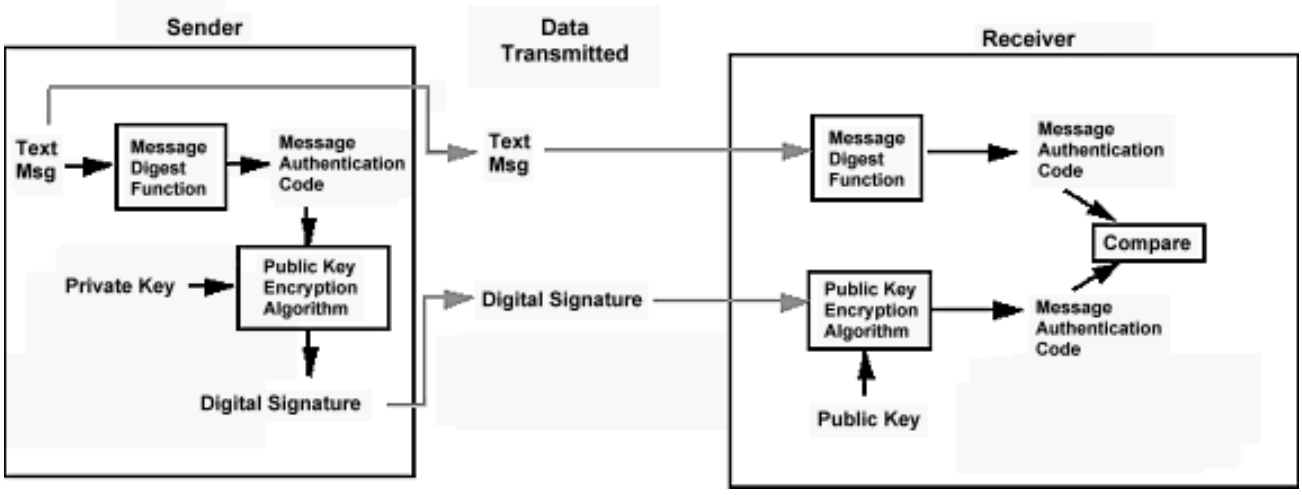


Figure 22: Public Key Digital Signatures

The Digital Signature of a message is produced in two steps:

1. The sender of the message uses the message digest function to produce a message authentication code (MAC).
2. This MAC is then encrypted using the private key and the public key encryption algorithm. This encrypted MAC is attached to the message as the digital signature.

The receiver of the message uses the public key to decrypt the digital signature. If it is decrypted successfully, the receiver of the message knows it came from the holder of the private key. The receiver then uses the message digest function to calculate the MAC associated with the received message contents. If this number compares to the one decrypted from the Digital Signature, the message was received unaltered and data integrity is assured. Together, this technique provides data source authentication and verification of message content integrity.

There are many message digest functions and public key encryption algorithms that may be used in developing the public key digital signature technique. A discussion of these alternative algorithms and their merits is in Schneier [Schneier 96].

Usage Considerations

This technology is most likely to be used in networks of computers where all the communication paths can not be physically protected and where the integrity of data and sender authenticity

aspects of trustability are essential. Military C4I networks and banking networks that are on a widespread local area network or a wide area network are prime examples of this use.

Implementation of the public key digital signature techniques establishes additional requirements on a network. The same message digest functions and public key cryptography algorithm used to process the digital signature must be used by both the sender and receiver. Public/private key pairs must be generated and maintained. Public keys must be distributed (or accessible in a public forum) and private keys protected.

Maturity

The components of this technology, public key encryption and message digest functions, have been in use since the early 1980s. The combined technology is mature and is available in implementations that range from small networks of PCs to protection of data being transferred over the Internet.

The algorithms supporting public key digital signatures have historically consumed large amounts of processing power. However, given recent advances in processors used in PCs and workstations; this is no longer a concern in most circumstances of use.

Costs and Limitations

Using this technology requires network management personnel with knowledge of public key cryptography and the use of software that implements public key cryptography and digital signature algorithms. It also requires security personnel and software that can generate, distribute, and control encryption/decryption keys and respond to the loss or compromise of keys.

Dependencies

Public key cryptography and message digest functions.

Alternatives

Data integrity and authentication can be provided by a combination of dedicated circuits, integrity protocols, and procedural control of sources and destinations. These approaches are not foolproof and can be expensive. Data integrity and authentication can also be provided using private key encryption and a third party arbitrator. This approach has the disadvantage that a third party must be trusted and the data must be encrypted and decrypted twice with two separate private keys.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Public Key Digital Signatures
Application category	System Security (AP.2.4.3)

Quality measures category	Trustworthiness (QM.2.1.4)
Computing reviews category	Computer-Communication Networks Security and Protection (C.2.0) Security and Protection (K.6.5)

References and Information Sources

[Abrams 95]	Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.
[Garfinkel 95]	Garfinkel, Simpson. *PGP: Pretty Good Privacy*. Sebastopol, CA: O'Reilly & Associates, 1995.
[Russel 91]	Russel, Deborah & Gangemi, G.T. Sr. *Computer Security Basics.* Sebastopol, CA: O'Reilly & Associates, Inc., 1991.
[Schneier 96]	Schneier, Bruce. *Applied Cryptography*. New York, NY: John Wiley & Sons, 1996.
[White 96]	White, Gregory B.; Fisch, Eric A.; & Pooch, Udo W. *Computer System and Network Security.* Boca Raton, FL: CRC Press, 1996.

Current Author/Maintainer

Tom Mills, Lockheed Martin

External Reviewers

Jim Ellis, SEI
Scott A. Hissam, SEI

Modifications

4 Nov 03 (typo correction) 26 Jun 00 (references to "secret key" changed to "private key")

10 Jan 97 (original)

Footnotes

[1] Of course they are not absolutely unique. We say unique here because it is extremely unlikely statistically for two files to have the same MAC and, more importantly, it is extremely difficult for an attacker/malicious user to create/craft two files having the same MAC.

URL: http://www.sei.cmu.edu/str/descriptions/pkds_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Rate Monotonic Analysis

Software Technology Roadmap

Status

Complete

Purpose and Origin

Rate Monotonic Analysis (RMA) is a collection of quantitative methods and algorithms that allows engineers to specify, understand, analyze, and predict the timing behavior of real-time software systems, thus improving their *dependability* and *evolvability*.

RMA grew out of the theory of fixed priority scheduling. A theoretical treatment of the problem of scheduling periodic tasks was first discussed by Serlin in 1972 [Serlin 72] and then more comprehensively treated by Liu and Layland in 1973 [Liu 73]. They studied an idealized situation in which all tasks are periodic, do not synchronize with one another, do not suspend themselves during execution, can be instantly preempted by higher priority tasks, and have deadlines at the end of their periods. The term "rate monotonic" originated as a name for the optimal task priority assignment in which higher priorities are accorded to tasks that execute at higher rates (that is, as a monotonic function of rate). Rate monotonic scheduling is a term used in reference to fixed priority task scheduling that uses a rate monotonic prioritization.

During the 1980s the limitations of the original theory were overcome and the theory was generalized to the point of being practicable for a large range of realistic situations encountered in the design and analysis of real-time systems [Sha 91a]. RMA can be used by real-time system designers, testers, maintainers, and troubleshooters, as it provides

- mechanisms for predicting real-time performance
- structuring guidelines to help ensure performance predictability
- insight for uncovering subtle performance problems in real-time systems

This body of theory and methods is also referred to as generalized rate monotonic scheduling (GRMS), a codification of which can be found in Klein [Klein 93].

Technical Detail

RMA provides the analytical foundation for understanding the timing behavior of real-time systems that must manage many concurrent threads of control. Real-time systems often have stringent latency requirements associated with each thread that are derived from the environmental processes with which the system is interacting. RMA provides the basis for predicting whether such latency requirements can be satisfied. Some of the important factors that are used in RMA calculations include:

- the worst-case execution time of each thread of control
- the minimum amount of time between successive invocations of each thread
- the priority levels associated with the execution of each thread
- sources of overhead such as those due to an operating system
- delays due to interprocess communication and synchronization
- allocation of threads of control to physical resources such as CPUs, buses, and networks

These factors and other aspects of the system design are used to calculate worst-case latencies for each thread of control. These worst-case latencies are then compared to each thread's timing requirements to determine if the requirement can be satisfied.

A problem commonly revealed as a result of rate monotonic analysis is priority inversion. Priority inversion is a state in which the execution of a higher priority thread is forced to wait for a resource while a lower priority thread is using the resource. Not all priority inversion can be avoided but proper priority management can reduce priority inversion. For example, priority inheritance is a useful technique for reducing priority inversion in cases where threads must synchronize [Rajkumar 91].

Since RMA is an analytic approach that can be used before system integration to determine if latency requirements will be met, it can result in significant savings in both system resources and development time.

Usage Considerations

RMA is most suitable for systems dominated by a collection of periodic or sporadic processes (i.e., processes with minimum inter-arrival intervals), for which the processing times can be bounded and are without excessive variability. RMA is also primarily focused on hard deadlines rather than soft. However, soft deadlines can be handled through the use of server mechanisms that allocate time to tasks with soft deadlines in a manner that ensures that hard deadlines are still met. Still, with soft deadline tasks, the aperiodic server predictions work best when the workload is primarily periodic.

Systems in which worst-case executions are realized very infrequently or in which there is no minimum inter-arrival interval between thread invocations might *not* be suitable for RMA analysis. For example, consider multimedia applications where voice and data transmissions involve a great deal of variability. Principles of RMA, such as priority representation, priority arbitration, and priority inheritance, can be used in multimedia systems to reduce response

times and meet deadlines at relatively high levels of use. However, deadlines in such environments may not be hard, and execution times can be stochastic, two requirements that are *not* currently handled well in the RMA framework. When most of the workload is aperiodic, one needs to move to queueing theory.

Maturity

Indicators of RMA maturity include the following:

- In 1989 IBM applied RMA to a sonar training system, allowing them to discover and correct performance problems [Lucas 92].
- Since 1990, RMA was recommended by IBM Federal Sector Division (now Lockheed Martin) for its real-time projects.
- RMA was successfully applied to active and passive sonar of a major submarine system of US Navy.
- RMA was selected by the European Space Agency as the baseline theory for its Hard Real-Time Operating System Project.
- The applicability of RMA to a typical avionics application was demonstrated [Locke 91].
- RMA was adopted in 1990 by NASA for development of real-time software for the space station data management subsystem. In 1992 Acting Deputy Administrator of NASA, Aaron Cohen stated, "Through the development of rate monotonic scheduling, we now have a system that will allow (Space Station) Freedom's computers to budget their time to choose [among] a variety of tasks, and decide not only which one to do first but how much time to spend in the process."
- Magnavox Electronics Systems Company incorporated RMA into real-time software development [Ignace 94].
- RMA principles have influenced the design and development of the following standards:
  - IEEE Futurebus+ [Sha 91b]
  - POSIX
  - Ada 95
- Tool vendors provide the capability to analyze real-time designs using RMA. RMA algorithms, such as priority inheritance, have been used by operating system and Ada compiler vendors.

Costs and Limitations

Case studies of RMA adoption show that "While RMA does require engineers to re-frame their understanding of scheduling issues to a more abstract level, only moderate training is required for people to be effective in using the technology" [Fowler 93]. A short (1-2 day) tutorial is usually sufficient to gain a working knowledge of RMA.

Additionally, the studies found "RMA can be incorporated into software engineering processes with relative ease over a period of several months.... RMA can be adopted incrementally; its adoption can range from application to an existing system by one engineer to application across an entire division as standard practice in designing new systems" [Fowler 93].

RMA can be applied with varying degrees of detail. Qualitative analysis through the application of design and trouble shooting heuristics can be very effective. Simple quantitative analysis using back-of-the-envelope calculations quickly yields insight into system timing behavior. More precise quantitative analysis can be performed as more precise system measurements become available during the development activity.

Dependencies

Application performance is influenced by system components such as operating systems networks and communication protocols. Therefore, it is important for such system components to be designed with RMA in mind.

Complementary Technologies

Simulation is often used to gain insight into a system's performance. Simulation can be used to corroborate RMA's performance predictions. Queueing theory is complementary to RMA. Whereas RMA is used to predict worst-case latencies when bounds can be placed on arrival dates and execution times, queueing theory can be used to predict average-case behavior when arrival rates and execution times are described stochastically. Together RMA and queuing theory solve a wide set of performance problems.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Rate Monotonic Analysis
Application category	Detailed Design (AP.1.3.5) System Analysis and Optimization (AP.1.3.6) Code (AP.1.4.2) Performance Testing (AP.1.5.3.5) Reapply Software Life Cycle (AP.1.9.3) Reengineering (AP.1.9.5)
Quality measures category	Real-Time Responsiveness/Latency (QM.2.2.2) Maintainability (QM.3.1) Reliability (QM.2.1.2)
Computing reviews category	Real-Time Systems (C.3)

References and Information Sources

[Audsley 95] Audsley, N.C., et al. "Fixed Priority Pre-Emptive Scheduling: An Historical Perspective." *Real Time Systems 8*, 2-3. (March-May 1995): 173-98.

[Fowler 93] Fowler, P. & Levine, L. *Technology Transition Push: A Case Study of Rate Monotonic Analysis* Part 1 (CMU/SEI-93-TR-29). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

[Ignace 94] Ignace, S. J.; Sedlmeyer, R. L.; & Thuente, D. J. "Integrating Rate Monotonic Analysis into Real-Time Software Development," 257-274. *IFIP Transactions, Diffusion, Transfer and Implementation of Information Technology* (A-45). Pittsburgh, PA, October 11-13, 1993. The Netherlands: International Federation of Information Processing, 1994.

[Klein 93] Klein, M.H., et al. *A Practitioners' Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Boston, MA: Kluwer Academic Publishers, 1993.

[Lehoczky 94] Lehoczky, J.P. "Real-Time Resource Management Techniques," 1011-1020. *Encyclopedia of Software Engineering*. New York, NY: J. Wiley & Sons, 1994.

[Liu 73] Liu, C. L. & Layland, J. W. "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment." *Journal of the Association for Computing Machinery 20*, 1 (January 1973): 40-61.

[Locke 91] Locke, C.D.; Vogel, D.R.; & Mesler, T.J. "Building a Predictable Avionics Platform in Ada: a Case Study," 181-189. *Proceedings of the Twelfth Real-Time Systems Symposium*. San Antonio, TX, December 4-6, 1991. Los Alamitos, CA: IEEE Computer Society Press, 1991.

[Lucas 92] Lucas, L. & Page, B. "Tutorial on Rate Monotonic Analysis." *Ninth Annual Washington Ada Symposium*. McLean, VA, July 13-16, 1992. New York, NY: Association for Computing Machinery, 1992.

[Rajkumar 91] Rajkumar, Ragunathan. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Boston, MA: Kluwer Academic Publishers, 1991.

[Serlin 72] Serlin, O. "Scheduling of Time Critical Processes," 925-932. *Proceedings of the Spring Joint Computer Conference*. Atlantic City, NJ, May 16-18, 1972. Montvale, NJ: American Federation of Information Processing Societies, 1972.

[Sha 91a] Sha, Klein & Goodenough, J. "Rate Monotonic Analysis for Real-Time Systems," 129-155. *Foundations of Real-Time Computing: Scheduling and Resource Management*. Boston, MA: Kluwer Academic Publishers, 1991.

[Sha 91b] Sha, L.; Rajkumar, R.; & Lehoczky, J. P. "Real-Time Computing with IEEE Futurebus+." *IEEE Micro 11*, 3 (June 1991): 30-38.

Current Author/Maintainer

Mark Klein, SEI

External Reviewers

Mike Gagliardi, SEI
John Goodenough, SEI
John Lehoczky, Professor, Statistics Department, Carnegie Mellon University
Ray Obenza, SEI
Raj Rajkumar, Carnegie Mellon University
Lui Sha, SEI

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/rma_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Reference Models, Architectures, Implementations--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

Much confusion exists regarding the definition, applicability, and scope of the terms *reference model*, *architecture*, and *implementation*. Understanding these terms facilitates understanding legacy system designs and how to migrate them to more open systems. The purpose of this technology description is to provide definitions, and more importantly, to describe how the terms are related.

Technical Detail

Reference model. A reference model is a description of all of the possible software components, component services (functions), and the relationships between them (how these components are put together and how they will interact). Examples of commonly-known reference models include the following:

- the Technical Architecture for Information Management (TAFIM) reference model (see TAFIM Reference Model)
- the Reference Model for Frameworks of Software Engineering Environments [ECMA 93]
- Project Support Environment Reference Model (PSERM)
- the Tri-Service Working Group Open Systems Reference Model

Architecture. An architecture is a description of a subset of the reference model's component services that have been selected to meet a specific system's requirements. In other words, not all of the reference model's component services need to be included in a specific architecture. There can be many architectures derived from the same reference model. The associated standards and guidelines for each service included in the architecture form the open systems architecture and become the criteria for implementing the system.

Implementation. The implementation is a product that results from selecting (e.g., commercial-off-the-shelf), reusing, building and integrating software components and component services according to the specified architecture. The selected, reused, and/or built components and component services must comply 100% with the associated standards and guidelines for the implementation to be considered compliant.

Usage Considerations

Figure 23 attempts to show the interrelationships of these concepts using the TAFIM as an example. TAFIM provides the reference model and a number of specific architectures can be derived from the TAFIM reference model based on specific program requirements. From there a number of implementations may be developed based on the products selected to meet the architecture's services,

so long as these products meet the required standards and guidelines. For instance, in one implementation, the product ORACLE might be selected and used to meet some of the data management services. In another implementation, the product Sybase might be selected and used.

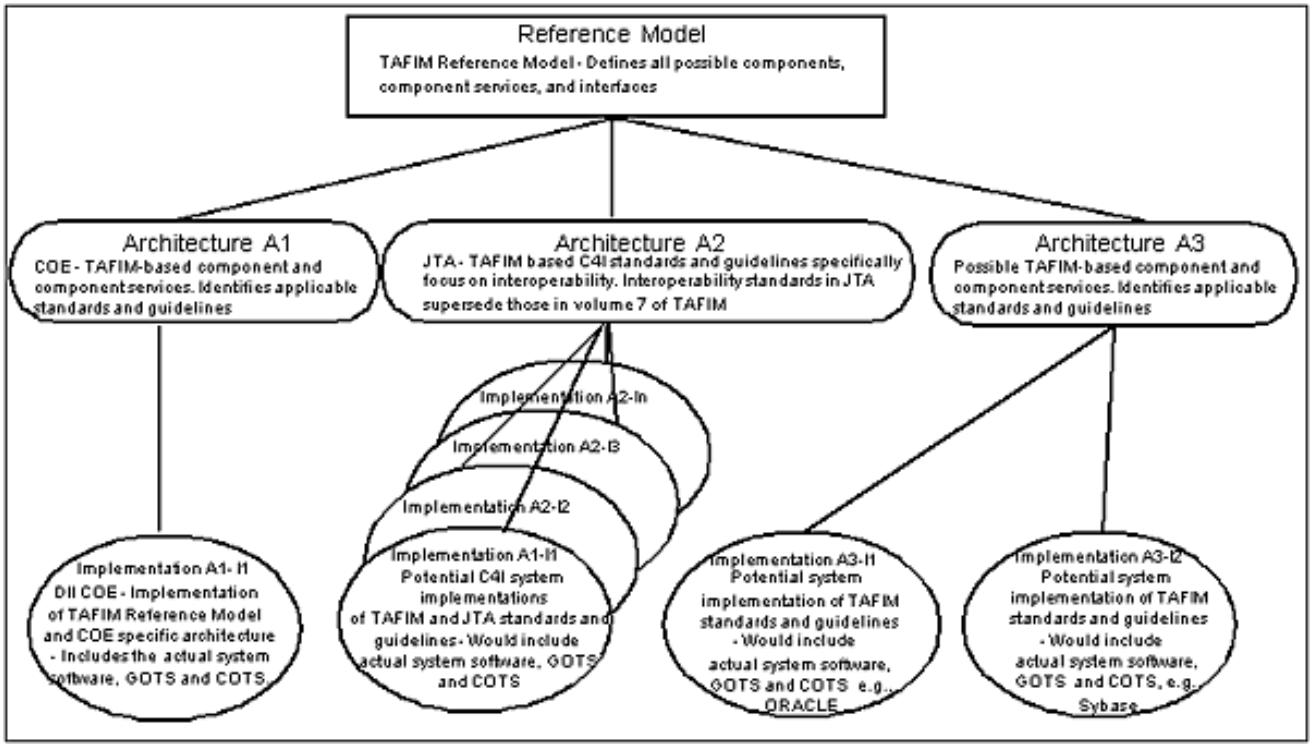


Figure 23: Reference Model, Architecture, and Implementation

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Reference Models, Architectures, Implementations - An Overview
Application category	Software Architecture Models (AP.2.1.1) Software Architecture (AP.2.1)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1) Portability (QM.4.2)
Computing reviews category	Distributed Systems (C.2.4) Software Engineering Design (D.2.10)

References and Information Sources

[ECMA 93] *Reference Model for Frameworks of Software Engineering Environments*, 3rd Edition (NIST Special Publication 500-211/Technical Report ECMA TR/55). Prepared jointly by NIST and the European Computer Manufacturers Association (ECMA). Washington, DC: U.S. Government Printing Office, 1993.

[Meyers 96] Meyers, Craig & Oberndorf, Tricia. *Open Systems: The Promises and the Pitfalls*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Tricia Oberndorf, SEI

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/refmodels_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Remote Procedure Call

Software Technology Roadmap

Status

Advanced

Note

We recommend Middleware *as prerequisite reading for this technology description.*

Purpose and Origin

Remote Procedure Call (RPC) is a client/server infrastructure that increases the *interoperability*, *portability,* and *flexibility* of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the *complexity* of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces--function calls are the programmer's interface when using RPC [Rao 1995].

The concept of RPC has been discussed in literature as far back as 1976, with full-scale implementations appearing in the late 1970s and early 1980s [Birrell 84].

Technical Detail

In order to access the remote server portion of an application, special function calls, RPCs, are embedded within the client portion of the client/server application program. Because they are embedded, RPCs do not stand alone as a discreet middleware layer. When the client program is compiled, the compiler creates a local stub for the client portion and another stub for the server portion of the application. These stubs are invoked when the application requires a remote function and typically support synchronous calls between clients and servers. These relationships are shown in Figure 32 [Steinke 95].

By using RPC, the complexity involved in the development of distributed processing is reduced by keeping the semantics of a remote call the same whether or not the client and server are collocated on the same system. However, RPC increases the involvement of an application developer with the

complexity of the master-slave nature of the client/server mechanism.

RPC increases the flexibility of an architecture by allowing a client component of an application to employ a function call to access a server on a remote system. RPC allows the remote component to be accessed without knowledge of the network address or any other lower-level information. Most RPCs use a synchronous, request-reply (sometimes referred to as "call/wait") protocol which involves blocking of the client until the server fulfills its request. Asynchronous ("call/nowait") implementations are available but are currently the exception.

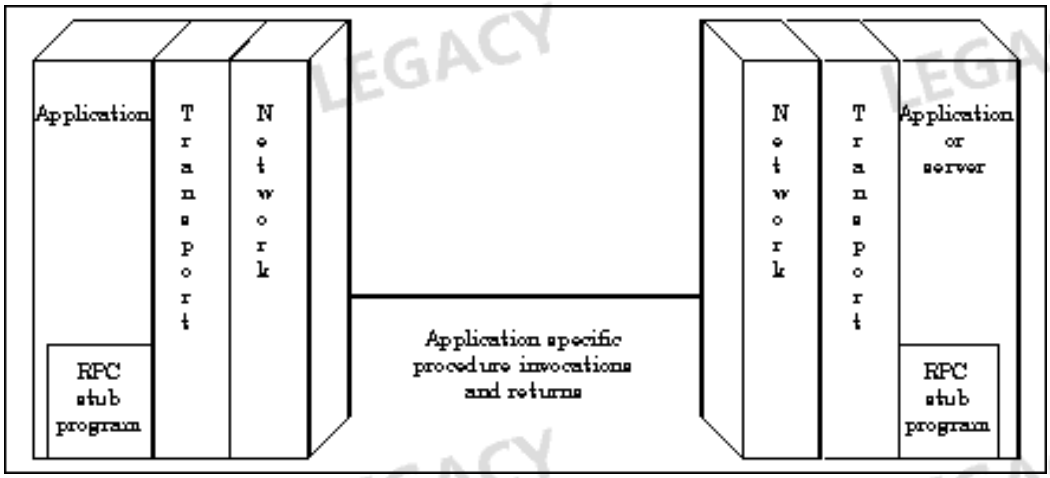


Figure 32: Remote Procedure Calls

RPC is typically implemented in one of two ways:

1. within a broader, more encompassing propriety product
2. by a programmer using a proprietary tool to create client/server RPC stubs

Usage Considerations

RPC is appropriate for client/server applications in which the client can issue a request and wait for the server's response before continuing its own processing. Because most RPC implementations do not support peer-to-peer, or asynchronous, client/server interaction, RPC is not well-suited for applications involving distributed objects or object-oriented programming (see Object-Oriented Programming Languages).

Asynchronous and synchronous mechanisms each have strengths and weaknesses that should be considered when designing any specific application. In contrast to asynchronous mechanisms employed by Message-Oriented Middleware, the use of a synchronous request-reply mechanism in RPC requires that the client and server are always available and functioning (i.e., the client or server is not blocked). In order to allow a client/server application to recover from a blocked condition, an implementation of a RPC is required to provide mechanisms such as error messages, request timers, retransmissions, or redirection to an alternate server. The complexity of the application using a RPC is dependent on the sophistication of the specific RPC implementation (i.e., the

more sophisticated the recovery mechanisms supported by RPC, the less complex the application utilizing the RPC is required to be). RPCs that implement asynchronous mechanisms are very few and are difficult (complex) to implement [Rao 1995].

When utilizing RPC over a distributed network, the performance (or load) of the network should be considered. One of the strengths of RPC is that the synchronous, blocking mechanism of RPC guards against overloading a network, unlike the asynchronous mechanism of Message-Oriented Middleware (MOM). However, when recovery mechanisms, such as retransmissions, are employed by an RPC application, the resulting load on a network may increase, making the application inappropriate for a congested network. Also, because RPC uses static routing tables established at compile-time, the ability to perform load balancing across a network is difficult and should be considered when designing an RPC-based application.

Maturity

Tools are available for a programmer to use in developing RPC applications over a wide variety of platforms, including Windows (3.1, NT, 95), Macintosh, 26 variants of UNIX, OS/2, NetWare, and VMS [Steinke 1995]. RPC infrastructures are implemented within the Distributed Computing Environment (DCE) , and within Open Network Computing (ONC), developed by Sunsoft, Inc. These two RPC implementations dominate the current Middleware market [Rao 1995].

Costs and Limitations

RPC implementations are nominally incompatible with other RPC implementations, although some are compatible. Using a single implementation of a RPC in a system will most likely result in a dependence on the RPC vendor for maintenance support and future enhancements. This could have a highly negative impact on a system's flexibility, maintainability, portability, and interoperability.

Because there is no single standard for implementing an RPC, different features may be offered by individual RPC implementations. Features that may affect the design and cost of a RPC-based application include the following:

- support of synchronous and/or asynchronous processing
- support of different networking protocols
- support for different file systems
- whether the RPC mechanism can be obtained individually, or only bundled with a server operating system

Because of the complexity of the synchronous mechanism of RPC and the proprietary and unique nature of RPC implementations, training is essential even for the experienced programmer.

Alternatives

Other middleware technologies that allow the distribution of processing across multiple processors and platforms are

- Object Request Brokers (ORB)
- Distributed Computing Environment (DCE)
- Message-Oriented Middleware (MOM)
- COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities)
- Transaction Processing Monitor Technology
- Three Tier Software Architectures

Complementary Technologies

RPC can be effectively combined with Message-Oriented Middleware (MOM)-MOM can be used for asynchronous processing.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Remote Procedure Call
Application category	Client/Server (AP.2.1.2.1) Client/Server Communication (AP.2.2.1)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1) Portability (QM.4.2) Complexity (QM.3.2.1)
Computing reviews category	Distributed Systems (C.2.4)

References and Information Sources

[Birrell 84]	Birrell, A.D. & Nelson, B.J. "Implementing Remote Procedure Calls." *ACM Transactions on Computer Systems 2*, 1 (February 1984): 39-59.
[Rao 95]	Rao, B.R. "Making the Most of Middleware." *Data Communications International 24*, 12 (September 1995): 89-96.
[Steinke 95]	Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

[Thekkath 93] Thekkath, C.A. & Levy, H.M. "Limits to Low-Latency Communication on High-Speed Networks." *ACM Transactions on Computer Systems 11*, 2 (May 1993): 179-203.

Current Author/Maintainer

Cory Vondrak, TRW, Redondo Beach, CA

Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM
10 Jan 97 (original)

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Requirements Tracing--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

The development and use of requirements tracing techniques originated in the early 1970s to influence the *completeness*, *consistency*, and *traceability* of the requirements of a system. They provide an answer to the following questions:

- What mission need is addressed by a requirement?
- Where is a requirement implemented?
- Is this requirement necessary?
- How do I interpret this requirement?
- What design decisions affect the implementation of a requirement?
- Are all requirements allocated?
- Why is the design implemented this way and what were the other alternatives?
- Is this design element necessary?
- Is the implementation compliant with the requirements?
- What acceptance test will be used to verify a requirement?
- Are we done?
- What is the impact of changing a requirement [SPS 94]?

The purpose of this technology description is to introduce the key concepts of requirements tracing. Detailed discussions of the individual technologies can be found in the referenced technology descriptions.

Technical Detail

Requirements traceability is defined as the ability to describe and follow the life of a requirement, in both a forward and backward direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases) [Gotel 95]. It can be achieved by using one or more of the following techniques:

- *Cross referencing*. This involves embedding phrases like "see section x" throughout the project documentation (e.g., tagging, numbering, or

indexing of requirements, and specialized tables or matrices that track the cross references).

- *Specialized templates and integration or transformation documents.* These are used to store links between documents created in different phases of development.
- *Restructuring.* The documentation is restructured in terms of an underlying network or graph to keep track of requirements changes (e.g., assumption-based truth maintenance networks, chaining mechanisms, constraint networks, and propagation) [Gotel 95].

Usage Considerations

For any given project, a key milestone (or step) is to determine and agree upon requirements traceability details. Initially, three important questions need to be answered before embarking on any particular requirements traceability approach:

1. What needs to be traceable?
2. What linkages need to be made?
3. How, when, and who should establish and maintain the resulting database?

Once the questions are answered, then selection of an approach can be made. One approach could be the structured use of general-purpose tools (e.g., hypertext editors, word processors, and spreadsheets) configured to support cross-referencing between documents. For large software development projects, an alternative approach could be the use of a dedicated workbench centered around a database management system providing tools for documenting, parsing, editing, decomposing, grouping, linking, organizing, partitioning, and managing requirements. Table 9 describes the strengths and weaknesses of each of the approaches.

Table 9: Comparing Requirements Tracing Approaches

Approaches	Strengths	Weaknesses
General purpose tools	· readily available · flexible · good for small projects	· need to be configured to support Requirements Traceability (RT) · potential high RT maintenance cost · limited control over RT information · potential limited integration with other software development tools

Workbenches	· fine-grained forward, backward, horizontal, and vertical RT · RT results may facilitate later development activities (i.e., testing) · suitable for large projects	· depend upon stakeholder buy-in · manual intervention may be required · RT in later development phases may be difficult

Regardless of the approach taken, requirements tracing requires a combination of models (i.e., representation forms), methods (i.e., step by step processes), and/or languages (i.e., semiformal and formal) that incorporate the above techniques. Some examples of requirements tracing methods are discussed in the following technology descriptions:

- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Argument-Based Design Rationale Capture Methods for Requirements Tracing

Maturity

Every major office tool manufacturer has spreadsheet and/or database capabilities that can be configured to support requirements tracing. There are at least ten commercial products that fall in the workbench category and support some level of requirements traceability [STSC 98]. At a minimum, they provide

- bidirectional requirement linking to system elements
- capture of allocation rationale, accountability, and test/validation
- identification of inconsistencies
- capabilities to view/trace links
- verification of requirements
- history of requirements changes.

Environments to support requirements traceability past the requirements engineering phase of the system/software life cycle are being researched. Areas include the development of a common language, method, model, and database repository structure, as well as mechanisms to provide data exchange between different tools in the environment. Prototypes exist and at least one commercial product provides support for data exchange through its object-oriented database facilities.

Costs and Limitations

In general, the implementation of requirements tracing techniques within an organization should facilitate reuse and maintainability of the system. However,

additional resources (time and manpower) to initially implement traceability processes (i.e., definition of traceability information, selection of automated tools, training, etc.) will be required. One case study found that the cost was more than twice the normal documentation cost associated with the development of a system of similar size and complexity. However, this was determined to be a one-time cost and the overall costs to maintain the software system are expected to be reduced. Almost immediate return was observed in the reduced amount of time to perform hardware upgrades [Ramesh 95].

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Requirements Tracing
Application category	Requirements Tracing (AP.1.2.3)
Quality measures category	Completeness (QM.1.3.1) Consistency (QM.1.3.2) Traceability (QM.1.3.3) Effectiveness (QM.1.1) Reusability (QM.4.4) Understandability (QM.3.2) Maintainability (QM.3.1)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2) Software Engineering Requirements/ Specifications (D.2.1)

References and Information Sources

[Bailin 90] Bailin, S., et al. "KAPTUR: Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationale," 95-104. *Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference*. Liverpool, NY, September 24-28, 1990. Rome, NY: Rome Air Development Center, 1990.

[Gotel 95] Gotel, Orlena. *Contribution Structures for Requirements Traceability*. London, England: Imperial College, Department of Computing, 1995.

[Ramesh 92] Ramesh, Balasubramaniam & Dhar, Vasant. "Supporting Systems Development by Capturing Deliberations During Requirements Engineering." *IEEE Transactions on Software Engineering 18,* 6 (June 1992): 498-510.

[Ramesh 95] Ramesh, Bala; Stubbs, Lt Curtis; & Edwards, Michael. "Lessons Learned from Implementing Requirements Traceability." *Crosstalk, Journal of Defense Software Engineering 8,* 4 (April 1995): 11-15.

[Shum 94] Shum, Buckingham Simon & Hammond, Nick. "Argumentation-Based Design Rationale: What Use at What Cost?" *International Journal of Human-Computer Studies 40,* 4 (April 1994): 603-652.

[SPS 94] *Analysis of Automated Requirements Management Capabilities.* Melbourne, FL: Software Productivity Solutions, 1994.

[STSC 98] Software Technology Support Center. *Requirements Management Tools* [online]. Available WWW <URL:http://www.stsc.hill.af.mil/RED/LIST.HTML> (1998).

Current Author/Maintainer

Liz Kean, Air Force Rome Laboratory

External Reviewers

Brian Gallagher, SEI

Modifications

4 Feb 98: added reference for [STSC 98]
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/reqtracing_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms

- Expansion of footnotes
- Lists of related topics

Certificates

Software Technology Roadmap

Status

Draft

Purpose and Origin

Digital certificates, or just certificates, are computer-based files or structures used to convey information about a user for identification purposes [Gerck 97]. Certificates are based on the ITU-T Recommendation X.509 [ITU-T 97]. There are a number of different certificates (called *end-entity certificates*) defined in the X.509 specification:

- *Personal certificates* represent individuals, and are typically used to secure e-mail and access to web servers.
- *Server certificates* indicate that a server belongs to the company it claims to belong to.
- *Developer certificates* are used by developers to sign software or other objects.

A certificate binds an identity (a name) to a public key. The certificate includes the name of the person (e.g., Bob), their public key (e.g., Bob's public key), and a digital signature sealing the data. This information can be verified (authenticated) by validating the digital signature.

Similar to the signature and stamp of a Notary Public, the digital signature is added by a trusted third party known as a *certificate authority* (CA). Certificate authorities confirm the relationship between identities and their public keys. Certificate authorities also publish public keys that then verify end-entity certificates. This process uses the public key of the authority that issued the certificate to validate the digital signature.

So, how do you get the public key of a certificate authority? In addition to end-entity certificates, the X.509 specification defines *certificate authority* certificates. These special certificates identify third party organizations entrusted to validate the identity of individuals requesting end-entity certificates. Similar to end-entity certificates, CA certificates contain a name (the name of the authority), a public key, and a digital signature sealing the data. CA certificates are critical in obtaining end-entity certificates and close the circle of trust.

Technical Detail

Certificates are obtained by sending a request to a certificate authority. Information about an individual (for personal certificates) a site, or company is sent to a CA along with a public key. The package sent to the CA is called a *certificate signing request* and is also defined in the X.509 specification.

Upon receiving the request, the certificate authority validates the contents through a process defined and published by the authority. The authority validates the digital signature placed on the signing request to ensure that it is a valid public key (i.e., it is part of a public/private key pair). Ultimately, confidence in the certificate is based on the trust you place on the certificate authority's assurance mechanism.

If the information contained within the certificate request is recognized as genuine, the authority generates the requested certificate. In addition, the authority *chains* their certificate to the new certificate. This allows an individual receiving the certificate to identify the authority that issued it, and to consider this information when deciding to accept or reject the certificate.

As this discussion demonstrates, the mathematical sophistication of public key encryption (PKI) ultimately rests upon a foundation of public trust. That is, we trust that a public key belongs to a particular individual, and this trust is vested in one or more authorities. Alice believes she possesses Bob’s public key because she trusts the authority that told her so.

Usage Considerations

Certificates are used when it is necessary to positively and uniquely identify and bind an end-entity for the purposes of non-repudiation, confidentiality, and integrity in public key encryption systems.

Maturity

In many cases use and application of public key cryptography requires the use of certificates (such notable exceptions include PGP [Pretty Good Privacy]) so that the owner of a public key can be known. Certificates are supposed to follow the X.509 specification which governs the format of certificates. X.509 is now in version 3 of the specification, commonly known as X.509v3. What is good about version 3 of the specification is that it permits certificate extensions - allowing application developers and system designers to embed additional information within the certificate itself (such as limitations on the specific use of a certificate). The X.509v3 specification is generally supported by all that report to support certificates, but known cases of incompatibilities exist between software that generates certificates and software that parses certificates. In most cases, the incompatibilities have been traced to poor implementation of the X.509v3 specification either in the generation or parsing of X.509v3 certificates. Some maturation of certificate technology should be expected.

Costs and Limitations

See Costs and Limitations associated with public key cryptography.

Although the specification of the base certificate (version 1 and version 2) is fairly mature (as it is backwards compatible), X.509v3 extension can cause incompatibilities between applications that generate and use certificates. Although many examples exist it is recommended that those looking into the use of certificate technology in public key cryptography systems learn about specific X.509v3 extension that are either required or expected to exist before selecting certificate management software.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Certificates
Application category	Information Security (AP.2.4)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating Systems Security & Protection (D.4.6), Security & Protection (K.6.5), Computer-Communications Networks Security and Protection (C.2.0)

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Gerck 97] Gerck, E. Overview of Certification Systems: X.509, CA, PGP and SKIP, Available WWW: <URL: http://www.mcg.org.br/cert.htm>

[ITU-T 97] ITU-T Recommendation X.509 (1997) | ISO/IEC 9594-8:1995 Information technology - Open Systems Interconnection - The Directory: Authentication framework.

Current Author/Maintainer

Scott A. Hissam, SEI

External Reviewers

Modifications

9 Dec 01 (original)

URL: http://www.sei.cmu.edu/str/descriptions/certificates_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Distributed/Collaborative Enterprise Architectures

Software Technology Roadmap

Status

Advanced

Note

We recommend Client/Server Software Architectures as prerequisite reading for this technology description.

Purpose and Origin

The distributed/collaborative enterprise architecture emerged in 1993. This software architecture is based on Object Request Broker (ORB) technology, but goes further than the Common Object Request Broker Architecture (CORBA) by using shared, reusable business models (not just objects) on an enterprise-wide scale.[1] The benefit of this architectural approach is that standardized business object models and distributed object computing are combined to give an organization *flexibility*, *scalability*, and *reliability* and improve organizational, operational, and technological effectiveness for the entire enterprise. This approach has proven more cost effective than treating the individual parts of the enterprise. For detailed information on distributed/collaborative enterprise architectures see Shelton and Adler [Shelton 93, Adler 95].

Technical Detail

The distributed/collaborative enterprise architecture allows a business to analyze its internal processes in new ways that are defined by changing business opportunities instead of by preconceived systems design (such as monolithic data processing applications). In this architectural design, an object model represents all aspects of the business; what is known, what the business does, what are the constraints, and what are the interactions and the relationships. A business model is used to integrate and migrate parts of legacy systems to meet the new business profile.

Distributed/collaborative enterprise builds its new business applications on top of distributed business models and distributed computing technology. Applications are built from standard interfaces with "plug and play" components. At the core of this infrastructure is an off-the-shelf, standards-based, distributed object

computing, messaging communication component such as an Object Request Broker (ORB) that meets Common Object Request Broker Architecture (CORBA) standards.

This messaging communication hides the following from business applications:

- the implementation details of networking and protocols
- the location and distribution of data, process, and hosts
- production environment services such as transaction management, security, messaging reliability, and persistent storage

The message communication component links the organization and connects it to computing and information resources via the organization's local or wide area network (LAN or WAN). The message communication component forms an enterprise-wide standard mechanism for accessing computing and information resources. This becomes a standard interface to heterogeneous system components.

Usage Considerations

The distributed/collaborative enterprise architecture is being applied in industries and businesses such as banking, investment, trading, credit-granting, insurance, policy management and rating, customer service, transportation and logistics management, telecommunications (long distance, cellular, and operating company), customer support, billing, order handling, product cross-selling, network modeling, manufacturing equipment, and automobiles [Shelton 93].

The most common implementations of objects and object models are written in C ++ or Smalltalk. Another popular language for implementing object and object models is Java.

Available for use in a distributed/collaborative enterprise architecture are products being built to open system standards, operating systems, database management systems, transaction processor monitors, and ORBs. These products are increasingly interchangeable.

Maturity

Since 1993 a number of companies have built and used distributed/collaborative architectures to address their long-term business needs because this model adapts to change and is built according to open system standards [Adler 95].

Costs and Limitations

Distributed/collaborative enterprise architectures are limited by the lack of commercially-available, object-oriented analysis and design method tools that focus on applications (rather than large scale business modeling).

Dependencies

The evolution of CORBA (see Common Object Request Broker Architecture) and COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities), and the results of standards bodies such as X/Open [X/Open 96] and Object Management Group (OMG) [OMG 96] will affect the evolution of distributed/collaborative architectures.

Alternatives

Three tier client/server architectures (see Three Tier Software Architectures) are an alternative approach to distributed/collaborative architectures. However, they do not address the need to evolve the business model over time as well as the distributed/collaborative architecture does.

Complementary Technologies

Distributed/collaborative enterprise architectures are enhanced by object-oriented design technologies (see Object-Oriented Design).

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Distributed/Collaborative Enterprise Architectures
Application category	Client/Server (AP.2.1.2.1)
Quality measures category	Scalability (QM.4.3) Reliability (QM.2.1.2) Maintainability (QM.3.1)
Computing reviews category	Distributed Systems (C.2.4) Software Engineering Design (D.2.10)

References and Information Sources

[Adler 95] Adler, R. M. "Distributed Coordination Models for Client/Sever Computing." *Computer 28,* 4 (April 1995): 14-22.

[Lewis 95] Lewis, T. G. "Where is Client/Server Software Headed?" *Computer 28,* 4 (April 1995): 49-55.

[OMG 96]	Object Management Group home page [online]. Available WWW <URL: http://www.omg.org> (1996).
[Shelton 93]	Shelton, Robert E. "The Distributed Enterprise (Shared, Reusable Business Models the Next Step in Distributed Object Computing)." *Distributed Computing Monitor 8*, 10 (October 1993): 1.
[X/Open 96]	*X/Open Web Site* [online]. Available WWW <URL: http://www.rdg.opengroup.org/> (1996).

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Larry Stafford, GTE

Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM
20 June 97: updated URLs for [OMG 96] and [X/Open 96]
10 Jan 97 (original)

Footnotes

[1] An enterprise is defined as a system comprised of multiple business systems or multiple subsystems.

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Distributed Computing Environment

Software Technology Roadmap

Status

Advanced

Note

We recommend Middleware *as prerequisite reading for this technology description.*

Purpose and Origin

Developed and maintained by the Open Systems Foundation (OSF), the Distributed Computing Environment (DCE) is an integrated distributed environment which incorporates technology from industry. The DCE is a set of integrated system services that provide an *interoperable* and *flexible* distributed environment with the primary goal of solving interoperability problems in heterogeneous, networked environments.

OSF provides a reference implementation (source code) on which all DCE products are based [OSF 96a].The DCE is *portable* and flexible- the reference implementation is independent of both networks and operating systems and provides an architecture in which new technologies can be included, thus allowing for future enhancements. The intent of the DCE is that the reference implementation will include mature, proven technology that can be used in parts- individual services- or as a complete integrated infrastructure.

The DCE infrastructure supports the construction and integration of client/server applications while attempting to hide the inherent *complexity* of the distributed processing from the user [Schill 93]. The OSF DCE is intended to form a comprehensive software platform on which distributed applications can be built, executed, and maintained.

Technical Detail

The DCE architecture is shown in Figure 10 [Schill 93].

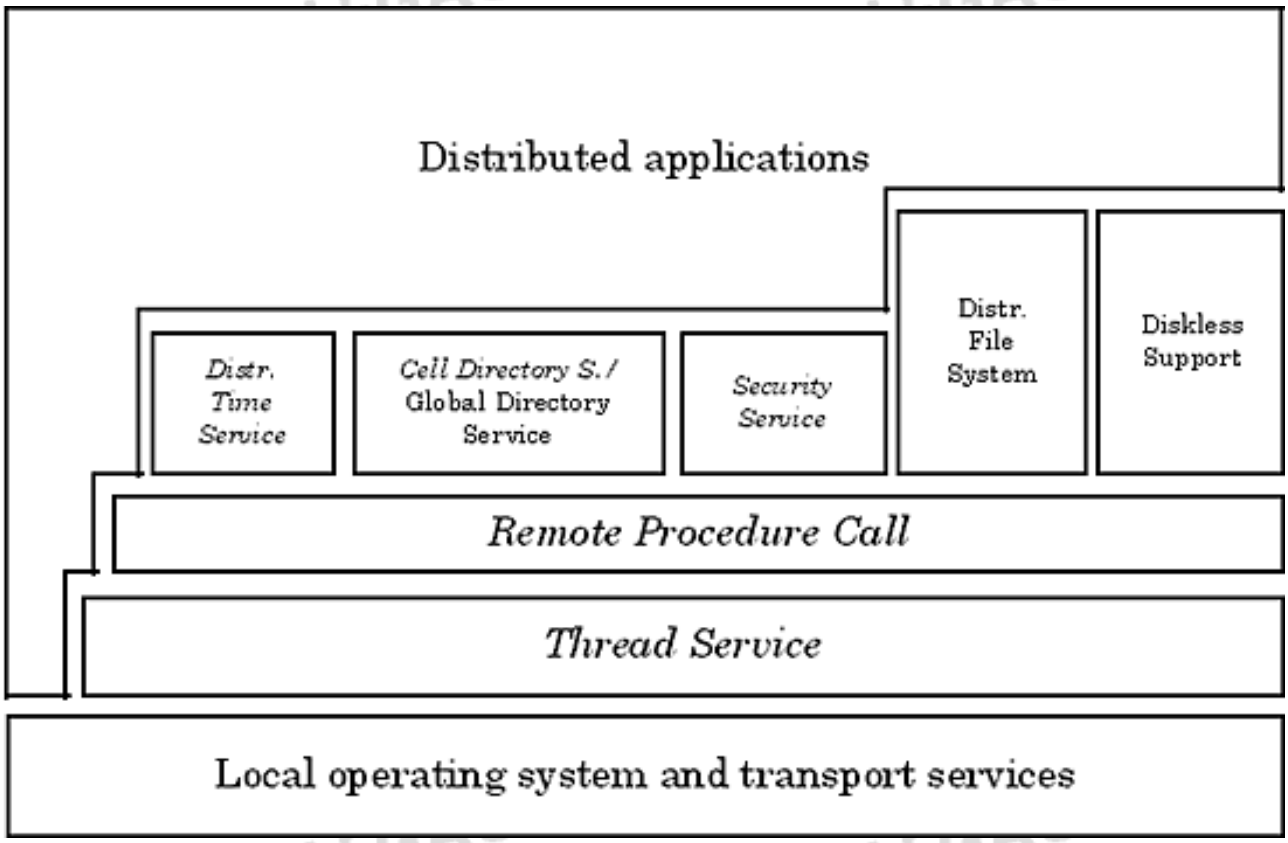


Figure 10: Distributed Computing Environment Architecture

DCE services are organized into two categories:

1. *Fundamental distributed services* provide tools for software developers to create the end-user services needed for distributed computing. They include
    - Remote Procedure Call, which provides portability, network *independence*, and *secure* distributed applications.
    - Directory services, which provide full X.500 support and a single naming model to allow programmers and maintainers to identify and access distributed resources more easily.
    - Time service, which provides a mechanism to monitor and track clocks in a distributed environment and accurate time stamps to reduce the load on system administrator.
    - Security service, which provides the network with authentication, authorization, and user account management services to maintain the *integrity*, privacy, and authenticity of the distributed system.
    - Thread service, which provides a simple, portable, programming model for building concurrent applications.
2. *Data-sharing services* provide end users with capabilities built upon the fundamental distributed services. These services require no programming on the part of the end user and facilitate better use of information. They include
    - Distributed file system, which interoperates with the network file system to provide a high-performance, *scalable*, and secure file access system.
    - Diskless support, which allows low-cost workstations to use disks on servers, possibly reducing the need/cost for local disks, and provides performance enhancements to reduce network overhead.

The DCE supports International Open Systems Interconnect (OSI) standards, which are critical

to global interconnectivity. It also implements ISO standards such as CCITT X.500, Remote Operations Service Element (ROSE), Association Control Service Element (ACSE), and the ISO session and presentation services. The DCE also supports Internet standards such as the TCP/IP transport and network protocols, as well as the Domain Name System and Network Time Protocol provided by the Internet.

Usage Considerations

The DCE can be used by system vendors, software developers, and end users. It can be used on any network hardware and transport software, including TCP/IP, OSI, and X.25. The DCE is written in standard C and uses standard operating system service interfaces like POSIX and X/Open guidelines. This makes the DCE portable to a wide variety of platforms. DCE allows for the extension of a network to large numbers of nodes, providing an environment capable of supporting networks of numerous low-end computers (i.e., PCs and Macintosh machines), which is important if downsizing and distributing of processing is desired. Because DCE is provided in source form, it can be tailored for specific applications if desired [OSF 96a].

DCE works internally with the client/server model and is well-suited for development of applications that are structured according to this model. Most DCE services are especially optimized for a structuring of distributed computing systems into a "cell" (a set of nodes/platforms) that is managed together by one authority.

For DCE, intra-cell communication is optimized and relatively secure and transparent. Inter-cell communication, however, requires more specialized processing and more complexity than its intra-cell counterpart, and requires a greater degree of programming expertise.

When using the thread services provided by DCE, the application programmer must be aware of thread synchronization and shared data across threads. While different threads are mutually asynchronous up to a static number defined at initialization, an individual thread is synchronous. The complexity of thread programming should be considered if these services are to be used.

DCE is being used or is planned for use on a wide variety of applications, including the following:

- *The Common Operating Environment*. DCE has been approved by DISA (Defense Information Systems Agency) as the distributed computing technology for the Common Operating Environment (COE) (see Defense Information Infrastructure Common Operating Environment).
- *The Advanced Photon Source (APS) system*. This is a synchrotron radiation facility under construction at Argonne National Laboratory.
- *The Alaska Synthetic Aperture Radar Facility (ASF)*. This is the ground station for a set of earth-observing radar spacecraft, and is one of the first NASA projects to use DCE in an operational system.
- *The Deep Space Network's Communications Complexes Monitor and Control Subsystem*. This project is deploying DCE for subsystem internal communications, with the expectation that DCE will eventually form the infrastructure of the entire information system.
- *The Multimission Ground Data System Prototype*. This project evaluated the applicability of DCE technology to ground data systems for support of JPL flight projects (Voyager, Cassini, Mars Global Surveyor, Mars Pathfinder).
- *Earth Observing Systems Data Information System*. This NASA system is one of the

largest information systems ever implemented. The system is comprised of legacy systems and data, computers of many varieties, networks, and satellites in space.
- *Command and control prototypes*. MITRE has prototyped command and control (C2) applications using DCE technology. These applications provide critical data such as unit strength, supplies, and equipment, and allow staff officers to view maps of areas of operation [OSF 96b].

Maturity

In early 1992, the OSF released the source code for DCE 1.0. Approximately 12 vendors had ported this version to their systems and had DCE 1.0 products available by June 1993. Many of these original products were "developer's kits" that were not robust, did not contain the entire set of DCE features (all lacked distributed file services), and were suited mostly for UNIX platforms [Chappell 93].

The DCE continues to evolve, but many large organizations have committed to basing their next generation systems on the DCE- over 14 major vendors provided DCE implementations by late 1994, when DCE 1.1 was released.

DCE 1.2.1, released in March 1996, provided the following new features:

- Interface definition language (IDL) support for C++ to include features such as inheritance and object references in support of object-oriented applications. This feature supports adoption of any object model or class hierarchy, thus providing developers with additional flexibility.
- Features to provide for coexistence with other application environments.
- Improvements over DCE 1.1 including enhancements to achieve greater reliability and better performance [OSF 96a].

Two other approaches to supporting objects are being considered besides the approach described for DCE 1.2:

1. Installing a CORBA-based product over DCE to provide additional support for distributed object technologies and a wide range of standardized service interfaces.
2. Integrating Network COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities) into the DCE infrastructure.

Costs and Limitations

DCE was not built to be completely object-oriented. The standard interfaces used by the DCE, as well as all the source code itself, are defined only in the C programming language. For object-oriented applications (i.e., applications being developed using an object-oriented language (see Object-Oriented Programming Languages) such as C++ or Ada 95, it may be more complex, less productive (thus more expensive), and less maintainable to use a non-object-oriented set of services like the DCE [Chappell 96].

Object-oriented extensions of the DCE have been developed by industry, but an agreed to vendor-neutral standard was still being worked in 1996.

Dependencies

Dependencies include Remote Procedure Call (RPC).

Alternatives

Alternatives include CORBA (see Common Object Request Broker Architecture), COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities), and message-oriented middleware (see Message-Oriented Middleware).

Complementary Technologies

DCE, in-part, has been used in building CORBA-compliant (see Common Object Request Broker Architecture) products as early as 1995. OSF is considering support for objects using COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities).

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Distributed Computing Environment
Application category	Distributed Computing (AP.2.1.2)
Quality measures category	Interoperability (QM.4.1) Portability (QM.4.2) Scalability (QM.4.3) Security (QM.2.1.5) Maintainability (QM.3.1) Complexity (QM.3.2.1) Throughput (QM.2.2.3)
Computing reviews category	Distributed Systems (C.2.4)

References and Information Sources

[Brando 96]	Brando, T. "Comparing CORBA & DCE." *Object Magazine 6*, 1 (March 1996): 52-7.
[Chappell 93]	Chappell, David. "OSF's DCE and DME: Here Today?" *Business Communications Review 23*, 7 (July 1993): 44-8.
[Chappell 96]	Chappell, David. *DCE and Objects* [online]. Available WWW <URL: http://www.opengroup.org/dce/info/dce_objects.htm> (1996).

[OSF 96a] Open Software Foundation. *The OSF Distributed Computing Environment* [online]. Available WWW <URL: http://www.osf.org/dce/> (1996).

[OSF 96b] Open Software Foundation. *The OSF Distributed Computing Environment: End-User Profiles* [online]. Available WWW URL: < http://www.osf.org/comm/lit/dce-eup/> (1996).

[Product 96] *DCE Product Survey Report* [online]. Available WWW <URL: http://nsdir.cards.com/Libraries/HTML/PDLC/DCE_prod_surv_rpt.html> (1996).

[Schill 93] Schill, Alexander. "DCE-The OSF Distributed Computing Environment Client/Server Model and Beyond," 283. *International DCE Workshop*. Karlsruhe, Germany, October 7-8, 1993. Berlin, Germany: Springer-Verlag, 1993.

Current Author/Maintainer

Cory Vondrak, TRW, Redondo Beach, CA

Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/dce_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Domain Engineering and Domain Analysis

Software Technology Roadmap

Status

Advanced

Purpose and Origin

The term domain is used to denote or group a set of systems or functional areas, within systems, that exhibit similar functionality. Domain engineering is the foundation for emerging "product line" software development approaches [Foreman 96], and affects the *maintainability*, *understandability*, *usability*, and *reusability* characteristics of a system or family of similar systems.

The purpose of this technology description is to introduce the key concepts of domain engineering and provide overview information about domain analysis. Detailed discussions of individual domain analysis methods can be found in the referenced technology descriptions.

Technical Detail

Domain engineering and domain analysis are often used interchangeably and/or inconsistently. Although domain analysis as a term may pre-date domain engineering, domain engineering is the more inclusive term, and is the process of

- defining the scope (i.e., domain definition)
- analyzing the domain (i.e., domain analysis)
- specifying the structure (i.e., domain architecture development)
- building the components (e.g., requirements, designs, software code, documentation)

for a class of subsystems that will support reuse [Katz 94].

Figure 11 [Foreman 96] shows the process and products of the overall domain engineering activity, and shows the relationships and interfaces of domain engineering to the conventional (individual) system development (application engineering) process. This has come to be known as the two life cycle model.

Domain engineering is related to system engineering, which is an integrated set of engineering disciplines that supports the design, development, and operation of large-scale systems [Eisner 94]. Domain engineering is distinguished from system engineering in that it involves designing assets[1] for a *set or class of multiple applications* as opposed to designing the best solution for a single application. In addition, system engineering provides the "whole solution," whereas domain engineering defines (i.e., limits) the scope of functionality addressed across multiple systems [Simos 96].

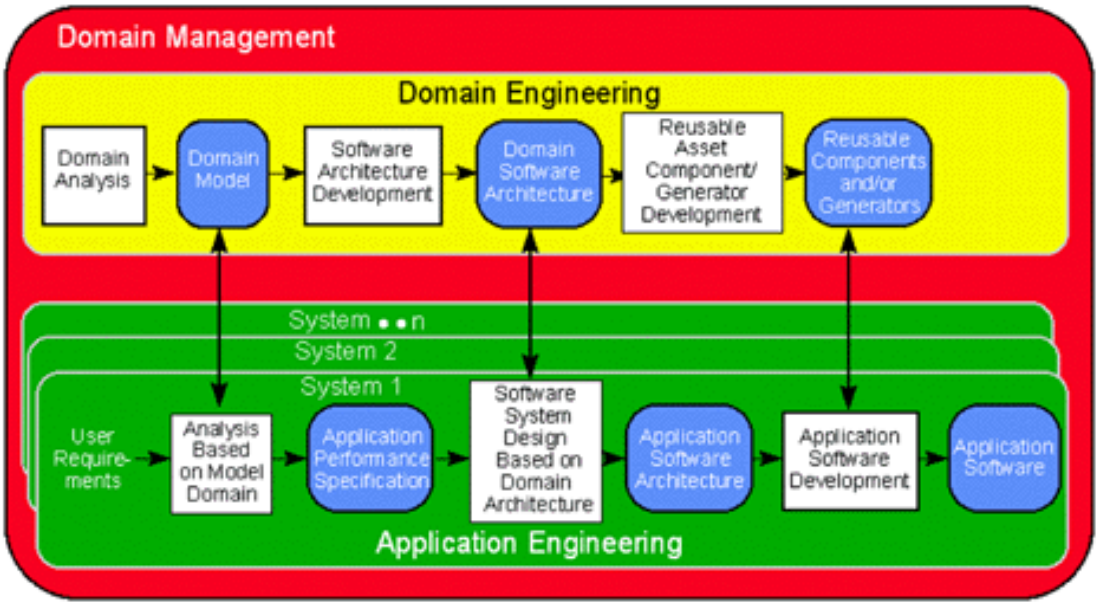


Figure 11: Domain Engineering and Application Engineering (Two Life Cycles)

Domain engineering supports systems engineering for individual systems by enabling coherent solutions across a family of systems: simplifying their construction, and improving the ability to analyze and predict the behavior of "systems of systems" composed of aggregations of those systems [Randall 96].

Domain analysis. Domain analysis (first introduced in the 1980s) is an activity within domain engineering and is the process by which information used in developing systems in a domain is identified, captured, and organized with the purpose of making it reusable when creating new systems [Prieto-Diaz 90]. Domain analysis focuses on supporting systematic and large-scale reuse (as opposed to opportunistic reuse, which suffers from the difficulty of adapting assets to fit new contexts) by capturing both the commonalities and the variabilities[2] of systems within a domain to improve the efficiency of development and maintenance of those systems. The results of the analysis, collectively referred to as a domain model, are captured for reuse in future development of similar systems and in maintenance planning of legacy systems (i.e., migration strategy) as shown in Figure 12 [Foreman 96].

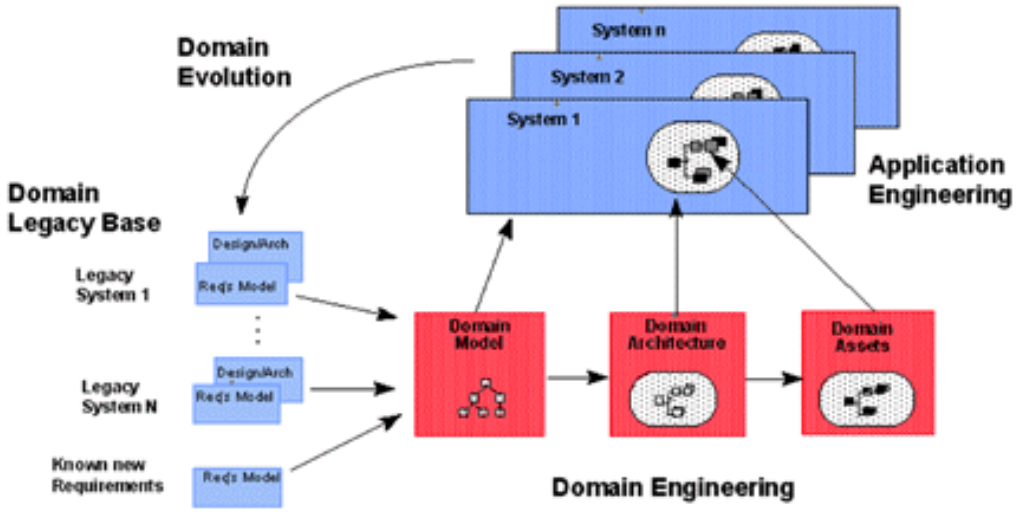


Figure 12: Domain Engineering and Legacy System Evolution

One of the major historical obstacles to reusing a software asset has been the uncertainty surrounding the asset. Questions to be answered included

- How does the software asset behave in its original context?
- How will it behave in a new context?
- How will adaptation affect its behavior [Simos 96]?

Design for reuse techniques (e.g., documentation standards, adaptation techniques) were developed to answer these questions; however, they did not provide the total solution, as a software asset's best scope needed to be determined (i.e., In which set of systems would the software asset be most likely reused?). Domain engineering and analysis methods were developed to answer more global questions, such as:

- Who are the targeted customers for the asset base (the designed collection of assets targeted to a specific domain)?
- Who are the other stakeholders in the domain?
- What is the domain boundary?
- What defines a feature of the domain?
- When is domain modeling complete?
- How do features vary across different usage contexts?
- How can the asset base be constructed to adapt to different usage contexts?

Goals of domain analysis include the following:

- Gather and correlate all the information related to a software asset. This will aid domain engineers in assessing the reusability of the asset. For example, if key aspects of the development documentation (e.g., chain of design decisions used in the development process) are available to a potential reuser, a more cost-effective reuse decision can be made.
- Model commonality and variability across a set of systems. This comparative analysis can reveal hidden contextual information in software assets and lead to insights about underlying rationale that would not have been discovered by studying a single system in isolation. It would answer questions like the following:
    - Why did developers make different design tradeoffs in one system than another?
    - What aspects of the development context influenced these decisions?
    - How can this design history be transformed into more prescriptive guidance to new developers creating systems within this domain?
- Derive common architectures and specialized languages that can leverage the software development process in a specific domain.

There is no standard definition of domain analysis; several domain analysis methods exist. Common themes among the methods include mechanisms to

- define the basic concepts (boundary, scope, and vocabulary) of the domain that can be used to generate a domain architecture
- describe the data (e.g., variables, constants) that support the functions and state of the system or family of systems
- identify relationships and constraints among the concepts, data, and functions within the domain
- identify, evaluate, and select assets for (re-)use
- develop adaptable architectures

Wartik provides criteria for comparing domain analysis methods [Wartik 92]. Major differences between the methods fall into three categories:

- *Primary product of the analysis.* In the methods, the results of the analysis and modeling activities may be represented differently. Examples include: different types of reuse library infrastructures (e.g., structured frameworks for cataloging the analysis results), application engineering processes, etc.
- *Focus of the analysis.* The methods differ in the extent they provide support for
    - context analysis: the process by which the scope of the domain is defined and analyzed to identify variability
    - stakeholder analysis: the process of modeling the set of stakeholders of the domain, which is the initial step in domain planning
    - rationale capture: the process for identifying and recording the reasoning behind the design of an artifact

    - scenario definition: mechanisms to capture the dynamic aspects of the system
    - derivation histories: mechanisms for replaying the history of design decisions
    - variability modeling: the process for identifying the ways in which two concepts or entities differ
    - legacy analysis: the process for studying and analyzing an existing set of systems
    - prescriptive modeling: the process by which binding decisions and commitments about the scope, architecture, and implementation of the asset base are made
- *Representation techniques.* An objective of every domain analysis method is to represent knowledge in a way that is easily understood and machine-processable. Methods differ in the type of representation techniques they use and in the ease with which new representation techniques can be incorporated within the method.

Examples of domain analysis methods include

- Feature-Oriented Domain Analysis (FODA), a domain analysis method based upon identifying the features of a class of systems, defines three basic activities: context analysis, domain modeling, and architecture modeling [Kang 90].
- Organization Domain Modeling (ODM), a domain engineering method that integrates organizational and strategic aspects of domain planning, domain modeling, architecture engineering and asset base engineering [Simos 96].

Randall, Arango, Prieto-Diaz, and the Software Productivity Consortium offer other domain engineering and analysis methods [Randall 96, Arango 94, Prieto-Diaz 91, SPC 93].

Usage Considerations

Domain analysis is best suited for domains that are mature and stable, and where context and rationale for legacy systems can be rediscovered through analysis of legacy artifacts and through consultation with domain experts. In general, when applying a domain analysis method, it is important to achieve independence from architectural and design decisions of legacy systems. Lessons learned from the design and implementation of the legacy system are essential; however, the over-reliance on precedented features and legacy implementations may bias new developments.

Maturity

See individual technologies.

Costs and Limitations

See individual technologies.

Complementary Technologies

Use of visual programming techniques can provide better understanding of key software assets like execution patterns, specification and design animations, testing plans, and systems simulation. Other complementary technologies include comparative/taxonomic modeling and techniques for the development of adaptable architectures/implementations (e.g., generation, decision-based composition).

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Domain Engineering and Domain Analysis
Application category	Domain Engineering (AP.1.2.4)
Quality measures category	Reusability (QM.4.4) Maintainability (QM.3.1) Understandability (QM.3.2)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2)

References and Information Sources

[Arango 94] Arango, G. "Domain Analysis Methods," 17-49. *Software Reusability*. Chichester, England: Ellis Horwood, 1994.

[Eisner 94] Eisner, H. "Systems Engineering Sciences," 1312-1322. *Encyclopedia of Software Engineering*. New York, NY: John Wiley and Sons, 1994.

[Foreman 96] Foreman, John. *Product Line Based Software Development-Significant Results, Future Challenges*. Software Technology Conference, Salt Lake City, UT, April 23, 1996.

[Hayes 94] Hayes-Roth, F. *Architecture-Based Acquisition and Development of Software: Guidelines and Recommendations from the ARPA Domain-Specific Software Architecture (DSSA) Program*. Palo Alto, CA: Teknowledge Federal Systems, 1994.

[IESE 98] Fraunhofer Institute for Experimental Software Engineering. *Domain Engineering Bibliography* [online]. Originally available WWW <URL:http://www.iese.fhg.de/ISE/DEbib/domain.html> (1998).

[Kang 90] Kang, K., et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-21, ADA 235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[Katz 94] Katz, S., et al. *Glossary of Software Reuse Terms*. Gaithersburg, MD: National Institute of Standards and Technology, 1994.

[Prieto-Diaz 90] Prieto-Diaz, R. "Domain Analysis: An Introduction." *Software Engineering Notes 15*, 2 (April 1990): 47-54.

[Prieto-Diaz 91] Prieto-Diaz, R. *Domain Analysis and Software Systems Modeling.* Los Alamitos, CA: IEEE Computer Society Press, 1991.

[Randall 96] Randall, Rick. *Space and Warning C2Product Line Domain Engineering Guidebook*, Version 1.0 [online]. Originally available WWW <URL: http://source.asset.com/stars/loral/domain/guide/delaunch.htm>

[Simos 96] Simos, M., et al. *Software Technology for Adaptable Reliable Systems (STARS) Organization Domain Modeling (ODM) Guidebook* Version 2.0 (STARS-VC-A025/001/00). Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996.

[SPC 93] *Reuse-Driven Software Processes Guidebook* Version 2.00.03 (SPC-92019-CMC). Herndon, VA: Software Productivity Consortium, 1993.

[Svoboda 96] Svoboda, Frank. *The Three "R's" of Mature System Development: Reuse, Reengineering, and Architecture* [online]. Available WWW <URL: http://source.asset.com/stars/darpa/Papers/ArchPapers.html> (1996).

[Wartik 92] Wartik, S. & Prieto-Diaz, R. "Criteria for Comparing Reuse-Oriented Domain Analysis Approaches." *International Journal of Software Engineering and Knowledge Engineering 2*, 3 (September 1992): 403-431.

Current Author/Maintainer

Liz Kean, Air Force Rome Laboratory

External Reviewers

Jim Baldo, MITRE, Washington, DC
Dick Creps, Lockheed Martin, Manassas, VA
Teri Payton, Lockheed Martin, Manassas, VA
Spencer Peterson, SEI
Rick Randall, Kaman Sciences, Colorado Springs, CO
Mark Simos, Organon Motives, Belmont, MA

Modifications

4 Feb 98: added reference for [IESE 98]
7 Oct 97: minor edits
10 Jan 97: (original)

Footnotes

[1] Examples include requirements, design, history of design decisions, source code, and test information.

[2] Commonality and variability refer to such items as functionality, data items, performance attributes, capacity, and interface protocols.

URL: http://www.sei.cmu.edu/str/descriptions/deda_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Feature-Based Design Rationale Capture Method for Requirements Tracing

Software Technology Roadmap

Status

Advanced

Note

We recommend Requirements Tracing--An Overview *as prerequisite reading for this technology description.*

Purpose and Origin

A design rationale is a representation of the reasoning behind the design of an artifact. The purpose of a feature-based design rationale capturing method is to provide mechanisms to track for each feature of the system; the description of the engineering decision that the feature represents includes

- the summary of the tradeoffs that were considered in arriving at the decision
- the ultimate rationale for the decision

The idea for feature-based design rationale capture originated during the performance of domain analysis in a software development project [Bailin 90]. The need to reverse engineer the rationales for various decisions suggested that a reuse environment should not simply present to the developer a set of alternative architectures that have been used on previous systems. It is necessary to present the rationales and issues involved in choosing among the alternatives. The feature-based approach evolved from the argumentation-based design rationale capture methods (see Argument-Based Design Rationale Capture Methods for Requirements Tracing). The major difference between the approaches is that the knowledge is organized around distinctive features of a system (feature-based) rather than around issues raised during the development process (argument-based).

Replaying the history of design decisions facilitates the understanding of the evolution of the system, identifies decision points in the design phase where alternative decisions could lead to different solutions, and identifies dead-end solution paths. The captured knowledge should enhance the *evolvability* of the

system and the *reusability* of components in the system.

Technical Detail

In the feature-based design rationale capture method, a feature is any distinctive or unusual aspect of a system, or a manifestation of a key engineering decision [Bailin 90]. (Note: The definition of a feature in this context is different from a feature in Feature-Oriented Domain Analysis(FODA) , in which a feature is a user-visible aspect or characteristic of the domain [Kang 90].) The features in a system make this system different from any other system in the domain. Examples of categories of features are: operational, interface, functional, performance, development methodology, design, and implementation. Each feature has a list of tradeoffs and rationale associated with it. Representations of the set of features may be entity relationship, dataflow, object communication, assembly, classification, stimulus-response, and state transition diagrams. The purpose of the multiple representations or views is to add flexibility in responding to evolving design paradigms, life cycle models, etc. A new way of looking at a system can be represented by adding a new view or way of looking at the features of the system. This provides a uniqueness and strength to this method that does not exist in other design rationale capturing methods. This approach makes the software engineering process become a process of answering questions about the features of a system rather than a cookbook-like procedure defined by a particular development method.

Usage Considerations

The use of this technology is oriented toward the entire organization, rather than single projects, because the big payoff occurs when a substantial database of corporate knowledge is organized and maintained. If an organization builds the same types of systems, the knowledge acquired in previous developments can be reused. Since the organization of information is around the features of a system as opposed to the issues that arise during a development project, only the issues that observably affect the content of the resulting system are saved.

The use of this technology requires the development of a shared, consistent, and coherent policy by a project team. A procedure for overall coordination must be developed.

Maturity

To date, there is at least one commercially-available tool to support the feature-based design rationale capture method. It is not a highly automated tool, but rather a bookkeeper to support an experience-based, learning-based development process. The commercial tool is based upon a prototype that has been used in laboratory experiments. The feature-based design rational capture method was used on the Software Technology for Adaptable, Reliable Systems (STARS) program to support the Organization Domain Modeling (ODM) process [Lettes 96].

Costs and Limitations

Feature-based design rationale capture methods and supporting tools require additional time and effort throughout the software life cycle. The system is described using multiple views that must be generated and maintained throughout the life of the project. Depending upon the size of the system, the number of views could be large. There is no integrated view of the system and this must be accomplished either mentally by the engineers on the project or through the use of an additional tool/technique. Training for the project team as well as the potential reuser is essential to make effective use of the method.

Alternatives

There are several alternative approaches to requirements traceability methods. Examples include Argument-Based Design Rationale Capture Methods for Requirements Tracing, an approach centered around the debate process (i.e., arguments and their resolution) that occurs during requirements analysis, and the Process Knowledge Method, an extension of the argument-based approach that includes a formal representation to provide two way traceability between requirements and artifacts and facilities for temporal reasoning (i.e., mechanisms to use the captured knowledge).

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Feature-Based Design Rationale Capture Method for Requirements Tracing
Application category	Requirements Tracing (AP.1.2.3) Domain Engineering (AP.1.2.4)
Quality measures category	Completeness (QM.1.3.1) Consistency (QM.1.3.2) Traceability (QM.1.3.3) Effectiveness (QM.1.1) Reusability (QM.4.4) Understandability (QM.3.2) Maintainability (QM.3.1)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2) Software Engineering Design (D.2.10)

References and Information Sources

[Bailin 90] Bailin, S., et al. "KAPTUR: Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationale," 95-104. *Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference*. Liverpool, NY, September 24-28, 1990. Rome, NY: Rome Air Development Center, 1990.

[Gotel 95] Gotel, Orlena. *Contribution Structures for Requirements Traceability*. London, England: Imperial College, Department of Computing, 1995.

[Kang 90] Kang, K., et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-21, ADA 235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[Lettes 96] Lettes, Judith A. & Wilson, John. *Army STARS Demonstration Project Experience Report* (STARS-VC-A011/003/02). Manassas, VA: Loral Defense Systems-East, 1996.

[Ramesh 92] Ramesh, Balasubramaniam & Dhar, Vasant. "Supporting Systems Development by Capturing Deliberations During Requirements Engineering." *IEEE Transactions on Software Engineering 18*, 6 (June 1992): 498-510.

[Shum 94] Shum, Buckingham Simon & Hammond, Nick. "Argumentation-Based Design Rationale: What Use at What Cost?" *International Journal of Human-Computer Studies 40*, 4 (April 1994): 603-652.

Current Author/Maintainer

Liz Kean, Air Force Rome Laboratory

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/featbased_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Feature-Oriented Domain Analysis

Software Technology Roadmap

Status

Advanced

Note

Domain Engineering and Domain Analysis *provides overview information about domain analysis.*

Purpose and Origin

Feature-oriented domain analysis (FODA) is a domain analysis method based upon identifying the prominent or distinctive features of a class of systems. FODA resulted from an in-depth study of other domain analysis approaches [Kang 90]. FODA affects the *maintainability*, *understandability*, and *reusability* characteristics of a system or family of systems.

Technical Detail

The FODA methodology was founded on two modeling concepts: abstraction and refinement. Abstraction is used to create domain products from the specific applications in the domain. These generic domain products abstract the functionality and designs of the applications in a domain. The generic nature of the domain products is created by abstracting away "factors" that make one application different from other related applications. The FODA method advocates that applications in the domain should be abstracted to the level where no differences exist between the applications. Specific applications in the domain are developed as refinements of the domain products.

Domain Engineering and Domain Analysis identifies three areas to differentiate between domain analysis methods. Distinguishing features for FODA include the following:

Primary Product of the Analysis. The primary product of FODA is a structured framework of related models that catalog the domain analysis results.

Focus of Analysis. The FODA process is divided into three phases:

- *Context analysis.* The purpose of context analysis is to define the scope of a domain. Relationships between the domain and external elements (e. g., different operating environments, different data requirements, etc.) are analyzed, and the variabilities are evaluated. The results are documented in a context model (e.g., block diagram, structure diagram, dataflow diagram, etc.).
- *Domain modeling.* Once the domain is scoped, the domain modeling phase provides steps to analyze the commonalities and differences addressed by the applications in the domain and produces several domain models. The domain modeling phase consists of three major activities:
    - *Feature analysis.* During feature analysis, a customer's or end user's understanding of the general capabilities or features of the class of systems is captured. The features, which describe the context of domain applications, the needed operations and their attributes, and representation variations are important results because the features model generalizes and parameterizes the other models produced in FODA. Examples of features include: function descriptions, descriptions of the mission and usage patterns, performance requirements, accuracy, time synchronization, etc. Features may be defined as alternative, optional, or mandatory. Mandatory features represent baseline features and their relationships. The alternative and optional features represent the specialization of more general features (i.e., they represent what changes are likely to occur in different circumstances). For optimal benefit, the resulting features model should be captured in a tool with access to rule-based language(s) so dependencies among features can be maintained and understood.
    - *Information analysis.* During information analysis, the domain knowledge and data requirements for implementing applications in the domain are defined and analyzed. Domain knowledge includes relevant scientific theory and engineering practice, capabilities and uses of existing systems, past system development and maintenance experience and work products, design rationales, history of design changes, etc. The purpose of information analysis is to represent the domain knowledge in terms of domain entities and their relationships, and to make them available for the derivation of objects and data definitions during operational analysis and architecture modeling. The information model may be of the form of an entity relationship (ER) model, a semantic network, or an object-oriented (OO) model.
    - *Operational analysis.* During operational analysis, the behavioral characteristics (e.g., dataflow and control-flow commonalities and differences, finite state machine model) of the applications in a domain are identified. This activity abstracts and then structures the common functions found in the domain and the sequencing of those actions into an operational model. Common features and information model entities form the basis for the abstract functional model. Unique features and information model entities complete the functional model. The control and data flow of an individual application can be instantiated or derived from the operational model with appropriate adaptation.

- *Architecture Modeling.* This phase provides a software solution for applications in the domain. An architectural model, which is a high-level design for applications in a domain, is developed. It focuses on identifying concurrent processes and domain-oriented common modules. It defines the process for allocating the features, functions, and data objects defined in the domain models to the processes and modules.

Representation Techniques. The use of COTS methods or tools must be integrated on a case-by-case basis. Currently FODA has been integrated with tools that support object-oriented models, entity relationship models, and semantic networks.

Usage Considerations

Based upon early pilot projects applying the FODA method [Kang 90, Cohen 92], the following lessons learned should be considered:

- A clear definition of the users of the domain model is essential. They should be well-defined during the context analysis phase.
- Early identification of the domain experts and sources of information is important. Effectively working with domain experts is the best means to achieving adoption of the domain model by potential users.
- The need for automated support for the domain modeling phase was identified. No modeling tools that support the FODA approach to ER modeling (i.e., ER + semantic data modeling) exist. Integration with existing modeling capabilities is achieved on a case-by-case basis. FODA was integrated with Hamilton Technologies 001 tool suite [Krut 93]. The integration was not automatic and there were areas where the 001 capabilities did not meet the FODA requirements. These were resolved through workarounds and negotiations with Hamilton Technologies.

Maturity

The FODA method is well-defined and has been applied on both commercial and military applications. It was applied to the

- Army Movement Control Domain [Cohen 92]
- In-Transit Visibility Modernization (ITVMOD) domain analysis effort [Petro 95, Devasirvatham 94]
- Telecommunication Automated Prompt and Response Domain at NORTEL (Northern Telecom) [Schnell 96]

Training is available.

Costs and Limitations

For small projects, use of the simulation capabilities of a commercial tool like Statemate was effective during operational analysis in demonstrating the capabilities of a system; however, for large projects potential users must be

convinced that the model and tool can be effectively used to specify a new system of the scale needed. The ability to use a modeling tool that can both capture the domain model and produce prototype code to simulate a system based upon feature selection would benefit the FODA method.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Feature-Oriented Domain Analysis
Application category	Domain Engineering (AP.1.2.4)
Quality measures category	Reusability (QM.4.4) Maintainability (QM.3.1) Understandability (QM.3.2)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2)

References and Information Sources

[Cohen 92]	Cohen, Sholom G., et al. *Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain* (CMU/SEI-91-TR-28, ADA 256590). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.
[Devasirvatham 94]	Devasirvatham, Josiah, et al. *In-Transit Visibility Modernization Domain Scoping Report Comprehensive Approach to Reusable Defense Software* (STARS-VC-H0002/001/00). Fairmont, WV: Comprehensive Approach to Reusable Defense Software, 1994.
[Kang 90]	Kang, K., et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-21, ADA 235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.
[Krut 93]	Krut, Robert W. Jr. *Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology* (CMU/SEI-93-TR-11, ESC-TR-93-188) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

[Krut 96]	Krut, R. & Zalman, N. *Domain Analysis Workshop Report for the Automated Prompt & Response System Domain* (CMU/SEI-96-SR-001). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
[Peterson 91]	Peterson, A. Spencer & Cohen, Sholom G. *A Context Analysis of Movement Control Domain for the Army Tactical Command and Control System* (CMU/SEI-91-SR-03). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1991.
[Petro 95]	Petro, James J.; Peterson, Alfred S.; & Ruby, William F. *In-Transit Visibility Modernization Domain Modeling Report Comprehensive Approach to Reusable Defense Software* (STARS-VC-H002a/001/00). Fairmont, WV: Comprehensive Approach to Reusable Defense Software, 1995.
[Schnell 96]	Schnell, K.; Zalman, N.; & Bhatt, Atul. *Transitioning Domain Analysis: An Industry Experience* (CMU/SEI-96-TR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

Current Author/Maintainer

Liz Kean, Air Force Rome Laboratory

External Reviewers

Spencer Peterson, SEI

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/foda_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references

- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Firewalls and Proxies

Software Technology Roadmap

Status

Complete.

Note

We recommend Computer System Security--an Overview as prerequisite reading for this technology description.

Purpose and Origin

Firewalls were developed in the early 1990s as the use of the Internet rapidly expanded. Intruders external to an organization often try to break into computers on a network to gain unauthorized access, obtain information illegally, or cause damage. Malicious users can also reside internal to an organization on Intranets and Local Area Networks (LANs). The purpose of a firewall or firewall system (which comprises one or more computers performing specific functions) is to serve as one element of an organization's perimeter defense. The perimeter can be defined as what separates the external world from the internal network or what separates internal sub-networks with differing access requirements. Ultimately, a firewall implements policy that specifies constraints on what network traffic is allowed to move between two or more networks.

A proxy is a software program that runs on a firewall system. It handles service requests between two networks by managing two connections: one between the requestor and the proxy server and one between the proxy server and the destination service. It evaluates all incoming and outgoing messages for a given service to determine if the message should be permitted to continue through to its destination network or blocked. Proxies are often provided for services such as email, FTP, Telnet, and World Wide Web (WWW) access.

Technical Detail

Firewall Architectures. A firewall can play several roles. It can be the primary line of defense against external threats from public networks such as the Internet. It can implement internal network partitioning to enforce access restrictions and protect against insider attacks. It can provide protection when interacting with partner networks and when merging with new organizational units (particularly those operating less securely). And it can serve as a central point where security policies can be implemented and logging/monitoring can occur. As shown in the figure below, there are

typically three basic firewall architectures:

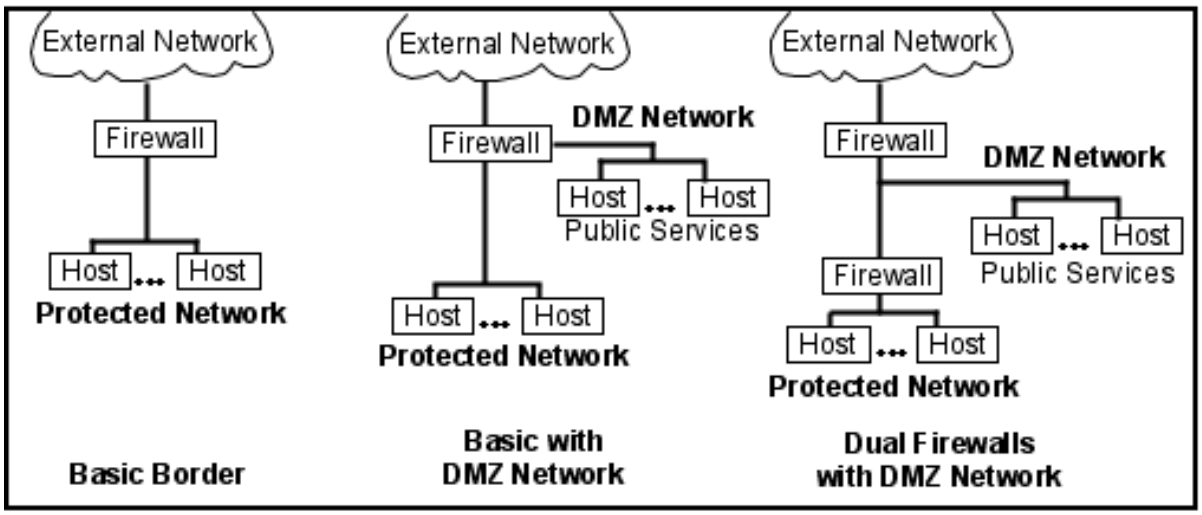


Figure: Three Firewall Architectures

The simplest approach is the Basic Border Firewall. The firewall includes a screening router and it performs certain packet filtering functions. The firewall host can be configured as a "Bastion Host," that is, a host that is minimally configured (containing only necessary software/services) and carefully managed to be as secure as possible[1]. This architecture is sometimes referred to as a Screened Host.[2]

The Basic with DMZ Network is a more secure architecture for protecting hosts that offer public services such as WWW as well as protecting the internal network from external users accessing public services. The firewall examines all incoming traffic to determine if it should be passed to the DMZ network (where one or more hosts providing public services reside) or to the protected network. It examines all outgoing traffic to determine if it should be passed from the protected network to the DMZ network (requesting public services), to the protected network from the DMZ network (responding to public service requests), or to the external world. This firewall architecture may also be referred to as a Dual-Homed Gateway (due to having two network connections, one to the DMZ Network and one to the protected network).[3]

One of the most secure firewall architectures is the Dual Firewalls with DMZ Network, sometimes referred to as a Sub-Network Firewall. In this architecture, the protected network is further isolated from the hosts offering public services and the external world by adding a second firewall host. By protecting the public services network with one firewall host and the protected network with a second firewall host (creating an additional DMZ between the two firewalls), traffic between the protected network and the Internet must traverse two firewalls.

Each firewall architecture can support one or more of the functions described below.

Firewall Functions. Static packet filters are "rules" that permit and deny Internet Protocol (IP) packets based on the contents of fields in the packet header (such as source/destination address, source/destination port, and protocol type). Each packet is processed individually with no reference as to what packets precede or follow. Dynamic packet filtering takes static packet filtering one step further by maintaining a connection

table in order to monitor the state or context of a communication session by attempting to match up outgoing and incoming packets. The information retained in the table usually includes the source and destination addresses and source and destination ports. Dynamic packet filtering is useful in handling "connectionless" protocols such as UDP[4] and ICMP[5] and is sometimes referred to as stateful filtering or stateful inspection.

A *proxy* is a software program that runs on a firewall. It understands the service protocol that it is responsible for processing, it implements protocol/service-specific security such as access control and levels of authentication, and makes all packet-forwarding decisions. Proxy servers evaluate the request and decide to permit or deny it based on a set of rules that apply to the individual network service (e.g., SMTP[6] for email, HTTP[7] for WWW, FTP,[8] Telnet, etc.) as well as host/user permissions. Proxy servers mirror the service as if it were running on the destination host [Smith 01]. Proxies provide a greater level of security by ensuring that two connecting hosts never exchange packets directly. Given they operate at the application layer in the OSI 7-layer protocol,[9] proxies can filter based on packet content, and provide a central point for more sophisticated and relevant alerts and logging information. Proxies can be transparent (totally invisible to the end user) or non-transparent (requiring some level of client knowledge and software configuration).

Network Address Translation (NAT) allows protected network users to gain access to the external network without allowing outsiders to get in. When a request is sent through the firewall, the NAT application substitutes its own address for the source address field. When a reply comes back to the NAT application, it replaces its own address in the destination field with that of the original client making the request. With NAT, external hosts cannot find the internal host addresses because they are aware of only one IP address, the firewall. The ability to attack internal hosts is greatly reduced using by employing NAT [Ogletree 01], [Smith 01]. Three NAT variations include static, dynamic, and overloading or port address translation [Tyson/Cisco].

Usage Considerations

In a single-layer architecture (Basic Border Firewall, Basic with DMZ Network), one network host is allocated all firewall functions and is connected to each network for which it is to control access. This approach is usually chosen when cost is a primary factor or when there are only two networks to interconnect. It has the advantage that everything there is to know about the firewall resides on the firewall host. In cases where the policy to be implemented is simple and there are few networks being interconnected, this approach can also be very cost-effective to operate and maintain over time. The greatest disadvantage of the single layer approach is its susceptibility to implementation flaws or configuration errors&emdash;depending on the type, a single flaw or error might allow firewall penetration.

In a multiple-layer architecture (Dual Firewalls with DMZ Network), the firewall functions are distributed among a small number of hosts, typically connected in series, with DMZ networks between them. This approach is more difficult to design and operate, but can provide substantially greater security by diversifying the defenses being implemented. Although more costly, it is advisable to use different technology in each of these firewall hosts. This reduces the risk that the same implementation flaws or configuration errors will exist in every layer.

With respect to firewall functions, start with implementing static packet filters. Add dynamic filtering for more accurate policy implementation, greater control, a higher level of security, and lower risk. Use application proxies for additional policy implementation, for packet content management, and for controlling application-program-specific/service access. Most firewalls implement some form of NAT as a default feature.

Maturity

There are a large number of commercial and open source/freeware products available that implement some or all of the firewall architectures and functions described above. This is a very mature product market and continues to evolve based on changing threats to network security. Recent developments include some function merging between the capabilities of firewalls and intrusion detection systems. One source of firewall evaluation information is the 2001 ICSA Labs Firewall Buyers Guide available at http://www.icsalabs.com/html/communities/firewalls/buyers_guide2001/index.shtml. TruSecure's/ICSA's list of certified firewall products is available at http://www.trusecure.com/corporate/press/2003/labs012703.shtml.

Costs and Limitations

The major tradeoffs to perform when selecting firewall architectures and functions are availability, performance, security, and cost. Availability is achieved by a combination of reliability and redundancy. Start by choosing hardware and software components that are reliable. If the level of reliability achieved is insufficient, consider using redundant components to meet availability requirements. Performance analysis is predominantly based on the anticipated traffic through the firewall system. An organization may need multiple firewall hosts to distribute the load and handle traffic at an acceptable rate. With respect to security, weigh the use of single versus dual firewall systems at the network perimeter. The factors to consider include:

- having outside traffic passing through two firewall systems instead of one (benefits vs. cost)
- ability to monitor traffic and the monitoring locations
- ability to recover from compromises including disconnecting one firewall system while keeping the other operational
- number of network ports needed
- performance
- failure characteristics
- expense

The Basic Border firewall is the least expensive to operate and maintain but also the least secure. Using only one firewall is a point of organizational and network vulnerability that needs to be managed from a risk perspective. The Basic with DMZ Network provides an additional level of protection for servers hosting public services but requires additional effort for ongoing operation and maintenance. The Dual Firewalls with DMZ Network is the most secure but also the most expensive to maintain and operate.

Dependencies

Firewall technology is driven by the capabilities of rapidly changing networking technologies, and the growing sophistication of intruder attack approaches. For instance, when Java applets became available on the WWW, it was possible to import malicious code hidden in the applets. To prevent this, it was desirable to block any Java applet at the firewall. If a proxy was being used in the firewall to filter WWW traffic, the proxy had to be enhanced to recognize Java applets from the WWW protocol. In addition, there is a growing number of products that perform email content and attachment examination and filtering. These products need to be integrated with firewall technologies so that both can work together effectively to protect organizational networks and hosts.

Alternatives

The security alternative to using firewalls to prevent theft of data or damage from malicious users is physical isolation of the networks. Doing so may conflict with mission performance needs if manual transfer of data from network to network is not acceptable. Data theft may be prevented through encryption, but that will not stop malicious damage.

Complementary Technologies

Complementary technologies include intrusion detection systems and content filtering applications. In a trusted computing environment, network security guards are a complementary technology as they provide similar functionality.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Firewalls and Proxies
Application category	System Security (AP.2.4.3)
Quality measures category	Vulnerability (QM.2.1.4.1) Security (QM.2.1.5)
Computing reviews category	Security & Protection (K.6.5) Computer-Communications Network Security and Protection (C.2.0)

References and Information Sources

[Allen 01] Allen, Julia. The CERT Guide to System and Network Security Practices. Boston, MA: Addison-Wesley, 2001.

[Fithen 99] Fithen, William, et al. Deploying Firewalls. (CMU/SEI-SIM-008). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. Online: http://www.cert.org/security-improvement/modules/m08.html.

[Cheswick 94] Cheswick, Willliam R.; & Bellovin, Steven M. Firewalls and Internet Security. Reading, MA: Addison-Wesley, 1994.

[Comer 95] Comer, Douglas E. Internetworking with TCP/IP, Vol. 1: Principles, Protocols,and Architecture. 3rd edition. New York: Prentice-Hall, 1995.

[Ogletree 00] Ogletree, Terry William. Practical Firewalls. Que, June 2000.

[Ranum 98, Curtin 00] Ranum, Marcus J.; & Curtin, Matt. "Internet Firewalls: Frequently Asked Questions," 1998, 2000. Available at http://www.interhack.net/pubs/fwfaq

[Smith 01] Smith, Gary. "A Brief Taxonomy of Firewalls - Great Walls of Fire." May 18, 2001. Available at http://www.sans.org/infosecFAQ/firewall/taxonomy.htm

[Stevens 94] Stevens, W. Richard. TCP/IP Illustrated, Vol. 1: The Protocols. Reading, MA: Addison-Wesley, 1994.

Tyson, Jeff] Tyson, Jeff. "How Network Address Translation Works." Online: http://www.howstuffworks/nat.htmand Cisco Systems Inc. "How NAT Works." Online: http://www.cisco.com/warp/public/556/nat-cisco.shtml

[Zwicky 00] Zwicky, Elizabeth. Cooper, Simon. Chapman, D. Brent. Building Internet Firewalls, 2d Edition. Sebastopol, CA: O'Reilly & Associates, June 2000.

Current Author/Maintainer

Julia Allen, Software Engineering Institute

External Reviewers

Numerous through review of [Fithen 99] and [Allen 01]

Modifications

10 Jan 1997: Original
12 Mar 2002: Update

URL: http://www.sei.cmu.edu/str/descriptions/firewalls_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Function Point Analysis

Software Technology Roadmap

Status

Advanced

Purpose and Origin

The function point metric was devised in 1977 by A. J. Albrecht, then of IBM, as a means of measuring software size and *productivity*. It uses functional, logical entities such as inputs, outputs, and inquiries that tend to relate more closely to the functions performed by the software as compared to other measures, such as lines of code. Marciniak provides a good capsule introduction to the application of function point measurement [Marciniak 94].

Function point definition and measurement have evolved substantially; the International Function Point User Group (IFPUG), formed in 1986, actively exchanges information on function point analysis (FPA) [IFPUG 96]. The original metric has been augmented and refined to cover more than the original emphasis on business-related data processing. FPA has become generally accepted as an effective way to

- estimate a software project's size (and in part, duration)
- establish productivity rates in function points per hour
- evaluate support requirements
- estimate system change costs
- normalize the comparison of software modules

However, uniformity of application and results are still issues (see Usage Considerations). For reasons explained below in Technical Detail, FPA has been renamed functional size measurement, but FPA remains the more commonly used term.

Technical Detail

Basic function points are categorized into five groups: outputs, inquiries, inputs, files, and Interfaces. A function point is defined as one end-user business function, such as a query for an input. This distinction is important because it tends to make a function point map easily into user-oriented requirements, but it also tends to hide internal functions, which also require resources to implement. To make up for this (and other) weaknesses, some refinements to and/or

variations of the basic Albrecht definition have been devised, including

- *Early and easy function points.* Adjusts for problem and data complexity with two questions that yield a somewhat subjective complexity measurement; simplifies measurement by eliminating the need to count data elements.
- *Engineering function points.* Elements (variable names) and operators (e.g., arithmetic, equality/inequality, Boolean) are counted. This variation highlights computational function [Umholtz 94]. The intent is similar to that of the operator/operand-based Halstead measures (see Halstead Complexity Measures).
- *Bang measure.* Defines a function metric based on twelve primitive (simple) counts that affect or show Bang, defined as "the measure of true function to be delivered as perceived by the user" [DeMarco 82]. Bang measure may be helpful in evaluating a software unit's value in terms of how much useful function it provides, although there is little evidence in the literature of such application. The use of Bang measure could apply when reengineering (either complete or piecewise) is being considered, as discussed in Maintenance of Operational Systems--An Overview.
- *Feature points.* Adds changes to improve applicability to systems with significant internal processing (e.g., operating systems, communications systems). This allows accounting for functions not readily perceivable by the user, but essential for proper operation.

Usage Considerations

There is a very large user community for function points; IFPUG has more than 1200 member companies, and they offer assistance in establishing a FPA program. The standard practices for counting and using function points are found in the IFPUG Counting Practices Manual [IFPUG 96]. Without some standardization of how the function points are enumerated and interpreted, consistent results can be difficult to obtain. Successful application seems to depend on establishing a consistent method of counting function points and keeping records to establish baseline productivity figures for your specific systems. Function measures tend to be independent of language, coding style, and software architecture, but environmental factors such as the ratio of function points to source lines of code will vary.

The proliferation of refinements and variations of FPA noted in Technical Detail has led to fragmentation. To remedy this, a Joint Technical Committee (JTC1) of the International Standards Organization (ISO) has been working since 1993 to develop ISO standards for sizing methods [Rehesaar 96]. This standardization effort is now called Functional Size Measurement.

Counting the function points needed for FPA remains largely a manual operation. This is an impediment to use. Wittig offers an approach to partial automation of function point counting [Wittig 94].

There are continuing concerns about the reliability and consistency of function point counts, such as

- whether two trained human counters will produce the same result for the same system
- the lack of inter-method reliability resulting from the variations described in Technical Detail

These reliability questions are addressed in a practical research effort described in Kemerer [Kemerer 93]. Siddiqee presents FPA as a good measure of productivity in a large software production environment in Lockheed Corporation [Siddiqee 93].

Any systematic FPA effort should collect the information into a database for ongoing analysis as the code is developed and/or modified.

Maturity

FPA is in use in many industrial software companies; IFPUG is large, with more than 1200 member companies, and offers many resources. As noted above, however, an ISO-level standard is still in the making.

Costs and Limitations

Currently, function point counting is a time-consuming and largely manual activity unless tools are built to assist the process. Wittig and Kemerer cite that it took more than five days to count a 4,000 function point system [Wittig 94, Kemerer 93]. However, the level of acceptance by software companies indicates that FPA is useful. Training in FPA is highly recommended; IFPUG can assist in securing training and locating FPA tools [IFPUG 96].

Alternatives

For estimation of effort, approaches based on lines of code (LOC) are an alternative. The now-classic COCOMO (constructive cost model) method and its REVIC (revised intermediate COCOMO) implementation provide a discipline for using LOC as a software size estimator [Boehm 81].

Complementary Technologies

LOC can also be used in a complementary sense as a check on results. There is also a technique called Backfiring that consists of a set of bidirectional equations for converting between function points and LOC [Jones 95]. This is reportedly useful when using sizing data from a combination of projects, some with metrics in LOC and some in function points. However, generalizing the Backfiring technique to yield a simple LOC-per-function point ratio is *not* advisable.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Function Point Analysis
Application category	Cost Estimation (AP.1.3.7)
Quality measures category	Productivity (QM.5.2)
Computing reviews category	Software Engineering Metrics (D.2.8) Software Engineering Management (D.2.9)

References and Information Sources

[Boehm 81] Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[DeMarco 82] DeMarco, Tom. *Controlling Software Projects: Management, Measurement, and Estimation*. New York, NY: Yourdon Press, 1982.

[Dreger 89] Dreger, J. Brian. *Function Point Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[Heller 95] Heller, Roger. "An Introduction to Function Point Analysis," *Crosstalk, Journal of Defense Software Engineering 8*, 11 (November/December 1995): 24-26.

[IFPUG 96] *The International Function Point Users' Group (IFPUG) Web site* [online]. Available WWW <URL: http://www.ifpug.org/> (1996).

[Jones 95] Jones, Capers. "Backfiring: Converting Lines of Code to Function Points." *IEEE Computer 28*, 11 (November 1995): 87-8.

[Kemerer 93] Kemerer, Chris. "Reliability of Function Points Measurement: A Field Experiment." *Communications of the ACM 36*, 2 (February 1993): 85-97.

[Marciniak 94] Marciniak, John J., ed. *Encyclopedia of Software Engineering*, 518-524. New York, NY: John Wiley & Sons, 1994.

[Rehesaar 96] Rehesaar, Hugo. "ISO/IEC Functional Size Measurement Standards," 311-318. *Proceedings of the GUFPI/IFPUG Conference on Software Measurement and Management*. Rome, Italy, February 5-9, 1996. Westerville, OH: International Function Point Users Group, 1996.

[Siddiqee 93] Siddiqee, M. Waheed. "Function Point Delivery Rates Under Various Environments: Some Actual Results," 259-264. *Proceedings of the Computer Management Group's International Conference*. San Diego, CA, December 5-10, 1993. Chicago, IL: Computer Management Group, 1993.

[Umholtz 94] Umholtz, Donald C. & Leitgeb, Arthur J. "Engineering Function Points and Tracking Systems."*Crosstalk, Journal of Defense Software Engineering 7*, 11 (November 1994): 9-14.

[Wittig 94] Wittig, G. E. & Finnie, G. R. "Software Design for the Automation of Unadjusted Function Point Counting," 613-623. *Business Process Re-Engineering Information Systems Opportunities and Challenges, IFIP TC8 Open Conference.* Gold Coast, Queensland, Australia, May 8-11, 1994. The Netherlands: IFIP, 1994.

Current Author/Maintainer

Edmond VanDoren, Kaman Sciences, Colorado Springs

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/fpa_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Graphic Tools for Legacy Database Migration

Software Technology Roadmap

Status

Advanced

Purpose and Origin

Graphic tools for legacy database migration are used to aid in the examination of legacy data in preparation for migration of one or more databases, often as part of a system migration or reengineering effort. They are intended to enhance *understandability* and *portability* of databases by providing easily-manipulated views of both content and structure that facilitate analysis [Selfridge 94].

Technical Detail

A graphical tool portrays a database's organization and data in graphical form. This enhances the understandability of the database(s) by allowing the analyst to assess the condition and organization of the data, including overlap and duplication of data items, in preparation for migration. This enhancement is desirable for several reasons:

1. Databases are typically complex, and may lack adequate documentation.
2. The information to be migrated may be contained in several separate databases built for different purposes.

The latter usually creates data redundancy, including multiple instances of a field, and even different representations of the same data (e.g., floating point in one place, fixed point or text in another). Important legacy information may be buried in text fields that must be found in order to capture the data's content. Bennett describes some of these problems [Bennett 95]. Legacy database migration is usually done to improve a system's *maintainability* (*modifiability*, *testability*, and/or ease of *life cycle evolution*). Database migration is typically performed as part of a larger system reengineering effort. It is a branch of database design and engineering, and requires the same set of disciplines.

Usage Considerations

A visualization tool is only part of the toolset of interest in migrating legacy data. Other tools might include the following:

- data modelers
- data entry and/or query screen translators
- report translators
- data-moving and translation utilities

A migration strategy is required to create a normalized file structure with referential integrity out of large, multiple databases. The tools must fit the environment, and the target database must be interfaced with the system and application software; this implies the need for *compatibility* with the languages used. The design of the target database can greatly affect *performance* and *maintainability*; therefore the first goal of the migration effort should be to define a target schema suitable for the application.

Maturity

Major database vendors offer tools of this type; the tools are typically optimized toward their database product as the target, but they accept other databases as input. There are also independent sources of visualization tools, as well as tools produced by research efforts [Selfridge 94, Gray 94]. Database migration, when offered as a service, often uses visualization tools to facilitate understanding between customer and consultant about the migration approach, process, and results [Ning 94].

Costs and Limitations

The cost of such a tool, including training, should be nominal compared to the total cost of the target database system's software, and may even be included. However, the migration itself can be costly in time and training; experience is required for good, normalized database design.

Dependencies

A migration effort would typically be coincident with a reengineering of the software that access the data, and would be intimately tied to the approaches used to do this reengineering.

Alternatives

An alternative to migration of the database is to link existing heterogeneous databases to each other. This approach eliminates the need to migrate the data, but also retains all the structural inefficiencies of the current databases, and may degrade performance. It may also create *maintainability* problems because each old database will require a separate knowledge set, and because their platforms may be not be supportable. The approach requires writing interface software that act as gateways to the other database management systems (DBMS), file systems, and/or other existing applications. The Object Request Broker technology exemplified by the emerging Common Object Request Broker Architecture (CORBA) standard, as well as products offered by commercial

database vendors, offer the capability to link existing heterogeneous databases. This includes the ability to associate data elements in different databases, and do JOINS across database boundaries.

Complementary Technologies

Other tools for analyzing data content and structure are available from commercial vendors and academic and research organizations. Knowledge-based approaches, for example, may have the ability to infer identity between multiple, differently-named instances of a data item. Other approaches such as these can compliment the use of graphical analyzers. Migration is typically done in the context of open systems (see COTS and Open Systems--An Overview), which implies a large number of technologies that would be helpful together.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Graphic Tools for Legacy Database Migration
Application category	Database Design (AP.1.3.2) Reengineering (AP.1.9.5)
Quality measures category	Understandability (QM.3.2) Maintainability (QM.3.1) Testability (QM.1.4.1) Compatibility (QM.4.1.1) Throughput (QM.2.2.3)
Computing reviews category	Database Management - Logical Design (H.2.1)

References and Information Sources

[Bennett 95] Bennett, K. "Legacy Systems: Coping With Stress." *IEEE Software 12*, 1 (January 1995): 19-23.

[Gray 94] Gray, W. A.; Wikramanayake, G. N.; & Fiddian, N. J. "Assisting Legacy Database Migration," 5/1-3. *IEE Colloquium: Legacy Information System- Barriers to Business Process Re-Engineering* (1994/246). London, UK, December 13, 1994. London, UK: IEE, 1994.

[Ning 94] Ning, Jim Q.; Engberts, Andre; & Kozaczynski, W. "Automated Support for Legacy Code Understanding." *Communications of the ACM 37*, 5 (May 1994): 50-57.

[Selfridge 94] Selfridge, Peter G. & Heineman, George T. "Graphical Support for Code-Level Software Understanding," 114-24. *Ninth Knowledge-Based Software Engineering Conference*. Monterey, CA, September 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

Current Author/Maintainer

Edmond VanDoren, Kaman Sciences, Colorado Springs

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/gtldm_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Graphical User Interface Builders

Software Technology Roadmap

Status

Advanced

Purpose and Origin

Graphical user interface (GUI) builders are software engineering tools developed to increase the productivity of user interface (UI) development teams, and to lower the cost of UI code in both the development and maintenance phases. One study found that an average of 48% of application code is devoted to the UI, and 50% of the development time required for the entire application is devoted to the UI portion [Myers 95]. Use of GUI builders can significantly reduce these numbers. For example, the MacApp system from Apple has been reported to reduce development time by a factor of four. Another study found that an application using a popular GUI tool wrote 83% fewer lines of code and took one-half the time compared to applications written without GUI tools [Myers 95]. Original GUI research was conducted at the Stanford Research Institute, Xerox Palo Alto Research Center, and Massachusetts Institute of Technology in the 1970s [Myers 95].

Technical Detail

A GUI development tool simplifies the coding of complex UI applications by providing the developer with building blocks (or widgets) of UI components. These building blocks are manipulated by the developer into a cohesive UI allowing a smaller workforce to develop larger amounts of user interface software in shorter time periods. A GUI builder enhances *usability* by providing a development team with a prototyping capability so that proposed UI changes can be rapidly demonstrated to the end user to secure requirements validation and acceptance. This aspect can decrease the turnaround time for making UI changes in the Operations and Maintenance (O&M) phase, which enhances *maintainability* as well.

GUI development tools can be broadly categorized into two types:

- Interface Development Tools (IDTs)
- User Interface Management Systems (UIMSs)

IDTs are used for building the interface itself, but nothing more. By contrast,

UIMSs extend the functionality of IDTs to include application development (code generation tools) or scripting tools. A UIMS also allows the developer to specify the behavior of an application with respect to the interface. These two types of GUI builders permit the interactive creation of the front-end GUI using a palette of widgets, a widget attribute specification form, a menu hierarchy (menu tree structure), a tool bar, and a view of the form. The UIMS adds the benefits of code generation tools, which can greatly increase the productivity of the GUI development staff. After the front-end is created by a UIMS, a code generator is used to produce C/C++ code, Motif User Interface Language (UIL) code, Ada code or some combination of C, Ada, and UIL.

Usage Considerations

GUI builders are useful for development of complex user interfaces because they increase software development speed by providing tools to lay out screens graphically and automatically generate interface code. Additionally, in applications that are susceptible to continuing user interface change such as command and control applications, the use of GUI builders greatly increases the ability to add/modify user interface functionality in minimal time to support mission changes or new requirements.

This technology works best when used in new development and reengineering. To take full advantage of the benefits of using GUI builders, the most desirable software architecture would be one that ensures the user interface software is isolated on a single layer as opposed to being embedded within several different software components. This isolation simplifies the UI portion of the software, thus making changes during development easier as well as enhancing future maintainability and evolvability.

Maturity

From the early GUI research started in the 1970s, GUI builder tools have grown into an estimated $1.2 billion business [Myers 95]. Today there are literally hundreds of GUI builders on the market supporting platforms ranging from UNIX to DOS. Virtually all new commercial and government applications use some form of UI builder tool. GUI builders have been successfully used on legacy systems when large changes or UI redesigns were applied to the user interface portion of the software [Myers 95].

Costs and Limitations

This technology requires workstations or PCs dedicated to support the development, rapid prototype, and validation of user interfaces. The most widely used GUI builders on the market today require minimal learning time for C and C ++ trained developers. These packages come with appropriate training materials, online help features, and vendor-supplied help lines which help make the developers productive in minimal time. There are few formal training costs associated with the use of GUI builders; however an organization would be well advised to provide internal training focusing on standardized approaches and techniques similar to design and coding standards for source code.

The prime costs with GUI builders are the initial license fees, annual maintenance agreements, and the cost of the workstations. In the UNIX environment, typical license costs for full UIMS GUI builders are in the range of $5k to $7.5k per single user license. For Windows or Macintosh environments, the costs range from $300 to $600 per user license. The maintenance agreements are key to keeping each GUI builder updated with vendor corrections and upgrades.

Dependencies

GUI development tools employ window managers as the foundation upon which a user interface can be built. A window manager allows the user to display, alter, and interact with more than one window at a time. The window manager's primary responsibility is to keep track of all aspects of each of the windows being displayed. In terms of numbers of applications in use, the two most popular window managers are Open Windows and Motif from Open Software Foundation (OSF) [OSF 96].

Alternatives

UI software can be developed without the use of GUI builders by using the features of window managers. For example, developers can use the X Windows based Motif (from OSF) and its rich set of widgets and features to design and implement UIs. This may be desirable for smaller, less complex UI applications for which the developer does not require the assistance (and extra cost) of GUI builders.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Graphical User Interface Builders
Application category	Interfaces Design (AP.1.3.3) Code (AP.1.4.2) Reapply Software Life Cycle (AP.1.9.3) Reengineering (AP.1.9.5)
Quality measures category	Usability (QM.2.3) Maintainability (QM.3.1)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2) User Interfaces (H.1.2)

References and Information Sources

[Myers 95]	Myers, Brad A. "User Interface Software Tools." *ACM Transactions on Computer-Human Interaction 2*, 1 (March 1995): 64-108.
[OSF 96]	OSF Home Page [online]. Available WWW <URL: http://www.osf.org> (1996).

Current Author/Maintainer

Mike Bray, Lockheed-Martin Ground Systems

External Reviewers

Brian Gallagher, SEI

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/guib_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Halstead Complexity Measures

Software Technology Roadmap

Status

Advanced

Note

We recommend that Maintainability Index Technique for Measuring Program Maintainability be read concurrently with this technology description. It illustrates a specific application of Halstead complexity to quantify the maintainability of software.

Purpose and Origin

Halstead complexity measurement was developed to measure a program module's *complexity* directly from source code, with emphasis on computational complexity. The measures were developed by the late Maurice Halstead as a means of determining a quantitative measure of complexity directly from the operators and operands in the module [Halstead 77]. Among the earliest software metrics, they are strong indicators of code complexity. Because they are applied to code, they are most often used as a maintenance metric. There are widely differing opinions on the worth of Halstead measures, ranging from "convoluted... [and] unreliable" [Jones 94] to "among the strongest measures of maintainability" [Oman 91]. The material in this technology description is largely based on the empirical evidence found in the Maintainability Index work, but there is evidence that Halstead measures are also useful during development, to assess code quality in computationally-dense applications.

Technical Detail

The Halstead measures are based on four scalar numbers derived directly from a program's source code:

n1 = the number of distinct operators

n2 = the number of distinct operands

N1 = the total number of operators

N2 = the total number of operands

From these numbers, five measures are derived:

Measure	Symbol	Formula
Program length	N	N= N1 + N2
Program vocabulary	n	n= n1 + n2
Volume	V	V= N * (LOG2 n)
Difficulty	D	D= (n1/2) * (N2/n2)
Effort	E	E= D * V

These measures are simple to calculate once the rules for identifying operators and operands have been determined (Szulewski notes that establishing these rules can be quite difficult [Szulewski 84]). The extraction of the component numbers from code requires a language-sensitive scanner, which is a reasonably simple program for most languages. Oman describes a tool for use in determining maintainability which, for Pascal and C, computes the following [Oman 91]:

V for each module; and

V(g), the average Halstead volume per module for a system of programs

For Pascal alone, the following are also computed:

E for each module; and

E(g), the average Halstead volume per module for a system of programs

Usage Considerations

Applicability. The Halstead measures are applicable to operational systems

and to development efforts once the code has been written. Because maintainability should be a concern during development, the Halstead measures should be considered for use during code development to follow complexity trends. A significant complexity measure increase during testing may be the sign of a brittle or high-risk module. Halstead measures have been criticized for a variety of reasons, among them the claim that they are a weak measure because they measure lexical and/or textual complexity rather than the structural or logic flow complexity exemplified by Cyclomatic Complexity measures . However, they have been shown to be a very strong component of the Maintainability Index measurement of maintainability (see Maintainability Index Technique for Measuring Program Maintainability). In particular, the complexity of code with a high ratio of calculational logic to branch logic may be more accurately assessed by Halstead measures than by Cyclomatic Complexity, which measures structural complexity.

Relation to other complexity measures. Marciniak describes all of the commonly-known software complexity measures and puts them in a common framework [Marciniak 94]. This is helpful background for any complexity measurement effort. Most measurement programs benefit from using several measures, at least initially; discarding those that do not suit the specific environment; and combining those that work (see Complementary Technologies). This is illustrated by Maintainability Index Technique for Measuring Program Maintainability, which describes the use of Halstead measures in combination with other complexity measures. When used in this context, the problems with establishing rules for identifying the elements to be counted are eliminated.

Maturity

Halstead measures were introduced in 1977 and have been used and experimented with extensively since that time. They are one of the oldest measures of program complexity. Because of the criticisms mentioned above, they have seen limited use. However, their properties are well-known and, in the context explained in Usage Considerations, they can be quite useful.

Costs and Limitations

The algorithms are free; the tool described in Technical Detail, contains Halstead scanners for Pascal and C, and some commercially-available CASE toolsets include the Halstead measures as part of their metric set. For languages not supported, standalone scanners can probably be written inexpensively, and the results can be exported to a spreadsheet or database to do the calculations and store the results for use as metrics. It should be noted that difficulties sometimes arise in uniquely identifying operators and operands. Consistency is important. Szulewski discusses this, defines consistent counting techniques for Ada, and points to other sources of counting techniques for some other languages [Szulewski 84]. Adding Halstead measures to an existing maintenance environment's metrics collection effort and then applying them to the software maintenance process will require not only the code scanner, but a collection system that feeds the resulting data to the metrics effort. Halstead measures may not be sufficient by themselves as software metrics (see Complementary

Technologies).

Alternatives

Common practice today is to combine measures to suit the specific program environment. Most measures are amenable for use in combination with others (although some overlap). Thus, many alternative measures are to some degree complementary. Oman presents a very comprehensive list of code metrics that are found in maintainability analysis work, and orders them by degree of influence on the maintainability measure being developed in that effort [Oman 94]. Some examples are (all are averages across the set of programs being measured)

- lines of code per module
- lines of comments per module
- variable span per module
- lines of data declarations per module

Complementary Technologies

Cyclomatic Complexity and its associated complexity measures measure the structural complexity of a program. Maintainability Index Technique for Measuring Program Maintainability, combines cyclomatic complexity with Halstead measures to produce a practical measure of maintainability.

Function point measures (see Function Point Analysis) provide a measure of functionality, with some significant limitations (at least in the basic function point enumeration method); the variant called engineering function points adds measurement of mathematical functionality that may complement Halstead measures.

Lines-of-code (LOC) metrics offer a gross measure of code, but do not measure content well. However, LOC in combination with Halstead measures may help relate program size to functionality.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Halstead Complexity Measures

Application category	Code (AP.1.4.2) Debugger (AP.1.4.2.4) Test (AP.1.4.3) Unit Testing (AP.1.4.3.4) Component Testing (AP.1.4.3.5) Reapply Software Life Cycle (AP.1.9.3) Reengineering (AP.1.9.5)
Quality measures category	Maintainability (QM.3.1) Testability (QM.1.4.1) Understandability (QM.3.2) Complexity (QM.3.2.1)
Computing reviews category	Software Engineering Distribution and Maintenance (D.2.7) Software Engineering Metrics (D.2.8) Complexity Classes (F.1.3) Tradeoffs Among Complexity Measures (F.2.3)

References and Information Sources

[Halstead 77]	Halstead, Maurice H. *Elements of Software Science, Operating, and Programming Systems Series* Volume 7. New York, NY: Elsevier, 1977.
[Jones 94]	Jones, Capers. "Software Metrics: Good, Bad, and Missing." *Computer 27, 9* (September 1994): 98-100.
[Marciniak 94]	Marciniak, John J., ed. *Encyclopedia of Software Engineering*, 131-165. New York, NY: John Wiley & Sons, 1994.
[Oman 91]	Oman, P. *HP-MAS: A Tool for Software Maintainability, Software Engineering* (#91-08-TR). Moscow, ID: Test Laboratory, University of Idaho, 1991.
[Oman 94]	Oman, P. & Hagemeister, J. "Constructing and Testing of Polynomials Predicting Software Maintainability." *Journal of Systems and Software 24*, 3 (March 1994): 251-266.
[Szulewski 84]	Szulewski, Paul, et al. *Automating Software Design Metrics* (RADC-TR-84-27). Rome, NY: Rome Air Development Center, 1984.

Current Author/Maintainer

Edmond VanDoren, Kaman Sciences, Colorado Springs

Modifications

10 Jan 97 (original)

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Intrusion Detection

Software Technology Roadmap

Status

Advanced

Note

We recommend Computer System Security--An Overview as prerequisite reading for this technology description.

Purpose and Origin

In the mid to late 1960s, as time sharing systems emerged, controlling access to computer resources became a concern. In the 1970s, the Department of Defense (DoD) Ware Report pointed out the need for computer security [Ware 79]. In the mid to late 1970s, a number of systems were designed and implemented using security kernel architectures. In the late 1970s, Tiger Teams began to evaluate the security of various systems. In 1983, the *Department of Defense Trusted Computer System Evaluation Criteria* - the "orange book" - was published and provided a set of criteria for evaluating computer security control effectiveness [DoD 85]. Research in this area continued through the 1980s, but many facets of computer security control remained a largely manual process. For example, the Internet Worm program of 1988 - which infected thousands of machines and disrupted normal activities for several days- was detected primarily through manual means [Spafford 88]. Today, there are primarily four approaches to achieving a secure computing environment [Kemmerer 94]:

1. the use of special procedures - such as password selection and use, access control, and manual review of output products- for working with a system
2. the inclusion of additional functions or mechanisms in the system
3. the use of assurance techniques - such as penetration analysis, formal specification and verification, and covert channel analysis - to increase one's confidence in the security of a system
4. the use of intrusion detection systems (IDSs)

The fourth approach, intrusion detection, is an emerging technology that seeks to automate the detection and elimination of intrusions. IDSs seek to increase

the *security* and hence the *availability*, *integrity*, and *confidentiality* of computer systems by eliminating unauthorized system/data access.

Technical Detail

Intrusion detection systems (IDSs) are predicated on the assumption that an intruder can be detected through an examination of various parameters such as network traffic, CPU utilization, I/O utilization, user location, and various file activities [Lunt 93]. System monitors or daemons convert observed parameters into chronologically sorted records of system activities. Called "audit trails," these records are analyzed by IDSs for unusual or suspect behavior. IDS approaches include

- Rule-Based Intrusion Detection
- Statistical-Based Intrusion Detection

IDSs designed to protect networks typically monitor network activity, while IDSs designed for single hosts typically monitor operating system activity.

Usage Considerations

Although IDSs are likely to increase the *security* of computer systems, the collection and processing of audit data will degrade system performance. Note that an IDS can be used to augment crypto-based security systems- which cannot defend against cracked passwords or lost or stolen keys- and to detect the abuse of privileges by authorized users [Mukherjee 94]. User authentication systems can be used to augment IDS systems.

Maturity

Prototypes of several intrusion detection systems have been developed, and some intrusion detection systems have been deployed on an experimental basis in operational systems. At least one network-based IDS - the Network Security Monitor (NSM) - successfully detected an attack in which an intruder exploited known security flaws to gain access to systems distributed over seven sites, three states, and two countries [Mukherjee 94]. However, additional work is required to determine appropriate levels of auditing, to strengthen the representation of intrusion attempts, and to extend the concept of intrusion detection to arbitrarily large networks [Lunt 93, Mukherjee 94].

Costs and Limitations

Audit trail analysis can be conducted either offline (after the fact) or in real time. Although offline analysis permits greater depth of coverage while shifting the processing of audit information to non-peak times, it can only detect intrusions after the fact. Real-time IDSs can potentially catch intrusion attempts before the

system state is compromised, but real-time IDSs must run concurrently with other system applications and will therefore negatively affect *throughput*. In addition to the costs associated with creating and analyzing audit trails, IDS systems cannot detect all intrusion attempts, primarily because only known intrusion scenarios can be represented. An intrusion attempt made using a scenario not represented by an IDS system may be successful, and some intrusion attempts have succeeded in either turning off the audit daemon or in modifying the audit data prior to its being processed by an IDS.

Although most IDSs are designed to support multiple operating systems, audit data collected by monitoring operating system activity will be operating system specific [Mukherjee 94]; this type of data may therefore need to be converted into a standard form before it can be processed by an IDS.

For these reasons, many IDS systems are designed as assistants to human computer security monitors.

Dependencies

System or network auditing tools and techniques are necessary enablers for this technology. Depending on the type of IDS, expert systems technology may also be needed.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Intrusion Detection
Application category	System Security (AP.2.4.3)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating Systems Security and Protection (D.4.6) Computer-Communication Networks Security and Protection (C.2.0) Security and Protection (K.6.5)

References and Information Sources

[DoD 85] *Department of Defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC)* (DoD 5200.28-STD 1985). Fort Meade, MD: Department of Defense, 1985. Also available WWW <URL: http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html> (1985).

[Kemmerer 94] Kemmerer, Richard A. "Computer Security," 1153-1164. *Encyclopedia of Software Engineering.* New York, NY: John Wiley and Sons, 1994.

[Lunt 93] Lunt, Teresa F. "A Survey of Intrusion Detection Techniques." *Computers and Security 12,* 4 (June 1993): 405-418.

[Mukherjee 94] Mukherjee, Biswanath, L.; Heberlein, Todd; & Levitt, Karl N. "Network Intrusion Detection." *IEEE Network 8,* 3 (May/June 1994): 26-41.

[Smaha 88] Smaha, Stephen E. "Haystack: An Intrusion Detection System," 37-44. *Proceedings of the Fourth Aerospace Computer Security Applications Conference*. Orlando, Florida, December 12-16, 1988. Washington, DC: IEEE Computer Society Press, 1989.

[Sundaram 96] Sundaram, Aurobindo. *An Introduction to Intrusion Detection* [online]. Available WWW <URL: http://www.acm.org/crossroads/xrds2-4/xrds2-4.html> (1996).

[Spafford 88] Spafford, Eugene H. *The Internet Worm Program: An Analysis* (CSD-TR-823). West Lafayette, IN: Purdue University, 1988.

[Ware 79] Ware, W. H. *Security Controls for Computer Systems: Report of Defense Science Board, Task Force on Computer Security*. Santa Monica, CA: The Rand Corporation, 1979.

Current Author/Maintainer

Mark Gerken, Air Force Rome Laboratory

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/intrusion_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Java(TM)

Software Technology Roadmap

Status

Advanced

Purpose and Origin

Java™ is an object-oriented programming language (see Object-Oriented Programming Languages) developed by a small team of people headed by James Gosling at Sun Microsystems (development began in 1991) [Sun 97e]. It was originally intended for use in programming consumer devices, but when the explosion of interest in the Internet began in 1995 it became clear that Java was an ideal programming language for Internet applications [van Hoff 96]. Java addresses many of the issues of software distribution over a network, including *interoperability*, *security*, *portability*, and *trustworthiness*. When they are embedded in a Web page, Java programs are called "applets." Applets, in conjunction with JavaBeans(tm)[Sun 99d] provide a developer the flexibility to develop a more sophisticated user interface on a Web page [Yourdon 96]. Java applets provide executable content, such as event-driven pop-up windows and graphical user interface (GUI) widgets (see Graphical User Interface Builders) via peer classes (see Figure 19), which can support a variety of applications. Java applets are the dominant player of client side Internet computing. However, the server side computing, i.e. the code that generates the HTML contents, was considered a stronghold of better performance languages as C++ or script languages as PERL. This situation is changing with the release of Java 2 Enterprise Edition(tm) (J2EE) [Sun 99a]. J2EE is a new Java platform specifically designed to address the needs of enterprise server side computing. J2EE provides *scalability*, *interoperability*, *reliability*, *security*. Java is also re-addressing its original purpose (consumer devices) through JINI(tm) connection technology [Sun 97b]. JINI enables devices to work together without the burden of setting up networks, loading drivers and so on. JINI devices such as TVs, DVDs and cameras will be able to self-install, self-organize into communities, self-configure, and self-diagnose. Jini technology reduces dependence on system administrators, potentially lowering support costs and allowing impromptu device communities to assemble in places far from the traditional office. JINI mainly addresses *usability*, *cost of ownership* and *interoperability*.

Technical Detail

Java is a high-level programming language similar in flavor to Smalltalk and similar in syntax to C and C++. However, the Java language is far less complex than C++. It is an object-oriented, statically typed language that is architecture-neutral, multi-threaded, and robust. It provides built-in garbage collection, supports a single-inheritance class hierarchy, and does not use pointers, thereby eliminating three of the primary sources of errors in many C++ programs. Because it borrows its syntax from the widely known C and C++, the Java language feels familiar to most developers. Java provides flexibility in that it provides dynamic functionality. Classes are linked in as required and can be downloaded from across networks. Incoming code is verified before execution. Such flexibility is a paradigm shift from the normal model of computing, which usually requires the entire suite of possible functionality to be installed onto a user's platform prior to execution [Yourdon 96]. Java programs start as Java source code, which is then compiled to bytecode and stored on a server or a local computer in ".class" files. In order to execute a Java program, a user invokes a Java Virtual Machine (JVM) that executes the Java bytecode. Unlike most other programming languages, Java bytecode is *not* platform-specific or native to any particular processor; it is a "write once, run anywhere" approach. This platform-neutrality at both source and binary levels means Java is inherently portable. The Java system also provides an extensive library of classes that provides access to the underlying operating system. All of today's popular Web browsers contain a Java Virtual Machine (JVM), including Netscape Navigator, Microsoft Internet Explorer, and Sun's HotJava Browser. Desktop platforms such as Microsoft Windows, MacOS, OS/2 Warp, and Sun Solaris also provide a standalone JVM which can execute Java code. A new generation of so-called Network Computers[1] executes Java code directly. Sun is extending the availability of the JVM to enable Java programs to be deployed on a wide range of consumer devices, such as pagers, telephones, and televisions [Clark 97]. The relationships of code, Java Virtual Machines, and platform independence/ neutrality is shown in Figure 19.

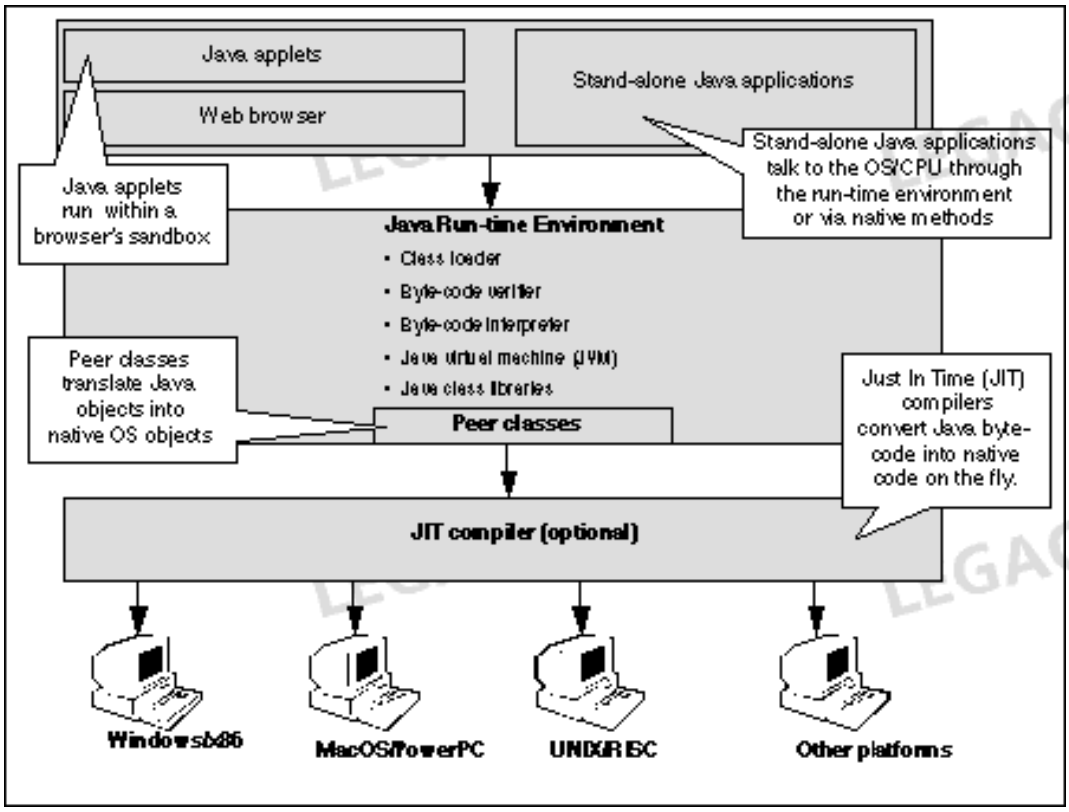


Figure 19: Multiple-Platform Application [Halfhill 97]

The Java platform provides portability, a measure of security, and inherent trustworthiness, including strong memory protection, encryption and signatures, rules enforcement, and runtime verification. Java is designed to allow applets to be downloaded and executed without introducing viruses or misbehaved code. It does this by placing strict limits on applets to prevent malicious actions. For example, applets cannot read from or write to the local disk. Unfortunately, while the Java model is theoretically secure, the various *implementations* of the JVM continue to show signs of weakness. Exploitation of security flaws in the implementations is still alarmingly common [Sun 97a]. An applet's actions are restricted to its "sandbox," an area of the Web browser dedicated to that applet and within which it may do anything it wants. But a Java applet can't read or alter any data outside its sandbox. Hence users can run untrusted Java code without compromising their trusted computing environments. Standalone windows created by Java applications are clearly labeled as being owned by untrusted software. Java applications are also prohibited from making network connections to other computers on a corporate Intranet, so malicious code can't exploit undiscovered security holes. Applets are not enough to build enterprise systems. Applets excel delivering functionality to remote clients, but enterprise applications need much more than remote access, like scalability and transactions. To address those needs Sun has developed Java(tm) 2 Platform, Enterprise Edition (J2EE). J2EE is a standard set of Java APIs that define a multi tier architecture (see Three Tier Software Architectures) suitable for the development, deployment, and management of enterprise applications written in the Java(tm) programming language. J2EE is functionally complete in the sense that it is possible to develop a large class of enterprise applications using only the J2EE APIs. Figure 20 illustrates the architecture of a J2EE application.

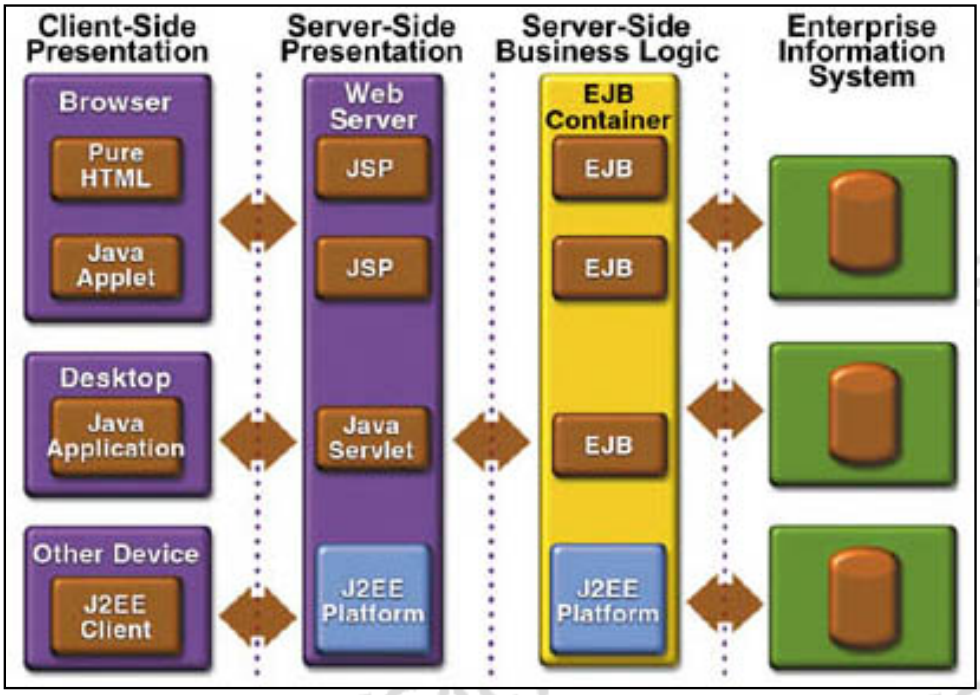


Figure 20: J2EE architecture [Sun 99f]

Remote clients are implemented as a combination of html pages and applets (or as Java Applications if Internet access is not required). The middle tier is split in two, the *Enterprise JavaBeans framework(tm)* (*EJB) [Sun 99e]* containing Enterprise Beans, which are reusable units that contain transactional business logic and the *Web Server* containing *JSP Pages and servlets that are* software entities that provide services in response to HTTP requests. The persistence layer can be implemented in any commercial database.

Usage Considerations

APIs. Java specifies a core set of Application Programming Interfaces (APIs) required in all Java implementations and an extended set of APIs covering a much broader set of functionality. The core set of APIs include interfaces for

- basic language types
- file and stream I/O
- network I/O
- container and utility classes
- abstract windowing toolkit

The extended set of APIs includes interfaces for 2D-rendering and 2D-animation; a 3D-programming model; telephony, time-critical audio, video, and MIDI[2] data; network and systems management; electronic commerce; and encryption and authentication [Hamilton 96]. J2EE introduces additional APIs to address specific need of enterprise environments. These APIs provide similar functionality to CORBA services including:

- Asynchronous communication through the Java Message Service (JMS)
- A naming service through the Java Naming and Directory Interface (JNDI)
- A transaction service through the Java Transaction API (JTA)
- Tabular data access though the JDBC API.

Platform-specific implementations. Recently, there has been some debate about the use of platform-specific APIs and the affect on the future of Java. For example, Microsoft's Internet Explorer 4.0 includes technology for J/Direct, which will provide a connection between Java and the Windows programming environment. Applications that make use of the J/Direct API will run only on the Windows platform, thereby curtailing one of Java's inherent benefits: platform neutrality. Providing Java developers with direct access to the Win32 API also breaks Java's security model and makes it more like Microsoft's platform-dependent ActiveX technology [Levin 97]. Sun has sued Microsoft for this practice, there is not a definitive resolution (by November 1999) but Microsoft has already been banned from using Java trademark with their modified versions.

Traning/education. The Java syntax for expressions and statements are almost identical to ANSI C, thus making the language easy to learn for C or C++ programmers. Because Java is a programming language, it requires a higher skill level for content developers than hypertext markup language (HTML). Programmers need to learn the Java standard library, which contains objects and methods for opening sockets, implementing the HTTP protocol, creating threads, writing to the display, and building a user interface. Java provides

mechanisms for interfacing with other programming languages such as C and existing libraries such as Xlib, Motif, or legacy database software.

Performance. Performance is a major consideration when deciding to use Java. In most cases, interpreted Java is much slower than compiled C or C++ (as much as 10-15 times slower). However, most recent versions of the popular Web browsers and Java development environments provide Just In Time (JIT) compilers that produce native binary code (while the program is loaded and executed) that is beginning to rival that of optimized C++. The Java 2 platform also provides the Java HotSpot(tm) Performance Engine [Sun 97c] that combines the functionality of a JIT with runtime optimizations that further improve Java performance. For real-time applications, the performance implications of the Java garbage collector should also be considered. Garbage collection may make it difficult to easily bound timing properties of the application.

Language migration. A number of items should be considered if migrating from C or C++ to Java, including the following:

- Java is totally object-oriented; thus everything must be done via a method invocation.
- Java has no pointers or parameterized types.
- Java supports multithreading and garbage collection.

Maturity

Java was made available to the general public in May 1995, and has enjoyed unprecedented rapid transition into practice. Web sites such as the Java Applet Rating Service (JARS) [JARS 97] and Gamelan [Gamelan 97] contain literally thousands of Java-based applications available for downloading. All of today's leading Web browsers provide support for Java by including a JVM as part of their product. There are multitudes of books available that describe all aspects of Java programming. Many commercial uses of Java have also appeared in a relatively short period of time. Sun provides a series of "customer success stories" at their web sites [Sun 97b, Sun 97c]. Some of the many commercial applications written in Java include

- TWSNet, a shipment tracking and processing application for CSX Corporation [Sun 96a]
- OC://WebConnect, a Web-based terminal emulation package for connecting to legacy SNA networks, from OpenConnect Systems
- via World Network, an online travel reservation system, from Andersen Consulting

There are now several development environments that support Java programming. These include IBM's Visual Age for Java, Symantec's Visual Café, Microsoft's J++, and Sun's Java Development Kit (JDK) [Sun 97d]. Most of these products provide integrated editors, debuggers, JIT compilers, and other tools commonly associated with computer-aided software engineering (CASE) tools. J2EE is one of the newest and less mature parts of Java. In fact, by November 1999 there is only a beta release of J2EE. Some constituents of the platform are quite stable but others are undertaking deep changes. EJB, for example, was

released in Dec 1997 and there already are more than thirty implementations [EJB-SIG 99] (including from IBM, BEA, Oracle and IONA). However, EJB has suffered important changes from the 1.1 to the 1.2 release and that volatility is expected to continue with subsequent releases. In summary, J2EE is currently usable but there are several important issues to be solved and some time is needed until it delivers all its potential.

Costs and Limitations

Java and the source for the Java interpreter are freely available for noncommercial use. Some restrictions exist for incorporating Java into commercial products. Sun Microsystems licenses Java to hardware and software companies that are developing products to run the Java virtual machine and execute Java code. Developers, however, can write Java code without a license. A complete Java Development Kit (JDK), including a Java compiler, can be downloaded for free [Sun 97d]. Although a J2EE reference implementation will be provided by Sun, this implementation is not expected to be usable in industrial deployments. Several vendors are providing J2EE solutions ranging from free open source distributions to industrial strength distributions with per developer fees and per server fees. Yourdon discusses the potential impact of Java on the cost of software applications in the future- purchased software packages could be replaced with transaction-oriented rental of Java applets attached to Web pages [Yourdon 96].

Alternatives

From a programming-language point of view, alternatives to Java include C/C++, Perl, and Tcl/Tk. Scripting languages often used in Web browsers, such as JavaScript and Visual Basic Scripting Edition (VB Script), can also be used to perform some of the tasks that Java can do, but not all of them. Perhaps the biggest challenge to client side Java's success is Microsoft's ActiveX technology. ActiveX is built on top of COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities). Microsoft provides tools for developers to create "ActiveX controls" that can serve a similar purpose to Java applets. The primary difference is that ActiveX is a proprietary technology that only runs on the Windows platform at present. It also provides a different security model based on its "Authenticode" certificate technology, security zones, and encrypted signatures. The ActiveX model itself is not secure in the way Java is; ActiveX controls have unlimited access to the user's machine when they are executing. This gives them more power to perform operations, but also makes them potentially more dangerous to the user's computing environment. The alternatives for the server side Java computing are CORBA and MTS. As CORBA and Java are basically complimentary technologies, only MTS can be considered as a J2EE's competitor. MTS is a product that provides to specifically designed COM objects with enterprise services as transactions and security. MTS and COM will converge into a single technology called COM+ that will be released with Windows 2000.

Complementary Technologies

The entire distributed object technology area (CORBA, COM/DCOM, ActiveX, etc.) offers technologies that can inter-operate with Java. There are standards available that let Java objects talk to CORBA objects, thus extending the capabilities of both technologies. Of particular relevance is the RMI-IIOP mapping that enable interoperability between RMI objects and IIOP objects. Also relevant is the CORBA 3 Component Model [OMG 99], this model is strongly based in EJB and has EJB interoperability as one of its main goals. Java provides solid foundations to component-based development (see Component-Based Software Development/COTS Integration). The additions of Remote Method Invocation (RMI), the JavaBeans component architecture [JavaSoft 97] and EJB component framework to Java facilitate the reuse of other people's software. Java components developed in this manner can have their interfaces examined, can communicate with one another over a network, and can be integrated with other components all without needing the source code. Java has evolved to become a serious option to implement three tier architectures (see Three Tier Software Architectures). Mobile and light clients can be implemented as Applets or JavaBeans, Enterprise JavaBeans are perfect for transactional business logic and Java Server Pages can be used to generate the html representation. The ability to deploy component-oriented enterprise multi-tiers systems, in a platform-neutral manner, can give fast moving enterprises a significant and measurable competitive edge.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Java
Application category	Distributed Computing (AP.2.1.2) Application Program Interfaces (AP.2.7) Programming Language (AP.1.4.2.1) Compiler (AP.1.4.2.3)
Quality measures category	Complexity (QM.3.2.1) Cost of Ownership (QM.5.1) Interoperability (QM.4.1) Maintainability (QM.3.1) Portability (QM.4.2) Reliability (QM.2.1.2) Scalability (QM.4.3) Trustworthiness (QM.2.1.4) Usability (QM.2.3)
Computing reviews category	Programming Languages (D.3) Distributed Systems (C.2.4)

References and Information Sources

[Clark 97] Clark, Don. "Sun Microsystems Pours Some New Java." *Wall Street Journal*, Page B-4. April 1, 1997.

[EJB-SIG 99] Special Interest Group - Enterprise JavaBeans [online] Available WWW, <URL: http://www.mgm-edv.de/ejbsig/ejbservers_tabled.html>

[Gamelan 97] Gamelan Web site [online]. Available WWW, <URL: http://www.gamelan.com> (1997).

[Gosling 96] Gosling, James & McGilton, Henry. *The Java Language Environment: A White Paper* [online]. Available WWW, <URL: http://java.sun.com/docs/white/langenv/> (1996).

[JARS 97] *Java Applet Rating Service* (JARS) [online]. Available WWW, <URL: http://www.jars.com> (1997).

[Halfhill 97] Halfhill, Tom R. "Today the Web, Tomorrow the World." *Byte 22*, 1 (January 1997): 68-80.

[Hamilton 96] Hamilton, Marc. "Java and the Shift to Net-Centric Computing." *Computer 29*, 8 (August 1996): 31-39.

[JavaSoft 97] *JavaBeans: The Only Component Architecture for Java* [online]. Available WWW, <URL: http://splash.javasoft.com/beans/> (1997).

[Levin 97] Levin, Rich and Patrizio, Andy. "Breaking Point" [online]. *Information Week*, June 23, 1997. Available WWW, <URL: http://techweb.cmp.com/iw/636/36iujav.htm> (1997).

[OMG 99] CORBA Component Model RFP [online]. Originally available WWW, <URL http://www.omg.org/techprocess/meetings/schedule/CORBA_Component_Model_RFP.html>

[Sun 96a] *CSX Gets on Track With Java* [online]. Available WWW, <URL: http://java.sun.com/features/1996/october/csx102996.html> (1996).

[Sun 96b] *Java Computing in the Enterprise. Strategic Overview: Java* [online]. Available WWW, <URL: http://www.sun.com/javacomputing> (1996).

[Sun 97a] *Frequently Asked Questions- Applet Security* [online]. Available WWW, <URL: http://java.sun.com/sfaq> (1997).

[Sun 97b] *Java in the Real World* [online]. Available WWW, <URL: http://java.sun.com/nav/used/> (1997).

[Sun 97c] *Customer Successes* [online]. Available WWW, <URL: http://www.sun.com/javastation/customersuccesses/> (1997).

[Sun 97d] *Java Development Kit 1.1.2* [online]. Available WWW, <URL: http://java.sun.com/products/jdk/1.1/> (1997).

[Sun 97e] Overview of Java [online]. Available WWW, <URL: http://java.sun.com/docs/Overviews/java/java-overview-1.html> (1997).

[Sun 99a] Java(tm) 2 Platform, Enterprise Edition [online]. Available WWW, <URL:http://java.sun.com/j2ee/>

[Sun 99b] Jini(tm) connection technology [online]. Available WWW, <URL: http://www.sun.com/jini/>

[Sun 99c] Java HotSpot(tm) Performance Engine [online]. Available WWW, <URL:http://java.sun.com/products/hotspot/>

[Sun 99d] JavaBeans Home Page [online]. Available WWW, <URL:http://java.sun.com/beans/index.html>

[Sun 99e] Enterprise JavaBeans Home Page [online]. Available WWW, <URL:http://java.sun.com/products/ejb/index.html>

[Sun 99f] J2EE Sun BluePrints(TM) [online]. Available WWW, <URL:http://java.sun.com/j2ee/blueprints/>

[van Hoff 96] van Hoff, A. *Hooked on Java*. Reading, MA: Addison-Wesley, 1996.

[Yourdon 96] Yourdon, Edward. "Java, the Web, and Software Development." *Computer 29*, 8 (August 1996): 25-30.

Current Author/Maintainer

Santiago Comella-Dorda, SEI
Scott Tilley, SEI

External Reviewers

Alan Brown, Texas Instruments
Hausi Müller, University of Victoria

Modifications

24 Feb 2000: Updated to cover new Java developments

More material added to cover Java 2, Jini, Java Beans, and Enterprise Java Beans. Many new references added.

30 June 1997: Substantially rewritten to reflect developments of the past 5-6 months.

More material added in Usage Considerations, Maturity, and Alternatives. Added figure to show applets, applications, Virtual Machines, and platform independence/neutrality. Many new references added.

10 Jan 1997 (original); author: Cory Vondrak, TRW, Redondo Beach, CA

Footnotes

[1] The network computer (NC) does not have an agreed-upon definition. Some NCs are new devices designed to run software written in Java, with gateways to existing programs and data. These are the official Network Computers (an Oracle trademark) and JavaStations (a Sun trademark). Other NCs are more like terminals in the classic sense: they don't execute programs at the desktop. Instead, applications run remotely on a server, and the client handles only the graphics locally. The generic term for these and the true NC alternatives to the personal computer (PC) is "thin client". They are referred to as "thin" because they are generally less complex and less expensive than a PC. However, recent

developments by Microsoft and others have muddied the waters a bit with the "NetPC," which is essentially a stripped-down and sealed PC that is meant to be centrally administered. [2] MIDI stands for "Musical Instrument Digital Interface". It is a hardware specification and protocol used to communicate note and effect information between synthesizers, computers, keyboards, controllers and other electronic music devices.

Java™ Copyright

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/java_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Mainframe Server Software Architectures

Software Technology Roadmap

Status

Complete

Note

We recommend Client/Server Software Architectures *as prerequisite reading for this technology description.*

Purpose and Origin

Since 1994 mainframes have been combined with distributed architectures to provide massive storage and to improve system security, *flexibility*, *scalability*, and *reusability* in the client/server design. In a mainframe server software architecture, mainframes are integrated as servers and data warehouses in a client/server environment. Additionally, mainframes still excel at simple transaction-oriented data processing to automate repetitive business tasks such as accounts receivable, accounts payable, general ledger, credit account management, and payroll. Siwolp and Edelstein provide details on mainframe server software architectures see [Siwolp 95, Edelstein 94].

Technical Detail

While client/server systems are suited for rapid application deployment and distributed processing, mainframes are efficient at online transactional processing, mass storage, centralized software distribution, and data warehousing [Data 96]. Data warehousing is information (usually in summary form) extracted from an operational database by data mining (drilling down into the information through a series of related queries). The purpose of data warehousing and data mining is to provide executive decision makers with data analysis information (such as trends and correlated results) to make and improve business decisions.

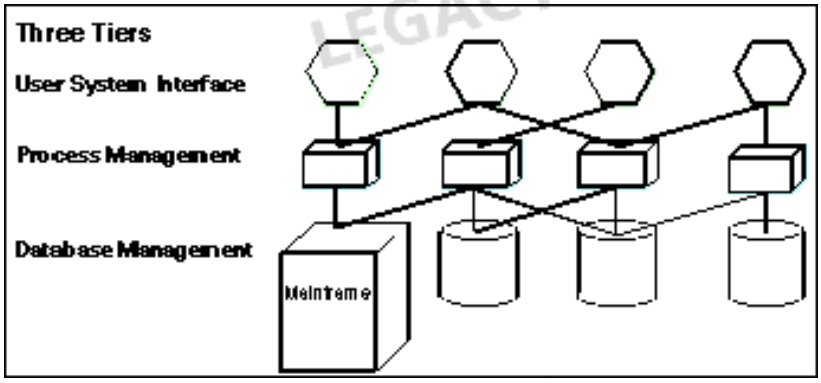


Figure 20: Mainframe in a Three Tier Client/Server Architecture

Figure 20 shows a mainframe in a three tier client/server architecture. The combination of mainframe horsepower as a server in a client/server distributed architecture results in a very effective and efficient system. Mainframe vendors are now providing standard communications and programming interfaces that make it easy to integrate mainframes as servers in a client/server architecture. Using mainframes as servers in a client/server distributed architecture provides a more modular system design, and provides the benefits of the client/server technology.

Using mainframes as servers in a client/server architecture also enables the distribution of workload between major data centers and provides disaster protection and recovery by backing up large volumes of data at disparate locations. The current model favors "thin" clients (contains primarily user interface services) with very powerful servers that do most of the extensive application and data processing, such as in a two tier architecture. In a three tier client/server architecture, process management (business rule execution) could be off-loaded to another server.

Usage Considerations

Mainframes are preferred for big batch jobs and storing massive amounts of vital data. They are mainly used in the banking industry, public utility systems, and for information services. Mainframes also have tools for monitoring performance of the entire system, including networks and applications not available today on UNIX servers [Siwolp 95].

New mainframes are providing parallel systems (unlike older bipolar machines) and use complementary metal-oxide semiconductor (CMOS) microprocessors, rather than emitter-coupler logic (ECL) processors. Because CMOS processors are packed more densely than ECL microprocessors, mainframes can be built much smaller and are not so power-hungry. They can also be cooled with air instead of water [Siwolp 95].

While it appeared in the early 1990s that mainframes were being replaced by client/server architectures, they are making a comeback. Some mainframe vendors have seen as much as a 66% jump in mainframe shipments in 1995 due to the new mainframe server software architecture [Siwolp 95].

Given the cost of a mainframe compared to other servers, UNIX workstations and personal computers (PCs), it is not likely that mainframes would replace all other servers in a distributed two or three tier client/server architecture.

Maturity

Mainframe technology has been well known for decades. The new improved models have been fielded since 1994. The new mainframe server software architecture provides the distributed client/server design with massive storage and improved security capability. New technologies of data warehousing and data mining data allow extraction of information from the operational mainframe server's massive storage to provide businesses with timely data to improve overall business effectiveness. For example, stores such as Wal-Mart found that by placing certain products in close proximity within the store, both products sold at higher rates than when not collocated.[1]

Costs and Limitations

By themselves, mainframes are not appropriate mechanisms to support graphical user interfaces. Nor can they easily accommodate increases in the number of user applications or rapidly changing user needs [Edelstein 94].

Alternatives

Using a client/server architecture without a mainframe server is a possible alternative. When requirements for high volume (greater than 50 gigabit), batch type processing, security, and mass storage are minimal, three tier or two tier architectures without a mainframe server may be viable alternatives. Other possible alternatives to using mainframes in a client/server distributed environment are using parallel processing software architecture or using a database machine.

Complementary Technologies

A complementary technology to mainframe server software architectures is open systems . This is because movement in the industry towards interoperable heterogeneous software programs and operating systems will continue to increase reuse of mainframe technology and provide potentially new applications for mainframe capabilities.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Mainframe Server Software Architectures
Application category	Client/Server (AP.2.1.2.1)
Quality measures category	Maintainability (QM.3.1) Scalability (QM.4.3) Reusability (QM.4.4)
Computing reviews category	Distributed Systems (C.2.4)

References and Information Sources

[Data 96]	*Data Warehousing* [online]. Available WWW <URL: http://www-db.stanford.edu/warehousing/publications.html> and <URL: http://www-db.stanford.edu/warehousing/warehouse.html> (1996).
[Edelstein 94]	Edelstein, Herb. "Unraveling Client/Server Architecture." *DBMS 7*, 5 (May 1994): 34(7).
[Siwolp 95]	Siwolp, Sana. "Not Your Father's Mainframe." *Information Week* 546 (Sept 25, 1995): 53-58.

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Frank Rogers, GTE

Modifications

10 Jan 97 (original)

Footnotes

[1] Source: Stodder, David. Open Session Very Large Data Base (VLDB) Summit, New Orleans, LA 23-26 April, 1995.

sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/mssa_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Rule-Based Intrusion Detection

Software Technology Roadmap

Status

Advanced

Note

We recommend Intrusion Detection as prerequisite reading for this technology description.

Purpose and Origin

Due to the voluminous, detailed nature of system audit data - some of which may have little if any meaning to a human reviewer - and the difficulty of discriminating between normal and intrusive behavior, one approach taken by developers of intrusion detection systems is to use expert systems technology to analyze automatically audit trail data for intrusion attempts [Lunt 93]. These security systems, known as rule-based intrusion detection (RBID) systems, can be used to analyze system audit trails for pending or completed computer security violations. This emerging technology seeks to increase the *availability* of computer systems by automating the detection and elimination of intrusions.

Technical Detail

Rule-based intrusion detection (RBID) is predicated on the assumption that intrusion attempts can be characterized by sequences of user activities that lead to compromised system states. RBID systems are characterized by their expert system properties that fire rules[1] when audit records or system status information begin to indicate illegal activity [Ilgun 93]. These predefined rules typically look for high-level state change patterns observed in the audit data compared to predefined penetration state change scenarios. If an RBID expert system infers that a penetration is in process or has occurred, it will alert the computer system security officers and provide them with both a justification for the alert and the user identification of the suspected intruder.

There are two major approaches to rule-based intrusion detection:

1. *State-based*. In this approach, the rule base is codified using the terminology found in the audit trails. Intrusion attempts are defined as

sequences of system state- as defined by audit trail information- leading from an initial, limited access state to a final compromised state [Ilgun 93].

2. *Model-based*. In this approach, known intrusion attempts are modeled as sequences of user behavior; these behaviors may then be modeled, for example, as events in an audit trail. Note, however, that the intrusion detection system itself is responsible for determining how an identified user behavior may manifest itself in an audit trail. This approach has many benefits, including the following:
    - More data can be processed, because the technology allows you to narrow the focus of the data selectively.
    - More intuitive explanations of intrusion attempts are possible.
    - The system can predict the intruder's next action.

Usage Considerations

RBID rule bases are affected by system hardware or software changes and require updates by system experts as the system is enhanced or maintained. The protection afforded by RBID systems would be most useful in an environment where physical protection of the computer system is not always possible (e.g., a battlefield situation), yet the data is of high value and requires stringent protection.

Maturity

Although RBID systems are in the research and early prototype stage, articles describing RBID systems date to at least the 1986 description of the Discovery system [Tener 86]. In 1987, Denning described an early, abstract model of a rule-based intrusion detection system (IDS) [Denning 87]; in 1989, Vaccarro and Liepins described the Wisdom and Sense system [Vaccarro 89]. More recent systems include USTAT [Ilgun 93] and the Intrusion Detection Expert System (IDES) [Lunt 93]; IDES combines statistical-based (see Statistical-Based Intrusion Detection) and model-based intrusion detection approaches to achieve a level of intrusion detection not feasible with either approach alone. Mukherjee describes several other recent RBID systems [Mukherjee 94]. Feasibility for an operational system has not yet been demonstrated.

Costs and Limitations

The use of RBID systems requires the following:

- personnel knowledgeable in rule-based systems, especially with respect to rule representation
- personnel who know how various activities may be represented in audit trails
- personnel experienced in intrusion detection and who have in-depth knowledge of the audit collection mechanism [Ilgun 93]

In addition to the costs associated with maintaining intrusion detection knowledge bases, there are several risks and limitations associated with this

technology:

- Only known vulnerabilities and attacks are codified in the knowledge base. The knowledge base of rules is thus always playing "catch-up" with the intruders [Lunt 93].
- The representation of intrusion scenarios- especially with respect to state-based approaches- is not intuitive.

For these reasons, RBIDs cannot detect all intrusion attempts.

Like all intrusion detection systems, RBIDs will negatively affect system *performance* due to their collecting and processing of audit trail information. For example, early prototyping of a real-time RBID system on a UNIX workstation showed the algorithm was using up to 50% of the available processor throughput to process and analyze the audit trail [Ilgun 93].

Dependencies

Expert systems are an enabler for this technology.

Alternatives

Other automated approaches to intrusion detection include statistical-based approaches (see Statistical-Based Intrusion Detection) and approaches based on genetic algorithms. Manual examination of recorded audit data and online monitoring of access activity by knowledgeable system security personnel are the only other known alternatives.

Complementary Technologies

RBID systems can be used in conjunction with Statistical-Based Intrusion Detection systems to catch a wider variety of intrusion attempts, and authentication systems can be used to verify user identity.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Rule-Based Intrusion Detection
Application category	System Security (AP.2.4.3)
Quality measures category	Security (QM.2.1.5)

Computing reviews category	Operating Systems Security and Protection (D.4.6) Computer-Communication Networks Security and Protection (C.2.0) Security and Protection (K.6.5)

References and Information Sources

[Bell 76] Bell, D. E. & LaPadula, L. J. *Secure Computer System: Unified Exposition and Multics Interpretation* Rev. 1 (MTR-2997). Bedford, MA: MITRE Corporation, 1976.

[Denning 87] Denning, Dorothy E., et al. "Views for Multilevel Database Security." *IEEE Transactions on Software Engineering SE-13,* 2 (February 1987): 129-140.

[CSC 83] Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria*. Fort George G. Meade, MD: DoD Computer Security Center, 1983.

[Ilgun 93] Ilgun, Koral. "USTAT: A Real-time Intrusion Detection System for UNIX," 16-28. *Proceedings of the 1993 Computer Society Symposium on Research in Security and Privacy*. Oakland, California, May 24-26, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

[Kemmerer 94] Kemmerer, Richard A. "Computer Security," 1153-1164. *Encyclopedia of Software Engineering.* New York, NY: John Wiley and Sons, 1994.

[Lunt 93] Lunt, Teresa F. "A Survey of Intrusion Detection Techniques." *Computers and Security 12,* 4 (June 1993): 405-418.

[Mukherjee 94] Mukherjee, Biswanath, L.; Heberlein, Todd; & Levitt, Karl N. "Network Intrusion Detection." *IEEE Network 8,* 3 (May/June 1994): 26-41.

[Sundaram 96] Sundaram, Aurobindo. *An Introduction to Intrusion Detection* [online]. Available WWW <URL: http://www.acm.org/crossroads/xrds2-4/xrds2-4.html> (1996).

[Tener 86] Tener, W. T. "Discovery: An Expert System in the Commercial Data Security Environment." *Computer Security Journal 6,* 1 (Summer 1990): 45.

[Vaccarro 89] Vaccarro, H. S. & Liepins, G. E. "Detection of Anomalous Computer Session Activity," 208-209. *Proceedings of the IEEE Symposium on Research in Security and Privacy*. Oakland, California, May 1-3, 1989. Washington, DC: IEEE Computer Society Press, 1989.

[Ware 79] Ware, W. H. *Security Controls for Computer Systems: Report of Defense Science Board, Task Force on Computer Security*. Santa Monica, CA: The Rand Corporation, 1979.

Current Author/Maintainer

Mark Gerken, Air Force Rome Laboratory

Modifications

10 Jan 97 (original)

Footnotes

[1] In an expert system, knowledge about a problem domain is represented by a set of rules. These rules consist of two parts:

1. The antecedent, which defines when the rule should be applied. An expert system will use pattern matching techniques to determine when the observed data matches or satisfies the antecedent of a rule.
2. The consequent, which defines the action(s) that should be taken if its antecedent is satisfied.

A rule is said to be "fired" when the action(s) defined in its consequent are executed. For RBID systems, rule antecedents will typically be defined in terms of audit trail data, while rule consequents may be used to increase or decrease the level of monitoring of various entities, or they may be used to notify system administration personnel about significant changes in system state.

URL: http://www.sei.cmu.edu/str/descriptions/rbid_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes

- Lists of related topics

Simple Network Management Protocol

Software Technology Roadmap

Status

Advanced

Note

We recommend Network Management -- An Overview *as prerequisite reading for this technology description.*

Purpose and Origin

Simple Network Management Protocol (SNMP) is a network management specification developed by the Internet Engineering Task Force (IETF),[1] a subsidiary group of the Internet Activities Board (IAB),[2] in the mid 1980s to provide standard, simplified, and extensible management of LAN-based internetworking products such as bridges, routers, and wiring concentrators [IETF 96, Henderson 95]. SNMP was designed to reduce the complexity of network management and minimize the amount of resources required to support it. SNMP provides for centralized, robust, interoperable network management, along with the flexibility to allow for the management of vendor-specific information.

Technical Detail

SNMP is a communication specification that defines how management information is exchanged between network management applications and management agents. There are several versions of SNMP, two of the most common are SNMPv1 [SNMPv1 Specs] and SNMPv2 [SNMPv2 Specs]. SNMPv2 and some of the less common versions will be discussed later in this text.

The architecture of SNMPv1 is shown in Figure 33, which is a more detailed version of the managed device and network management application shown in Figure 27 of Network Management-An Overview. SNMPv1 is a simple message based request/response application-layer protocol which typically uses the User Datagram Protocol (UDP) [RFC 96] for data delivery. The SNMPv1 network management architecture contains:

- Network Management Station (NMS) - Workstation that hosts the network management application.

- SNMPv1 network management application - Polls management agents for information and provides control information to agents.
- Management Information Base (MIB) - Defines the information that can be collected and controlled by the management application.
- SNMPv1 management agent(s) - Provides information contained in the MIB to management applications and may accept control information.

A MIB is basically a database of managed objects[3] that resides on the agent. Managed objects are a characteristic of a managed device that can be monitored, modified or controlled, such as a threshold, network address or counter. The management application or user can define the relationship between the SNMPv1 manager and the management agent.

Attributes of managed objects may be monitored or set by the network management application using the following operations:

- GET_NEXT_REQUEST - Requests the next object instance from a table or list from an agent
- GET_RESPONSE - Returned answer to *get_next_request*, *get_request*, or *set_request*
- GET_REQUEST - Requests the value of an object instance from the agent
- SET_REQUEST - Set the value of an object instance within an agent
- TRAP - Send trap (event) asynchronously to *network management application.* Agents can send a trap when a condition has occurred, such as change in state of a device, device failure or agent initialization/restart.

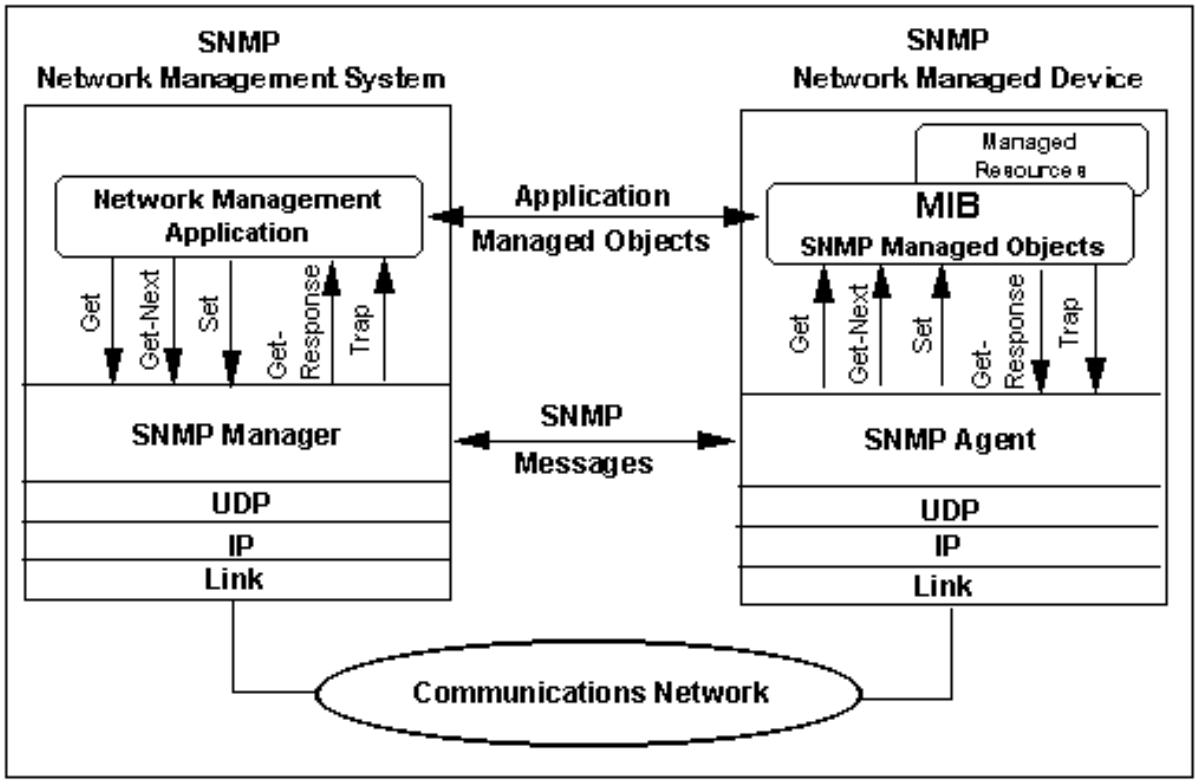


Figure 33: The SNMPv1 Architecture [Lake 96]

By specifying the protocol to be used between the network management application and

management agent, SNMP allows products (software and managed devices) from different vendors (and their associated management agents) to be managed by the same SNMP network management application. A "proxy function" is also specified by SNMP to enable communication with non-SNMP devices to accommodate legacy equipment.

The main attributes of SNMP are as follows [Moorhead 95]:

- It is simple to implement, making it easy for a vendor to accommodate it into its device.
- It does not require large computational or memory resources from the devices that do accommodate it.

Network management, as defined by SNMP, is based on polling and asynchronous events. The SNMP manager polls for information gathered by each of the agents. Each agent has the responsibility of collecting information (e.g., performance statistics) pertaining to the device it resides within and storing that information in the agent's own management information base (MIB). This information is sent to the SNMP manager in response to the manager's polling.

SNMP events (alerts) are driven by trap messages generated as a result of certain device parameters. These parameters can be either generic or vendor device specific. Enterprise-specific trap messages are vendor proprietary and generally provide more device-specific detail.

The SNMPv2 [SNMPv2 Specs] (SNMP Version 2) specification included the following new capabilities:

- manager to manager communication to support the coexistence of multiple/ distributed managers and mid-level managers, increasing the flexibility and scalability of the network being managed
- enhanced security (known as "Secure SNMP") by specifying three layers of security
  - encryption: Used to keep content of messages private. Encryption is based on the Data Encryption Standard (DES) [DES 93] defined by the National Institute of Standards and Technology (NIST) and the American National Standards Institute (ANSI)[4].
  - authentication: Proof of the identity of the sender of a message.
  - authorization: Provides access restrictions thru access control lists.
- improved efficiency and performance through the addition of bulk transfers of data. This means that in some cases, using SNMPv2 instead of SNMPv1, network management can be provided over low-bandwidth, wide-area links.
- support for additional network protocols besides UDP/IP, for example, OSI, NetWare IPX/SPX and Appletalk [Broadhead 95]

Usage Considerations

Problem isolation. Neither version of SNMP does an effective job at helping network managers isolate problem devices in large, complex networks. It sometimes becomes difficult for an SNMP manager to determine which network events/alarms are significant-- all are treated equally.

Focus. SNMPv1 provides information only on individual devices, not on how the devices work as a system.

Incompatibilities. SNMPv1 and SNMPv2 are incompatible with each other and can not interact, however, some SNMP network management applications packages support both specifications.

Performance. The performance impact on the network being managed should be considered when using the polling scheme that SNMP uses for collecting information from distributed agents. A higher frequency of polling, which may be required to manage a network effectively, will increase the overhead on a network, possibly resulting in a need for additional networking or processor resources. The frequency of polling can be controlled by the SNMP manager, but can be dependent on what kind of messages (generic or enterprise-specific) a device vendor supports. Many vendors offer generic trap messages on their devices rather than enterprise-specific messages, because it is easier and takes less time for the vendor to implement. Devices that provide only generic trap information must be polled frequently to obtain the granularity of information to manage the device effectively.

Maturity

SNMPv1 has been incorporated into many products and management platforms. It has been deployed by virtually all internetworking vendors. It has been widely adopted for the enterprise (business organization) networks and may be the manager of choice for the internetworking arena in the future because it is well-suited for managing TCP/IP networks. Limitations are discussed below in Costs and Limitations.

SNMPv2 has many unresolved issues and was supported by few vendors as of January 1998. The members of IETF subcommittee can not agree upon several parts of the SNMPv2 specification (primarily the security and administrative needs of the protocol); as a result only certain parts of SNMPv2 specification have reached draft standard status within the IETF [SNMP FAQ 98]. There has been several attempts to achieve acceptance of SNMPv2 through the release of experimental modified versions commonly known as SNMPv2*, SNMPv2c, SNMPv2u, SNMPv1+ and SNMP1.5 that do not contain the contentious parts.

SNMPv3 is the latest proposed version for the next generation of SNMP functionality. It is based upon the protocol operations, data types, and proxy support from SNMPv2 with user-based seucurity from SNMPv2u and SNMPv2*. It may take years before a new version is accepted.

Costs and Limitations

The attractiveness of SNMP is its simplicity and relative ease of implementation. With this comes a price: e.g., the more fine grained information that is need or required, such as the variance in interarrival time (jitter) of packets sent to a particular local address, the less likely it is that it will be available.

SNMPv1 uses the underlying User Datagram Protocol (UDP) for data delivery, which

does not ensure reliability of data transfer. The loss of data may be a limitation to a network manager, depending on the criticality of the information being gathered and the frequency at which the polling is being performed.

SNMP is best suited for network monitoring and capacity planning. SNMP does not provide even the basic troubleshooting information that can be obtained from simple network troubleshooting tools [Wellens 96]. SNMP agents do not analyze information, they just collect information and provide it to the network management application.

SNMPv1 has minimal security capability. Because SNMPv1 lacks the control of unauthorized access to critical network devices and systems, it may be necessary to restrict the use of SNMP management to non-critical networks. Lack of authentication in SNMPv1 has led many vendors to not include certain commands, thus reducing extensibility and consistency across managed devices. SNMPv2 addresses these security problems but is difficult and expensive to set up and administer (e.g., each MIB must be locally set up).

Vendors often include SNMP agents with their software and public domain agents are available. Management applications are available from a variety of vendors as well as the public domain, however they can differ greatly in terms of functionality, plots and visual displays.

SNMP out-of-the-box can not be used to track information contained in application/user level protocols (e.g., radar track message, http, mail). However these might be accomplished through the use of a extensible (customized) SNMP agent that has user defined MIB.[5] It is important to note that a specialized or extensible network manager may be required for use with the customized agents.

There are also concerns about the use of SNMP in the real-time domain where bounded response, deadlines, and priorities are required.

SNMPv2 is intended to be able to coexist with existing SNMPv2, but in order to use SNMPv2 as the SNMP manager or to migrate from SNMPv1 to SNMPv2, all SNMPv1 compliant agents must be entirely replaced with SNMPv2 compliant agents-gateways or bilingual managers and proxy agents were not available to support the gradual migration as of early-1995. Since SNMPv1 and SNMPv2 are incompatible with each other and SNMPv2 is not stable, it is important when procuring a managed device to determine which network management protocol(s) is supported.

Alternatives

Common Management Information Protocol (CMIP) may be a better alternative for large, complex networks or security-critical networks.

CMIP is similar to SNMP and was developed to address SNMP's shortcomings. However, CMIP takes significantly more system resources than SNMP, is difficult to program, and is designed to run on the ISO protocol stack [X.700 96]. (However, the technology standard used today in most systems is TCP/IP.)

The biggest feature in CMIP is that an agent can perform tasks or trigger events based

upon the value of a variable or a specific condition. For example, when a computer can not reach its network fileserver for a predetermined number of times, an event can be generated to notify the appropriate personnel [Vallillee 96]. With SNMP, this task would have to be performed by a user, because an SNMP agent does not analyze information.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Simple Network Management Protocol
Application category	Protocols (AP.2.2.3) Network Management (AP.2.2.2)
Quality measures category	Maintainability (QM.3.1) Simplicity (QM.3.2.2) Complexity (QM.3.2.1) Efficiency/ Resource Utilization (QM.2.2) Scalability (QM.4.3) Security (QM.2.1.5)
Computing reviews category	Network Operations (C.2.3) Distributed Systems (C.2.4)

References and Information Sources

[Broadhead 95] Broadhead, Steve. "SNMP Too Simple for Security?" *Secure Computing* (April 1995): 24-29.

[DES 93] Federal Information Processing Standards Publication 46-2 DATA ENCRYPTION STANDARD, 1993 [online]. Available WWW <URL: http://csrc.ncsl.nist.gov/fips/fips46-2.txt> (1996).

[Feit 94] Feit, Sidnie. *A Guide to Network Management*. New York, NY: McGraw Hill, 1994.

[Henderson 95] Henderson and Erwin. "SNMP Version 2: Not So Simple." *Business Communications Review 25,* 5 (May 1995): 44-48.

[Herman 94] Herman, James. "Network Computing Inches Forward." *Business Communications Review 24,* 5 (May 1994): 45-50.

[IETF 96] Internet Engineering Task Force home page [online]. Available WWW <URL: http://www.ietf.cnri.reston.va.us/> (1996).

[Kapoor 94] Kapoor, K. "SNMP Platforms: What's Real, What Isn't." *Data Communications International 23,* 12 (September 1994): 115-18.

[Lake 96] Lake, Craig. *Simple Network Management Protocol (SNMP)* [online]. Available WWW <URL: http://www.sei.cmu.edu/str/docs/SNMP.html> (1996).

[MIB 96] *Development of an MIB for http* [online]. Available WWW <URL: http://http-mib.onramp.net/bof/> (1996).

[Moorhead 95] Moorhead, R.J. & Amirthalingam, K. "SNMP- An Overview of its Merits and Demerits," 180-3. *Proceedings of the Twenty-Seventh Southeastern Symposium on System Theory*. Starkvill, MS, March 12-14, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Phifer 94] Phifer, L.A. "Tearing Down the Wall: Integrating ISO and Internet Management." *Journal of Network and Systems Management 2*, 3 (September 1994): pp. 317-22.

[RFC 96] Postel T. *User Datagram Protocol* (RFC 768) [online]. Available WWW <URL: http://ds.internic.net/rfc/rfc768.txt> (1996).

[Rose 94] Rose, Marshall T. *The Simple Book: An Introduction to Internet Management.* Englewood Cliffs, NJ: Prentice-Hall, 1994.

[SNMP 98] *Simple Network Management Protocol* [online]. Available WWW <URL: http://www.snmp.com> and <URL: http://www.snmp.com/snmppages.html> (1998).

[SNMP FAQ 98] Simple Network Management Protocol FAQ [online]. Available WWW <URL: http://www.snmp.com/FAQs/snmp-faq-part1.txt> and <URL: http://www.snmp.com/FAQs/snmp-faq-part2.txt> (1998).

[SNMPv1 Specs] The following RFC's identify the major components of SNMPv1 online]. Available WWW <URL: http://www.cis.ohio-state.edu/htbin/rfc/rfcXXXX.html> (1996).

Historical

RFC 1156 - Management Information Base Network Management of TCP/IP based internets
RFC 1161 - SNMP over OSI

Informational

RFC 1215 - A Convention for Defining Traps for use with the SNMP
RFC 1270 - SNMP Communication Services
RFC 1303 - A Convention for Describing SNMP-based Agents
RFC 1470 - A Network Management Tool Catalog

Standard and Draft

RFC 1089 - SNMP over Ethernet
RFC 1140 - IAB Official Protocol Standards
RFC 1155 - Structure and Identification of Management Information for TCP/IP based internets.

RFC 1157 - A Simple Network Management Protocol
RFC 1158 - Management Information Base Network Management of TCP/IP based internets: MIB-II
RFC 1187 - Bulk Table Retrieval with the SNMP
RFC 1212 - Concise MIB Definitions
RFC 1213 - Management Information Base for Network Management of TCP/IP-based internets: MIB-II
RFC 1224 - Techniques for Managing Asynchronously-Generated Alerts
RFC 1418 - SNMP over OSI
RFC 1419 - SNMP over AppleTalk
RFC 1420 - SNMP over IPX

[SNMPv2 Specs] The following RFC's identify the major components of SNMPv2 online]. Available WWW <URL: http://www.cis.ohio-state.edu/htbin/rfc/rfcXXXX.html> (1996).

Historical

RFC 1441 - Introduction to SNMP v2
RFC 1442 - SMI For SNMP v2
RFC 1443 - Textual Conventions for SNMP v2
RFC 1444 - Conformance Statements for SNMP v2
RFC 1445 - Administrative Model for SNMP v2
RFC 1446 - Security Protocols for SNMP v2
RFC 1447 - Party MIB for SNMP v2
RFC 1448 - Protocol Operations for SNMP v2
RFC 1449 - Transport Mappings for SNMP v2
RFC 1450 - MIB for SNMP v2
RFC 1451 - Manager to Manager MIB
RFC 1452 - Coexistence between SNMP v1 and SNMP v2

Draft

RFC 1902 - SMI for SNMPv2
RFC 1903 - Textual Conventions for SNMPv2
RFC 1904 - Conformance Statements for SNMPv2
RFC 1905 - Protocol Operations for SNMPv2
RFC 1906 - Transport Mappings for SNMPv2
RFC 1907 - MIB for SNMPv2
RFC 1908 - Coexistence between SNMPv1 and SNMPv2

Experimental

RFC 1901 - Introduction to Community-based SNMPv2
RFC 1909 - An Administrative Infrastructure for SNMPv2
RFC 1910 - User-based Security Model for SNMPv2

[Stallings 93] Stallings, William. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards.* Reading, MA: Addison-Wesley, 1993.

[Vallillee 96] Vallillee, Tyler. *SNMP & CMIP: An Introduction To Network Management* [online]. Available WWW <URL: http://www.inforamp.net/~kjvallil/t/snmp.html> (1996).

[Wellens 96] Wellens, Chris & Auerbach, Karl. "Towards Useful Management" [online]. *The Quarterly Newsletter of SNMP Technology, Comment, and Events(sm) 4,* 3 (July 1996). Available WWW <URL: http://www.iwl.com/Press/thefuture.html> (1996).

[X.700 96] *X.700 and Other Network Management Services* [online]. Available WWW <URL: http://ganges.cs.tcd.ie/4ba2/x700/index.html> (1996).

Current Author/Maintainer

Dan Plakosh, SEI

External Reviewers

Craig Meyers, SEI
Patrick Place, SEI

Modifications

16 Jan 98: Changes included

- Increased the consistency of terminology
- Minor change to the SNMPv1 architecture figure
- Updated status of SNMPv2 and added information about other SNMP versions
- Clarified some areas
- Updated references

19 Jun 97: Changes included

- Creating an overview technical description on network management, which includes overview material and figures applicable to all network management techniques
- Clarifying the discussion of SNMPv1 and SNMPv2
- Minor changes to the SNMPv1 architecture figure
- Increased the consistency of terminology
- added many new references

10 Jan 97 (original); author for this version: Cory Vondrak, TRW, Redondo Beach, CA

Footnotes

[1] The IETF is a large open community of network designers, operators, vendors, and researchers whose purpose is to coordinate the operation, management and evolution of the Internet, and to resolve short- and mid-range protocol and architectural issues. It is a major source of proposed protocol standards which are submitted to the Internet Engineering Steering Group for final approval. The IETF meets three times a year and extensive minutes of the plenary proceedings are issued.

[2] The IAB is a technical advisory group of the Internet Society. The IAB provides oversight of the architecture for the protocols and procedures used by the Internet, the process used to create Internet Standards and serves as an appeal board for complaints of improper execution of the standards process.

[3] Managed objects: a characteristic of a managed device that can be monitored, modified or controlled.

[4] This organization is responsible for approving U.S. standards in many areas, including computers and communications. Standards approved by this organization are often called ANSI standards (e.g., ANSI C is the version of the C language approved by ANSI).

[5] There is an MIB being developed for http [MIB 96], and the MIB for mail monitoring is now a proposed standard.

URL: http://www.sei.cmu.edu/str/descriptions/snmp_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Six Sigma

Software Technology Roadmap

Status

Complete

Note

Purpose and Origin

Six Sigma (6σ) is a business-driven, multi-faceted approach to process improvement, reduced costs, and increased profits. With a fundamental principle to improve customer satisfaction by reducing defects, its ultimate performance target is virtually defect-free processes and products (3.4 or fewer defective parts per million (ppm)). The Six Sigma methodology, consisting of the steps "Define - Measure - Analyze - Improve - Control," is the roadmap to achieving this goal. Within this improvement framework, it is the responsibility of the improvement team to identify the process, the definition of defect, and the corresponding measurements. This degree of flexibility enables the Six Sigma method, along with its toolkit, to easily integrate with existing models of software process implementation.

Six Sigma originated at Motorola in the early 1980s in response to a CEO-driven challenge to achieve tenfold reduction in product-failure levels in five years. Meeting this challenge required swift and accurate root-cause analysis and correction. In the mid-1990s, Motorola divulged the details of their quality improvement framework, which has since been adopted by several large manufacturing companies. [Harry 00, Arnold 99, Harrold 99]

Technical Detail

The primary goal of Six Sigma is to improve customer satisfaction, and thereby profitability, by reducing and eliminating defects. Defects may be related to any aspect of customer satisfaction: high product quality, schedule adherence, cost minimization. Underlying this goal is the Taguchi Loss Function [Pyzdek 01], which shows that increasing defects leads to increased customer dissatisfaction and financial loss. Common Six Sigma metrics include defect rate (parts per million or ppm), sigma level, process capability indices, defects per unit, and yield. Many Six Sigma metrics can be mathematically related to the others.

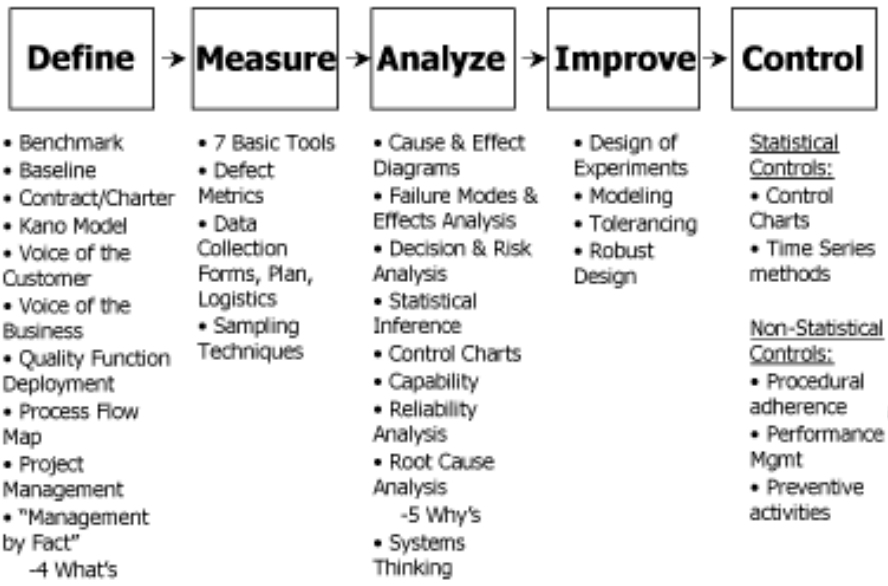
The Six Sigma drive for defect reduction, process improvement and customer

satisfaction is based on the "statistical thinking" paradigm [ASQ 00], [ASA 01]:

- Everything is a process
- All processes have inherent variability
- Data is used to understand the variability and drive process improvement decisions

As the roadmap for actualizing the statistical thinking paradigm, the key steps in the Six Sigma improvement framework are Define - Measure - Analyze - Improve - Control (see Figure 1). Six Sigma distinguishes itself from other quality improvement programs immediately in the "Define" step. When a specific Six Sigma project is launched, the customer satisfaction goals have likely been established and decomposed into subgoals such as cycle time reduction, cost reduction, or defect reduction. (This may have been done using the Six Sigma methodology at a business/organizational level.) The Define stage for the specific project calls for baselining and benchmarking the process to be improved, decomposing the process into manageable sub-processes, further specifying goals/sub-goals and establishing infrastructure to accomplish the goals. It also includes an assessment of the cultural/organizational change that might be needed for success.

Once an effort or project is defined, the team methodically proceeds through Measurement, Analysis, Improvement, and Control steps. A Six Sigma improvement team is responsible for identifying relevant metrics based on engineering principles and models. With data/information in hand, the team then proceeds to evaluate the data/information for trends, patterns, causal relationships and "root cause," etc. If needed, special experiments and modeling may be done to confirm hypothesized relationships or to understand the extent of leverage of factors; but many improvement projects may be accomplished with the most basic statistical and non-statistical tools. It is often necessary to iterate through the Measure-Analyze-Improve steps. When the target level of performance is achieved, control measures are then established to sustain performance. A partial list of specific tools to support each of these steps is shown in Figure 1.



Note:
Many tools can be effectively used in multiple steps of the framework. Tools that are not particularly relevant to software applications have not been included in this list.

Figure 1: Six Sigma Improvement Framework and Toolkit

An important consideration throughout all the Six Sigma steps is to distinguish which process substeps significantly contribute to the end result. The defect rate of the process, service or final product is likely more sensitive to some factors than others. The analysis phase of Six Sigma can help identify the extent of improvement needed in each substep in order to achieve the target in the final product. It is important to remain mindful that six sigma performance (in terms of the ppm metric) is not required for every aspect of every process, product and service. It is the goal only where it quantitatively drives (i.e, is a significant "control knob" for) the end result of customer satisfaction and profitability.

The current average industry runs at four sigma, which corresponds to 6210 defects per million opportunities. Depending on the exact definition of "defect" in payroll processing, for example, this sigma level could be interpreted as 6 out of every 1000 paychecks having an error. As "four sigma" is the average current performance, there are industry sectors running above and below this value. Internal Revenue Service (IRS) phone-in tax advice, for instance, runs at roughly two sigma, which corresponds to 308,537 errors per million opportunities. Again, depending on the exact definition of defect, this could be interpreted as 30 out of 100 phone calls resulting in erroneous tax advice. ("Two Sigma" performance is where many noncompetitive companies run.) On the other extreme, domestic (U. S.) airline flight fatality rates run at better than six sigma, which could be interpreted as fewer than 3.4 fatalities per million passengers - that is, fewer than 0.00034 fatalities per 100 passengers [Harry 00], [Bylinsky 98], [Harrold 99].

As just noted, flight fatality rates are "better than six sigma," where "six sigma" denotes the actual performance level rather than a reference to the overall combination of philosophy, metric, and improvement framework. Because customer demands will likely drive different performance expectations, it is useful to understand the mathematical origin of the measure and the term "six-sigma process." Conceptually, the sigma level of a process or product is where its customer-driven specifications intersect with its distribution. A centered six-sigma process has a normal distribution with mean=target and specifications placed 6 standard deviations to either side of the mean. At this point, the portions of the distribution that are beyond the specifications contain 0.002 ppm of the data (0.001 on each side). Practice has shown that most manufacturing processes experience a shift (due to drift over time) of 1.5 standard deviations so that the mean no longer equals target. When this happens in a six-sigma process, a larger portion of the distribution now extends beyond the specification limits: 3.4 ppm.

Figure 2 depicts a 1.5σ-shifted distribution with "6σ" annotations. In manufacturing, this shift results from things such as mechanical wear over time and causes the six-sigma defect rate to become 3.4 ppm. The magnitude of the shift may vary, but empirical evidence indicates that 1.5 is about average. Does

this shift exist in the software process? While it will take time to build sufficient data repositories to verify this assumption within the software and systems sector, it is reasonable to presume that there are factors that would contribute to such a shift. Possible examples are declining procedural adherence over time, learning curve, and constantly changing tools and technologies (hardware and software).

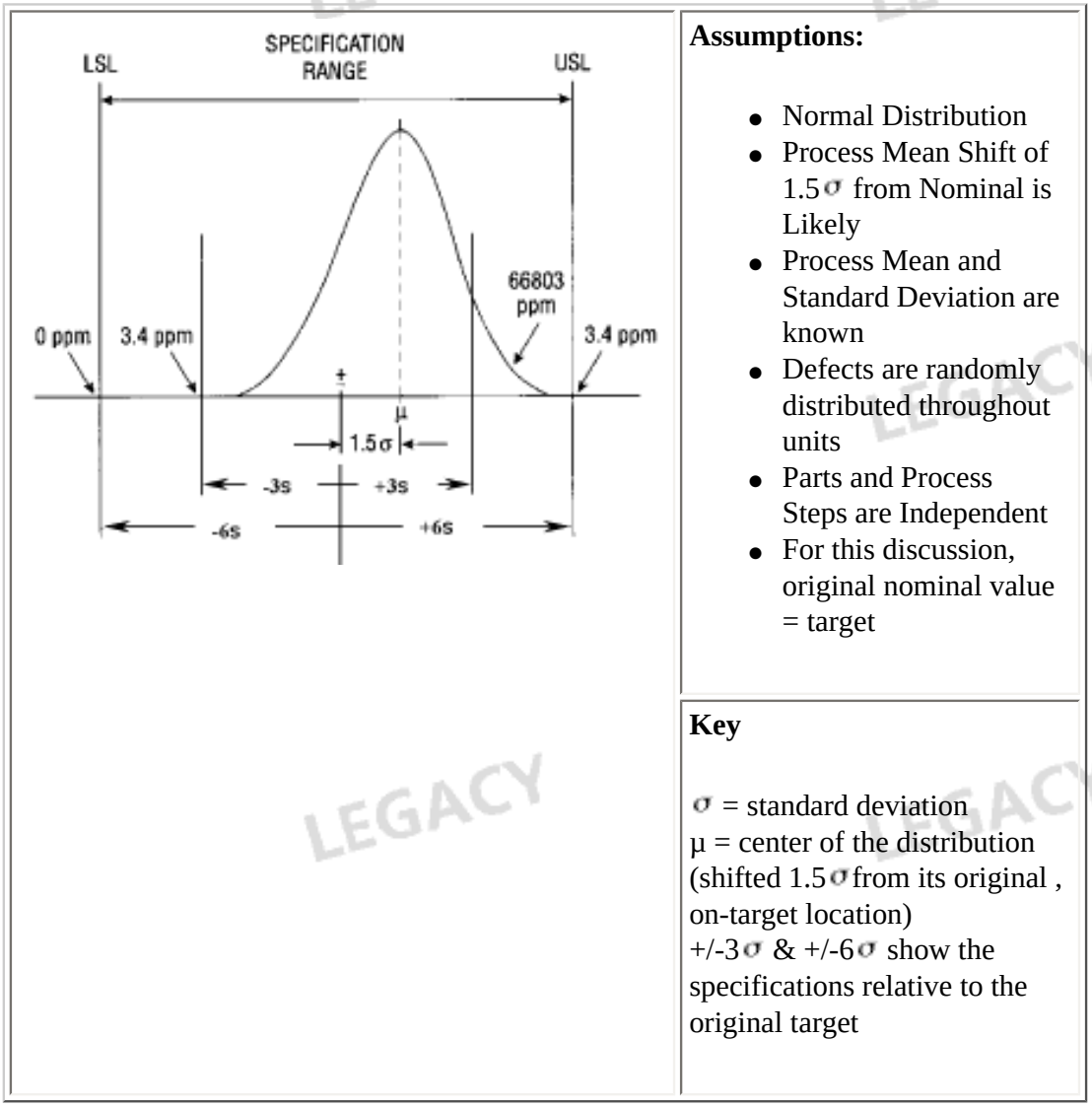


Figure 2: Six Sigma Process with Mean Shifted from Nominal by 1. 5

Usage Considerations

In the software and systems field, Six Sigma may be leveraged differently based on the state of the business. In an organization needing process consistency, Six Sigma can help promote the establishment of a process. For an organization striving to streamline their existing processes, Six Sigma can be used as a refinement mechanism.

In organizations at CMM® level 1-3, "defect free" may seem an overwhelming stretch. Accordingly, an effective approach would be to use the improvement framework ('Define-Measure-Analyze-Improve-Control') as a roadmap toward

intermediate defect reduction goals. Level 1 and 2 organizations may find that adopting the Six Sigma philosophy and framework reinforces their efforts to launch measurement practices; whereas Level 3 organizations may be able to begin immediate use of the framework. As organizations mature to Level 4 and 5, which implies an ability to leverage established measurement practices, accomplishment of true "six sigma" performance (as defined by ppm defect rates) becomes a relevant goal.

Many techniques in the Six Sigma toolkit are directly applicable to software and are already in use in the software industry. For instance, "Voice of the Client" and "Quality Function Deployment" are useful for developing customer requirements (and are relevant measures). There are numerous charting/ calculation techniques that can be used to scrutinize cost, schedule, and quality (project-level and personal-level) data as a project proceeds. And, for technical development, there are quantitative methods for risk analysis and concept/ design selection. The strength of "Six Sigma" comes from consciously and methodically deploying these tools in a way that achieves (directly or indirectly) customer satisfaction.

As with manufacturing, it is likely that Six Sigma applications in software will reach beyond "improvement of current processes/products" and extend to "design of new processes/products." Named "Design for Six Sigma" (DFSS), this extension heavily utilizes tools for customer requirements, risk analysis, design decision-making and inventive problem solving. In the software world, it would also heavily leverage re-use libraries that consist of robustly designed software.

Maturity

Six Sigma is rooted in fundamental statistical and business theory; consequently, the concepts and philosophy are very mature. Applications of Six Sigma methods in manufacturing, following on the heels of many quality improvement programs, are likewise mature. Applications of Six Sigma methods in software development and other 'upstream' (from manufacturing) processes are emerging.

Costs and Limitations

Institutionalizing Six Sigma into the fabric of a corporate culture can require significant investment in training and infrastructure. There are typically three different levels of expertise cited by companies: Green Belt, Black Belt Practitioner, Master Black Belt. Each level has increasingly greater mastery of the skill set. Roles and responsibilities also grow from each level to the next, with Black Belt Practitioners often in team/project leadership roles and Master Black Belts often in mentoring/teaching roles. The infrastructure needed to support the Six Sigma environment varies. Some companies organize their trained Green/ Black Belts into a central support organization. Others deploy Green/Black Belts into organizations based on project needs and rely on communities of practice to maintain cohesion.

Alternatives

In past years, there have been many instances and evolutions of quality improvement programs. Scrutiny of the programs will show much similarity and also clear distinctions between such programs and Six Sigma. Similarities include common tools and methods, concepts of continuous improvement, and even analogous steps in the improvement framework. Differences have been articulated as follows:

- Six Sigma speaks the language of business. It specifically addresses the concept of making the business as profitable as possible.
- In Six Sigma, quality is not pursued independently from business goals. Time and resources are not spent improving something that is not a lever for improving customer satisfaction.
- Six Sigma focuses on achieving tangible results.
- Six Sigma does not include specific integration of ISO900 or Malcolm Baldridge National Quality Award criteria.
- Six Sigma uses an infrastructure of highly trained employees from many sectors of the company (not just the Quality Department). These employees are typically viewed as internal change agents.
- Six Sigma raises the expectation from 3-sigma performance to 6-sigma. Yet, it does not promote "Zero Defects" which many people dismiss as "impossible."

Sources: [Pyzdek 2-01, Marash 99, Harry 00]

Complementary Technologies

It is difficult to concisely describe the ways in which Six Sigma may be interwoven with other initiatives (or vice versa). The following paragraphs broadly capture some of the possible interrelationships between initiatives.

Six Sigma and improvement approaches such as CMM,, CMMI[SM], PSP[SM]/ TSP[SM] are complementary and mutually supportive. Depending on current organizational, project or individual circumstances, Six Sigma could be an enabler to launch CMM®, CMMI[SM], PSP[SM], or TSP[SM]. Or, it could be a refinement toolkit/methodology within these initiatives. For instance, it might be used to select highest priority Process Areas within CMMI[SM] or to select highest leverage metrics within PSP[SM].

Examination of the Goal-Question-Metric (GQM), Initiating-Diagnosing-Establishing-Acting-Leveraging (IDEAL[SM]), and Practical Software Measurement (PSM) paradigms, likewise, shows compatibility and consistency with Six Sigma. GQ(I)M meshes well with the Define-Measure steps of Six Sigma. IDEAL and Six Sigma share many common features, with IDEAL[SM] being slightly more focused on change management and organizational issues and Six Sigma being more focused on tactical, data-driven analysis and decision making. PSM provides a software-tailored approach to measurement that may well serve the Six Sigma improvement framework.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Six Sigma
Application category	Detailed Design (AP.1.3.5) Code (AP.1.4.2) Unit Testing (AP.1.4.3.4) Component Testing (AP.1.4.3.5)
Quality measures category	Reliability (QM.2.1.2) Availability (QM.2.1.1) Maintenance Control (QM.5.1.2.3) Productivity (QM.5.2)
Computing reviews category	Management (D.2.9)

References and Information Sources

[Arnold 99] Arnold, Paul V. *Pursuing the Holy Grail* [online]. Available WWW <URL: http://www.mrotoday.com/mro/archives/Editorials/editJJ1999.htm > (1999).

[ASQ 00] ASQ Statistics Division. *Improving Performance Through Statistical Thinking.* Milwaukee, WI: ASQ Quality Press, 2000.

[ASA 01] American Statistical Association, Quality & Productivity Section. *Enabling Broad Application of Statistical Thinking* [online]. Available WWW <URL: http://web.utk.edu/~asaqp/thinking.html> (2001).

[Bylinsky 98] Bylinsky, Gene. *How to Bring Out Better Products Faster* [online]. Available WWW <URL: http://www.amsup.com/media/fortune.htm> (1998).

[Harrold 99] Harrold, Dave. *Designing for Six Sigma Capability* [Online]. Available WWW <URL: http://www.controleng.com/archives/1999/ctl0101.99/01a103.htm> (1999).

[Harry 00] Harry, Mikel. "Six Sigma: The Breakthrough Management Strategy Revolutionizing the World's Top Corporations." New York, N.Y. Random House Publishers, 2000.

[Lahiri 99] Lahiri, Jaideep. *The Enigma of Six Sigma* [online]. Available WWW <URL: http://www.india-today.com/btoday/19990922/cover.html> (1999).

[Marash 99] Marash, Stanley A. *Six Sigma: Passing Fad or a Sign of Things to Come?* [online]. Available WWW <URL: http://www.thesamgroup.com/sixsigmaarticle.htm> (1999).

[Pyzdek 01] Pyzdek, Thomas. *The Six Sigma Handbook.* New York, N.Y.: McGraw-Hill Professional Publishing, 2001.

[Pyzdek 2-01] Pyzdek, Thomas. *Six Sigma and Beyond: Why Six Sigma Is Not TQM* [online]. Available WWW <URL: http://www.qualitydigest.com/feb01/html/sixsigma.html> (2001).

Current Author/Maintainer

Jeannine Siviy, SEI

External Reviewers

Anita Carleton, SEI
Wolfhart Goethert, SEI
David Zubrow, SEI

Modifications

1 May 2001 (original)

URL: http://www.sei.cmu.edu/str/descriptions/sigma6_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references

- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Simplex Architecture

Software Technology Roadmap

Status

complete

Purpose and Origin

Real-time applications that play a mission-critical role are prevalent throughout the DoD and industry. The complexity of these systems make them expensive to design, *maintain*, and support. Their mission critical nature requires assurance of operational *availability*. These systems are often *safety-critical*, requiring a high degree of *reliability*. The long life cycles of these systems usually result in multiple capability upgrades as well as platform migrations. As the use of COTS products increases, upgrade cycles will become shorter.

Simplex architecture is a paradigm and an engineering framework that permits the quick, easy, and reliable insertion of new capabilities and technologies into mission critical real-time systems [Sha 96]. Simplex is the synthesis of selected best practices in several technology areas that support the safe, online upgrade of hardware and software, in spite of residual errors in the new components. Through the use of Simplex, it becomes possible to shift resources from static design and extensive testing to reliable incremental evolution.

Technical Detail

Software is pervasive within the critical systems that form the infrastructure of modern society, both military and civilian. These systems are often large and complex and require periodic and extensive upgrading. The important technical problems include the following:

- *Integration of new and revised components.* The need for periodic and extensive upgrading and technology refreshment of systems challenges developers to integrate new or changed components into systems without compromising the strict reliability and availability requirements of the applications. There are significant strategic and tactical advantages afforded by the ability to adapt quickly to changing situations. These potential advantages challenge developers to find ways of modifying, upgrading, or adding system components more quickly while reducing the possibility of error.
- *Vendor driven upgrade.* To cut costs and gain leverage from technical advances in the commercial sector, the DoD has encouraged more

frequent use of COTS components in its software. For similar reasons, industry is often following suit. COTS components have a short life cycle (roughly one year.) DoD platforms change at a much slower rate and typically have longer life cycles (often 25-30 years or more). This make the DoD platform susceptible to a problem that occurs when the vendor releases a new version of the COTS component. The upgrade can either be ignored or incorporated into the system. Ignoring it will eventually result in a system that is burdened with unsupported and obsolete components. Incorporating it forces the DoD platform to change on a schedule determined by the vendor, rather than the system developer, maintainer, or customer. New releases usually add features and fix existing bugs, but in the process they also often introduce new bugs. So upgrading is risky; a way to manage the risk is needed.

- *Upgrade paradox.* The upgrade paradox results from the use of replication or functional redundancy and majority voting. A minority upgrade will have no effect because it will be voted out of the system by the majority. A majority upgrade with residual errors can cause the system to fail.

Collectively, these technical problems present a formidable challenge to the developers and maintainers of systems with long life cycles.

Simplex is a framework for system integration and evolution. It integrates a number of technologies, including:

- *Analytic Redundancy.* These technologies are used for integrated availability and reliability management. They employ sophisticated monitoring and switching logic which includes a *simple leadership protocol.* Analytic redundancy allows high-performance, but possibly less-reliable, components to be used in systems demanding a high degree of reliability. This is accomplished without sacrificing the performance and reliability levels provided by existing highly reliable components.
- *Replaceable Units.* These technologies (dynamic binding) allow the replacement of software modules at runtime without having to shut down and restart the system.
- *Publish/Subscribe.* These are flexible real-time group communication technologies that allow components to dynamically publish and subscribe to needed information [Rajkumar 95].
- *Rate Monotonic Scheduling.* These technologies for real-time computing (see Rate Monotonic Analysis) allow components to be replaced or modified in real time, transparently to the applications, while still meeting deadlines. These technologies are integrated into the real-time operating system.

The above technologies are shown in the context of the overall structure of a Simplex-based application in Figure 33.

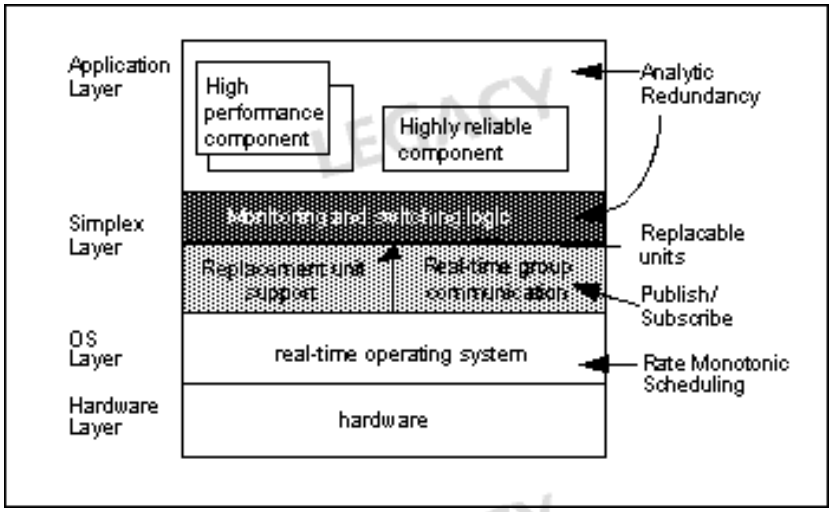


Figure 33: Simplex Technologies and Architecture

Figure 34 is a highly simplified view of the data flow in a system using Simplex. Notice that multiple versions of a component are employed-a *Highly Reliable Component* (HRC) and a *High Performance Component* (HPC). The HRC might be legacy software designed to control the device. It has known performance characteristics and presumably, due to long use, is relatively bug free. If we suppose that the HPC is a new version of the software with improved performance characteristics, but possibly also containing bugs since it has not yet been used extensively, the following scenario takes place.

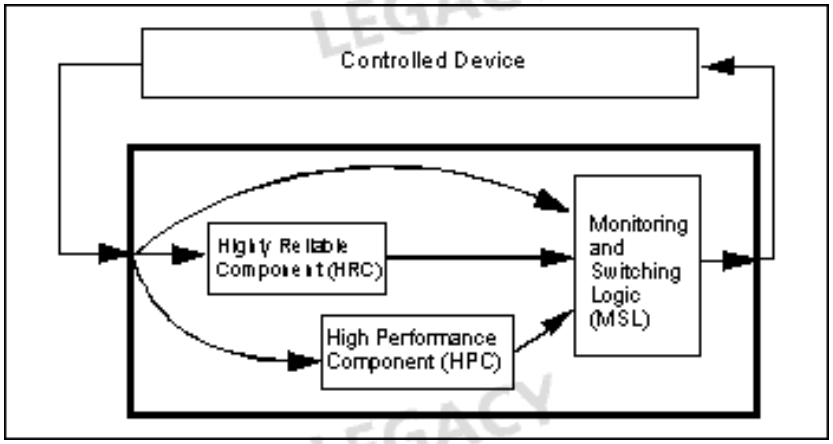


Figure 34: Simplex: Simplified Data Flow

The device under control is sampled at a regular interval. The data is processed by both HRC and HPC. Instead of controlling the device directly, a *simple leadership protocol* is used. Under this protocol, both modules send their results to the *Monitoring and Switching Logic* (MSL), which also uses inputs obtained from the device under control to decide which output to pass back to the device. As long as HPC is behaving properly, it is the leader and its output will be transmitted to the device. Should MSL decide that HPC is not behaving correctly, it makes the HRC the leader and uses its output instead. Thus the device will perform no worse than it did before the upgrade to HPC occurred. This solves the upgrade paradox even in the presence of multiple alternatives because at any instant only the output of one of the alternatives is used. Not shown, for reasons of complexity, is the module that would actually remove a

failed HPC from the system and allow it to be replaced with a corrected version for another try.

Usage Considerations

Simplex is most suitable for systems that have high availability and reliability requirements. It seems especially suitable for systems such as control systems (real-time or process) whose behavior can be modeled and monitored.

Because Simplex is relatively immature, pilot studies will be needed to determine its suitability for any intended application. This would involve developing a rapid prototype, using Simplex, of a simplified instance of the intended application.

Maturity

The safe, online upgrade of both software and hardware, including COTS components, using Simplex has been successfully demonstrated in the laboratory. Simplex is being transitioned into practice via several pilot studies:

- *Silicon Wafer Manufacturing*. The objective was to demonstrate the use of Simplex as the basis for the control architecture in manufacturing process-control software. This was a joint effort between the Software Engineering Institute and the Department of Electrical and Computer Engineering at Carnegie Mellon, guided by engineers from SEMATECH.
- *NSSN (new attack submarine program).* This study involved a US Navy program whose goal is the development, demonstration, and transition of a COTS-based fault-tolerant submarine control system that can be upgraded inexpensively and dependably.
- *INSERT (INcremental Software Evolution for Real-Time Systems)*. This project was funded by the Air Force/DARPA EDCS (evolutionary design of complex software) program, whose goal is to evaluate the possible use of Simplex in the context of onboard avionics systems. Work is proceeding with Lockheed-Martin Tactical Aircraft Systems to investigate the application of this technology to the automated maneuvering capability of the F-16 fighter.

Costs and Limitations

Simplex is designed to support the evolution of mission-critical systems that have high availability or reliability requirements. Its suitability for management information systems (e.g., MIS) applications that do not have such requirements has yet to be determined. Its usefulness in C4I systems is currently being investigated.

Although Simplex has been designed to reduce the life-cycle cost of systems, data on its impact on system life-cycle cost is not available at this time. Much of Simplex is built upon COTS components such as a POSIX compliant real-time operating system running on modern hardware. This tends to reduce costs relative to custom designs.

When using Simplex, engineering costs are increased by the need to analyze and create the analytically redundant modules. Additionally, there is some overhead involved in the operation of the monitoring and switching logic. Finally, the need to run multiple copies of an application (i.e., the HRC and HPC simultaneously) requires additional resources-at the very least additional memory and CPU cycles. These factors tend to have an upward effect on costs-compensated for by the increased reliability and flexibility which Simplex provides.

A perhaps more important consideration is the savings that Simplex provides by reducing the required testing and downtime when installing an upgraded component. The expectation is that the use of Simplex will provide a significant savings in total life-cycle cost.

Complementary Technologies

Software and hardware reliability modeling and analysis allow users to estimate the impact of Simplex on system reliability. System life-cycle cost estimation techniques will allow users to estimate the cost impact.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of Technology	Simplex Architecture
Application category	Reapply Software Life Cycle (AP.1.9.3) Reengineering (AP.1.9.5) Software Architecture (AP.2.1) Restart/Recovery (AP.2.10)
Quality measures category	Availability/Robustness (QM.2.1.1) Reliability (QM.2.1.2) Safety (QM.2.1.3) Real-time Responsiveness/Latency (QM.2.2.2) Maintainability (QM.3.1)
Computing reviews category	Fault-tolerance (D.4.5) Real-time and embedded systems (D.4.7) Network communication (D.4.4)

References and Information Sources

[Altman 97] Altman, Neal. *The Simplex Architecture* [online]. Available WWW <URL: http://www.sei.cmu.edu/simplex/simplex_architecture.html> (May 6, 1997).

[Sha 96] Sha, L.; Rajkumar, R.; & Gagliardi, M. "Evolving Dependable Real Time Systems," 335-346. *Proceedings of the 1996 IEEE Aerospace Applications Conference*. Aspen, CO, February 3-10, 1996. New York, NY: IEEE Computer Society Press, 1996.

[Rajkumar 95] Rajkumar, R.; Gagliardi, M.; & Sha, L. "The Real-Time Publisher/Subscriber Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation," 66-75. *The First IEEE Real-Time Technology and Applications Symposium*. Chicago, IL, May 15-17, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

Current Author/Maintainer

Charles B. Weinstock, SEI
Lui R. Sha, SEI

External Reviewers

John Lehoczky, Professor, Statistics Department, CMU

Modifications

29 Oct 97 changes include:

· Updated list of pilot studies.

· Provided additional detail on constituent technologies.

· Added application architecture diagram.

· Improved data flow diagram and enhanced the explanation.

· Added additional information on anticipated costs (where these are generally understood.)

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/simplex_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Software Inspections

Software Technology Roadmap

Status

Complete

Purpose and Origin

Software Inspections are a disciplined engineering practice for detecting and correcting defects in software artifacts, and preventing their leakage into field operations. Software Inspections were introduced in the 1970s at IBM, which pioneered their early adoption and later evolution [Fagan 76]. Software Inspections provide value in improving software reliability, availability, and maintainability.

Many organizations have made commitments to initiatives in the Capability Maturity Model® (CMM®)[1], ISO 9000, or Six Sigma in order to deliver superior quality. Each of these initiatives has one thing in common: the practice of Software Inspections.

Experienced software practitioners and managers understand that software development is a process of experimentation involving the continuous discovery of technical information associated with the function, form, and fit of the software product. Software Inspections are an integral practice in the process of experimentation.

Software inspections provide value in improving *reliability*, *availability*, and *maintainability*.

Technical Detail

Software Inspections are strict and close examinations conducted on requirements, specifications, architectures, designs, code, test plans and procedures, and other artifacts [Ebenau 94], [O'Neill 01a]. Leading software indicators of excellence for each artifact type provide the exit criteria for the activities of the software life cycle. For example, these indicators include completeness, correctness, style, rules of construction, and multiple views [O'Neill 88,92].

Completeness is based on traceability of the requirements to the code, essential

for maintainability. Correctness is based on the clear specification of intended function and its faithful elaboration in code, essential for reliability and availability [Linger 79]. Style is based on consistency of recording, essential for maintainability. Rules of construction are based on the software application architecture and the specific protocols, templates, and conventions used to carry it out, essential for reliability and availability. Multiple views are based on the various perspectives and viewpoints required to be reflected in the software product, essential for maintainability. By detecting defects early and preventing their leakage into subsequent activities, the need for later detection and rework (which is essential for reduced cycle time and lower cost) is eliminated.

Software Inspections are a reasoning activity performed by practitioners playing the defined roles of moderator, recorder, reviewer, reader, and producer. Each role carries with it the specific behaviors, skills, and knowledge needed to achieve the expert practice of Software Inspections [Freedman 90].

The adoption of Software Inspections practice is competency enhancing and meets little resistance among practitioners trained in their use. The adopting organization benefits by improved predictability in cost and schedule performance, reduced cost of development and maintenance, reduced defects in the field, increased customer satisfaction, and improved morale among practitioners.

The Return on Investment for Software Inspections is defined as net savings divided by detection cost [O'Neill 01a,c]. Savings result from early detection and correction avoiding the increased cost that comes with the detection and correction of defects later in the life cycle. An undetected major defect that escapes detection and leaks to the next phase may cost two to ten times to detect and correct [Basili/Boehm 01]. A minor defect may cost two to four times to detect and correct. The net savings then are up to nine times for major defects and up to three times for minor defects. The detection cost is the cost of preparation effort and the cost of conduct effort.

Usage Considerations

While Software Inspections originated and evolved in new development, its usefulness in maintenance is now well established. Certain measurements obtained during Software Inspections reflect this context of use. For example, the lines of code inspected per conduct hour range from 250 to 500 for new development and from 1000 to 1500 for maintenance. Other measurements reveal no distinction between these contexts of usage. For example, the defects detected per session range from five to ten for both new development and maintenance.

The organization adopting Software Inspections practice seeks to prevent defect leakage from one life cycle activity to another. Following training, the organization can expect to detect 50% of the defects present. It may take 12 to 18 months to achieve expert practice where defect detection is expected to range from 60% to 90% [O'Neill 89], [O'Neill 01a].

Maturity

The maturity of a technology can be reasoned about in terms of its long-term, widespread use in a variety of usage domains and its transition from early adopters through late adopters. Software Inspections have been evolving for 25 years. They are known to deliver economic value.

The data discussed in Usage Considerations above and Costs and Limitations below are drawn from the National Software Quality Experiment (NSQE) [O'Neill 95,96,00] where thousands of participants from dozens of organizations are populating the experiment database with thousands of defects of all types along with pertinent information needed to pinpoint their root causes. The analysis bins identified in the experiment include software process maturity level (1,2,3...), organization type (government, Department of Defense (DoD) industry, commercial), product type (embedded, organic), programming language (old style, modern), and global region (North America, Pacific Rim, Latin America).

Organizations are invited to calibrate their Software Inspection results with the NSQE results using the Software Inspection Measurement and Derived Metrics tool [O'Neill 01b] found at http://members.aol.com/ONeillDon/nsqe-assessment.html.

Software Inspections are a rigorous form of peer reviews, a Key Process Area (kpa) of the CMM [Paulk 95], [Humphrey 89]. Although peer reviews are part of achieving CMM level 3, and many organizations limit their software process improvement agenda to the kpas for the maturity level they are seeking to achieve, the population of Software Inspections adopters ranges from level 1 to 5.

Costs and Limitations

The rollout and operating costs associated with Software Inspections include the initial training of practitioners and managers, the ongoing preparation and conduct of inspection sessions, and the ongoing management and use of measurement data for defect prevention and return on investment computations.

To properly adopt Software Inspections practice, each participant is trained in the structured review process, defined roles of participants, system of process and product checklists, and forms and reports. The lost opportunity cost to acquire the knowledge, skills, and behaviors is twelve hours per practitioner [O'Neill 89]. In addition, each manager is trained in the responsibilities for rolling out the technology and the interpretation and use of measurements taken. The management training is accomplished in four hours.

The cost of performing Software Inspections includes the individual preparation effort of each participant before the session and the conduct effort of participants in the inspections session. Typically, 4-5 people participate and expend 1-2 hours of preparation and 1-2 hours of conduct each. This cost of 10 to 20 hours of total effort per session results in the early detection of 5-10 defects in 250-500 lines of new development code or 1000-1500 lines of legacy code [O'Neill

95,96,00].

The National Software Quality Experiment (NSQE) [O'Neill 95,96,00] reveals that the Return on Investment (net savings/detection cost) for Software Inspections ranges from four to eight independent of the context of usage. Organizations are invited to calibrate their software inspection return on investment using the tool [O'Neill 01c] found at http://members.aol.com/ONeillDon/nsqe-roi.html.

Dependencies

In order for Software Inspections to be systematically used in statistical process control, there must be a life cycle model with defined software artifacts. In this context, Software Inspections provide the exit criteria for each life cycle activity. Furthermore, the standard of excellence of leading indicators for each type of artifact must be specified and used in practice.

Alternatives

While Software Inspections are a rigorous form of peer reviews, software walkthroughs are a less rigorous form of peer reviews [O'Neill 01a]. Walkthroughs may cost as much as inspections, but they deliver less. Notably, walkthroughs provide no measured results and that precludes the application of statistical process control needed to advance software process maturity.

Complementary Technologies

To optimize the practice of Software Inspections on legacy code during maintenance operations, all modules are rank ordered by cyclomatic complexity. Candidates for inspection are selected from those with highest complexity rating where the defect density is expected to be high.

This legacy code maintenance strategy can be extended by rank ordering all modules based upon incidents encountered in the past year and by rank ordering the modules expected to be adapted and perfected in the coming year. Modules for inspection are then selected based on their rank ordering in cyclomatic complexity, defect history, and expected rework.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Software Inspections

Application category	Detailed Design (AP.1.3.5), Code (AP.1.4.2), Unit Testing (AP.1.4.3.4), Component Testing (AP.1.4.3.5)
Quality measures category	Correctness (QM.1.3), Reliability (QM.2.1.2), Availability (QM.2.1.1), Maintainability (QM.3.1)
Computing reviews category	Program Verification (D.2.4), Testing and Debugging (D.2.5)

References and Information Sources

[Basili/Boehm 01] Basili, Vic, & Barry Boehm. "Software Defect Reduction Top 10 List." *Computer 34*,1, (January 2001): 135-137.

[Ebenau 94] Ebenau, Robert G. & Strauss, Susan H. *Software Inspection Process*. New York, NY: McGraw-Hill, 1994.

[Fagan 76] Fagan, M. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal 15*, 3 (1976): 182-211.

[Freedman 90] Freedman, D.P. & Weinberg, G.M. *Handbook of Walkthroughs, Inspections, and Technical Reviews*. New York, NY: Dorset House, 1990.

[Gilb 93] Gilb, Tom & Graham, Dorothy. *Software Inspection.* Essex, England: Addison-Wesley Longman Ltd., 1993.

[Humphrey 89] Humphrey, Watts S. Managing the Software Process. Reading, MA: Addison-Wesley, 1989.

[Linger 79] Linger, R.C.; Mills, H.D.; & Witt, B.I. Structured Programming: Theory and Practice. Reading, MA: Addison-Wesley, 1979.

[O'Neill 88] O'Neill, Don & Ingram, Albert L. "Software Inspections Tutorial," 92-120. *Software Engineering Institute Technical Review 1988*. Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute, 1988.

[O'Neill 89] O'Neill, Don. *Software Inspections Course and Lab*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1989.

[O'Neill 92] O'Neill, Don. "Software Inspections: More Than a Hunt for Errors." Crosstalk, Journal Of Defense Software Engineering 30 (January 1992): 8-10.

[**O'Neill 95,96,00**] O'Neill, Don. "National Software Quality Experiment: Results 1992-1999." Software Technology Conference, Salt Lake City, 1995, 1996, and 2000.

[O'Neill 01a] O'Neill, Don. *Peer Reviews.* Encyclopedia of Software Engineering. New York, New York: Wiley Publishing, Inc., to appear 2001.

[O'Neill 01b] O'Neill, Don. Software Inspection Measurements and Derived Metrics Tool [online]. Available WWW <URL: http://members.aol.com/ONeillDon/nsqe-assessment.html> (2001).

[O'Neill 01c] O'Neill, Don. Return on Investment Tool [online]. Available WWW <URL: http://members.aol.com/ONeillDon/nsqe-roi.html> (2001).

[Paulk 95] Paulk, Mark C. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley Publishing Company, 1995.

Current Author/Maintainer

Don O'Neill, Don O' Neill Consulting

External Reviewers

Alex Elentukh, Fidelity Investments
Rick Linger, SEI
Watts Humphrey, SEI
Joan Weszka, Lockheed Martin

Modifications

10 Jan 97 (original)
1 April 01 (updated)

Footnotes

[1] Capability Maturity Model and CMM are service marks of Carnegie Mellon University.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/inspections_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Statistical-Based Intrusion Detection

Software Technology Roadmap

Status

Advanced

Note

We recommend Intrusion Detection as prerequisite reading for this technology description.

Purpose and Origin

Intrusion detection systems (IDS) automate the detection of security violations through computer processing of system audit information. One IDS approach, Rule-Based Intrusion Detection (RBID) , seeks to identify intrusion attempts by matching audit data with known patterns of intrusive behavior. RBID systems rely on codified rules of known intrusions to detect intrusive behavior. Intrusion attempts not represented in an RBID rule base will go undetected by these systems. To help overcome this limitation, statistical methods have been employed to identify audit data that may *potentially* indicate intrusive or abusive behavior. Known as statistical-based intrusion detection (SBID) systems, these systems analyze audit trail data by comparing them to typical or predicted profiles in an effort to find pending or completed computer security violations. This emerging technology seeks to increase the *availability* of computer systems by automating the detection and elimination of intrusions.

Technical Detail

SBID systems seek to identify abusive behavior by noting and analyzing audit data that deviates from a predicted norm. SBID is based on the premise that intrusions can be detected by inspecting a system's audit trail data for unusual activity, and that an intruder's behavior will be noticeably different than that of a legitimate user. Before unusual activity can be detected, SBID systems require a characterization of user or system activity that is considered "normal." These characterizations, called *profiles*, are typically represented by sequences of events that may be found in the system's audit data. Any sequence of system events deviating from the expected profile by a statistically significant amount is flagged as an intrusion attempt [Sundaram 96]. The main advantage of SBID systems is that intrusions can be detected without *a priori* information about the security flaws of a system [Kemmerer 94].

SBID systems typically employ statistical anomaly and rule-based misuse models [Mukherjee 94]. System profiles, user profiles, or both may be used to define expected behavior. User profiles, if used, are specific to each user and are dynamically maintained. As a user's behavior changes over time, so too will his user profile. No such profiles are used in RBID systems. As is the case with RBID systems, known intrusion scenarios can be codified into the rule base of SBID systems.

Interesting variations on this theme include the following:

- Predictive pattern generation, which uses a rule base of user profiles defined as statistically-weighted event sequences [Teng 90]. This method of intrusion detection attempts to predict future events based on events that have already occurred. Advantages of this approach include its ability to detect misuse as well as intrusions and its ability to detect and respond quickly to anomalous behavior.
- Connectionist approaches in which neural networks are used to create and maintain behavior profiles [Lunt 93]. Advantages of neural approaches include their ability to cope with noisy data and their ability to adapt to new user communities. Unfortunately, trial and error is required to train the net, and it is possible for an intruder to train the net during its learning phase to ignore intrusion attempts [Sundaram 96].

Usage Considerations

An advantage of SBID systems is that they are able to adaptively learn the behavior of the users they monitor and are thus potentially more sensitive to intrusion attempts than are humans [Sundaram 96, Lunt 93]. However, SBID systems require the creation and maintenance of user/system profiles. These profiles are sensitive to hardware and software modifications, and will need to be updated whenever the system or network they used to protect is modified. Additional work is required to determine how statistical user/system profiles should be created and maintained [Lunt 93].

Maturity

Statistical intrusion detection algorithms have been in existence since at least 1988. Several prototype systems have been developed, including Haystack [Smaha 88], IDES [Lunt 93], and MIDAS [Mukherjee 94]. MIDAS is a deployed real-time SBID that provides security protection for the National Computer Center's networked mainframe computer. IDES, which is deployed at both SRI and FBI locations, is an IDS that combines SBID with RBID to detect a wider range of intrusion attempts. Another deployed security system containing aspects of SBID technology is AT&T Bell Lab's Dragons system which protects their Internet gateway;[1] the Dragons system has succeeded in detecting intrusion attempts ranging from attempted "guest" logins to forged NFS packets [Mukherjee 94].

Costs and Limitations

In addition to the costs associated with creating audit trails and maintaining user profiles, there are several risks and limitations associated with SBID technology:

- Because user profiles are updated periodically, it is possible for an insider to slowly modify his behavior over time until a new behavior pattern has been established within which an attack can be safely mounted [Lunt 93].
- Determining an appropriate threshold for "statistically significant deviations" can be difficult. If the threshold is set too low, anomalous activities that are not intrusive are flagged as intrusive (false positive). If the threshold is set too high, anomalous activities that are intrusive are not flagged as intrusive (false negative).
- Defining user profiles may be difficult, especially for those users with erratic work schedules/habits.

Like RBID systems, SBID systems will negatively affect *throughput* because of to the need to collect and analyze audit data. However, in contrast with RBID systems, SBID systems do not always lag behind the intruders. Detection of anomalous behavior, whether or not it is codified as a known intrusion attempt, may be sufficient grounds for an SBID system to detect an intruder.

Use of this technology requires personnel who are experienced in statistics and intrusion detection techniques and who have in-depth knowledge of audit collection mechanisms.

Dependencies

Expert systems are an enabler for this technology.

Alternatives

Other approaches to intrusion detection include model-based or rule-based approaches (see Rule-Based Intrusion Detection), and approaches based on genetic algorithms. Manual examination of recorded audit data and online monitoring of access activity by knowledgeable personnel are the only other known alternatives.

Complementary Technologies

Rule-Based Intrusion Detection systems can be used in conjunction with statistical-based intrusion detection systems to catch a wider variety of intrusion attempts, and user authentication systems can be used to help verify user identify.

Index Categories

This technology is classified under the following categories. Select a category for

a list of related topics.

Name of technology	Statistical-Based Intrusion Detection
Application category	System Security (AP.2.4.3)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating Systems Security and Protection (D.4.6) Computer-Communication Networks Security and Protection (C.2.0) Security and Protection (K.6.5)

References and Information Sources

[Bell 76]	Bell, D. E. & LaPadula, L. J. *Secure Computer System: Unified Exposition and Multics Interpretation* Rev. 1 (MTR-2997). Bedford, MA: MITRE Corporation, 1976.
[Kemmerer 94]	Kemmerer, Richard A. "Computer Security," 1153-1164. *Encyclopedia of Software Engineering.* New York, NY: John Wiley and Sons, 1994.
[Lunt 93]	Lunt, Teresa F. "A Survey of Intrusion Detection Techniques." *Computers and Security 12,* 4 (June 1993): 405-418.
[Mukherjee 94]	Mukherjee, Biswanath, L.; Heberlein, Todd; & Levitt, Karl N. "Network Intrusion Detection." *IEEE Network 8,* 3 (May/June 1994): 26-41.
[Smaha 88]	Smaha, Stephen E. "Haystack: An Intrusion Detection System," 37-44. *Proceedings of the Fourth Aerospace Computer Security Applications Conference*. Orlando, Florida, December 12-16, 1988. Washington, DC: IEEE Computer Society Press, 1989.
[Spafford 88]	Spafford, Eugene H. *The Internet Worm Program: An Analysis* (CSD-TR-823). West Lafayette, IN: Purdue University, 1988.
[Sundaram 96]	Sundaram, Aurobindo. *An Introduction to Intrusion Detection* [online]. Available WWW <URL: http://www.acm.org/crossroads/xrds2-4/xrds2-4.html> (1996).
[Teng 90]	Teng, Henry S.; Chen, Kaihu; & Lu, Stephen C. "Security Audit Trail Analysis Using Inductively Generated Predictive Rules," 24-29. *Sixth Conference on Artificial Intelligence Applications*. Santa Barbara, CA, May 5-9, 1990. Los Alamitos, CA: IEEE Computer Society Press, 1990.

Current Author/Maintainer

Mark Gerken, Air Force Rome Laboratory

Modifications

10 Jan 97 (original)

Footnotes

[1] See http://www.research.att.com for more details.

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Statistical Process Control for Software

Software Technology Roadmap

Status

Complete

Purpose and Origin

The demand for increased efficiency and effectiveness of our software processes places measurement demands on the software engineering community beyond those traditionally practiced. Statistical and process thinking principles lead to the use of statistical process control methods to determine the consistency and capability of the many processes used to develop software.

Technical Detail

Over the past decade, the concepts, methods, and practices associated with process management and continual improvement have gained wide acceptance in the software community. These concepts, methods, and practices embody a way of thinking, a way of acting, and a way of understanding the data generated by processes that collectively result in improved quality, increased productivity, and competitive products. The acceptance of this "process thinking" approach has motivated many to start measuring software processes that are responsive to questions relating to process performance [Florac 99]. In that vein, traditional software measurement and analysis methods of measuring "planned versus actual" is not sufficient for measuring process performance or for predicting process performance. The time has come to marry, if you will, "process thinking" with "statistical thinking."

"Statistical thinking" [Britz 97] embraces three principles

1. all work occurs in a system of interconnected processes
2. variation exists in all processes
3. understanding and reducing variation are keys to success

If we examine the basis for these "process thinking" and "statistical concepts", we find that they are founded on the principles of statistical process control. These principles hold that by establishing and sustaining stable levels of variability, processes will yield predictable results. We can then say that the processes are under statistical control. Controlled processes are stable processes, and stable processes enable you to predict results. This in turn

enables you to prepare achievable plans, meet cost estimates and scheduling commitments, and deliver required product functionality and quality with acceptable and reasonable consistency. If a controlled process is not capable of meeting customer requirements or other business objectives, the process must be improved or retargeted.

When we relate these notions of process and statistical thinking to the operational level, we realize a key concern of process management is that of process performance &endash; how is the process performing now (effectiveness, efficiency), and how can it be expected to perform in the future? In the context of obtaining quantified answers to these questions, we can address this issue by deconstructing the question of process performance into three parts.

First we should examine process performance in terms of compliance. For example, is the process being executed properly? Is the personnel trained? Are the right tools available? If the process is not in compliance, we know there is little chance of it performing satisfactorily.

If a process is compliant, the next question is: Is the process performance (execution) reasonably consistent over time? Is the effort, cost, elapsed time, delivery, and quality consumed and produced by executing the process consistently? Realizing that variation exists in all processes, is the variation in process performance predictable?

Finally, if the process performance is consistent, we ask the question: Is the process performing satisfactorily? Is it meeting the needs of interdependent processes and/or of the needs of the customers? Is it effective and efficient?

Historically, software organizations have addressed the question of compliance by conducting assessments, such as comparing the organizations' software process against a standard (e.g., the CMM). Such an assessment provides a picture of the process status at a point in time and indicates the organization's capacity to execute various software processes according to the standard's criteria. However, it does not follow that the process is executed consistently or efficiently merely because the assessment results satisfied all the criteria.

The questions of process consistency, effectiveness, and efficiency require a measurement of process behavior as it is executed over time. Other disciplines have addressed this issue by using statistical process control methods, specifically using Shewhart control charts. They have concluded that control charts provide the basis for making process decisions and predicting process behavior.

Successful use of control charts by other disciplines suggest it is time to examine how statistical process control techniques can help to address our software process issues. In so doing, we find that Shewhart's control charts provide a statistical method for distinguishing between variation caused by normal process operation and variation caused by anomalies in the process. Additionally, Shewhart's control charts provide an operational definition for determining process stability or consistency and predictability as well as quantitatively establishing process capability to meet criteria for process

effectiveness and efficiency.

We use the term software process to refer not just to an organization's overall software process, but to any process or subprocess used by a software project or organization. In fact, a good case can be made that it is only at subprocess levels that true process management and improvement can take place. Thus, we view the concept of software process as applying to any identifiable activity that is undertaken to produce or support a software product or service. This includes planning, estimating, designing, coding, testing, inspecting, reviewing, measuring, and controlling, as well as the subtasks and activities that comprise these undertakings.

Process Performance Variation

The basis for control charts is recognition of two types of variation: common cause variation and assignable cause variation.

Common cause variation is variation in process performance due to normal or inherent interaction among the process components (people, machines, material, environment, and methods). Common cause variation of process performance is characterized by a stable and consistent pattern over time, as illustrated in Figure 1. Variation in process performance due to common cause is thus random, but will vary within predictable bounds. When a process is stable, the random variations that we see all come from a constant system of chance causes. The variation in process performance is predictable, and unexpected results are extremely rare.

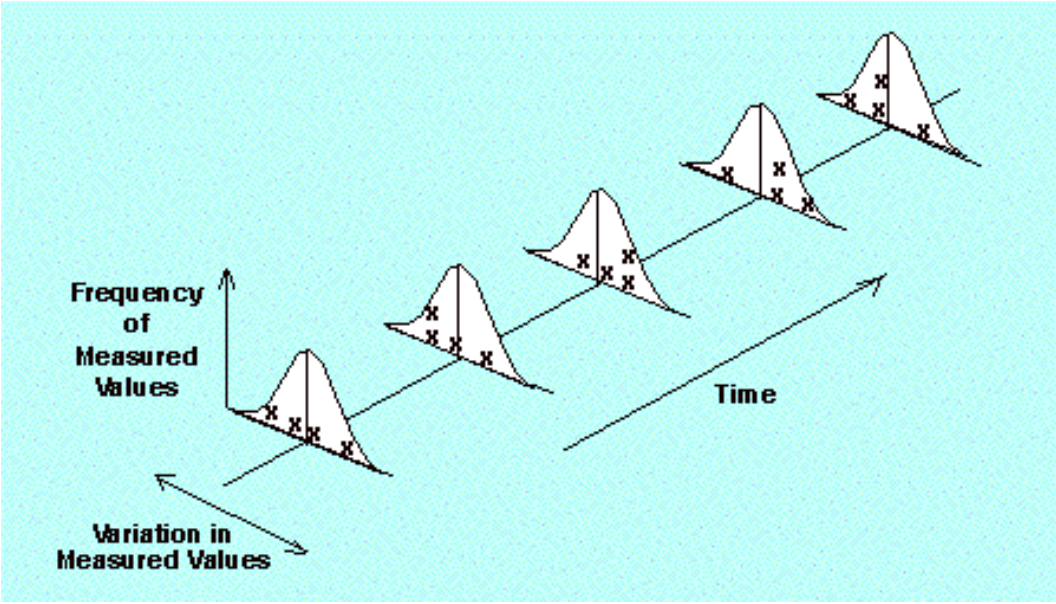


Figure 1: The Concept of Controlled Variation

The key word in the paragraph above is "predictable." Predictable is synonymous with "in control."

The other type of variation in process performance is due to assignable causes. Assignable cause variation has marked impacts on product characteristics and other measures of process performance. These impacts create significant

changes in the patterns of variation. This is illustrated in Figure 2, which we have adapted from Wheeler and Chambers [Wheeler 92]. Assignable cause variations arise from events that are not part of the normal process. They represent sudden or persistent abnormal changes to one or more of the process components. These changes can be in things such as inputs to the process, the environment, the process steps themselves, or the way in which the process steps are executed. Examples of assignable causes of variation include shifts in the quality of raw materials, inadequately trained people, changes to work environments, tool failures, altered methods, failures to follow the process, and so forth.

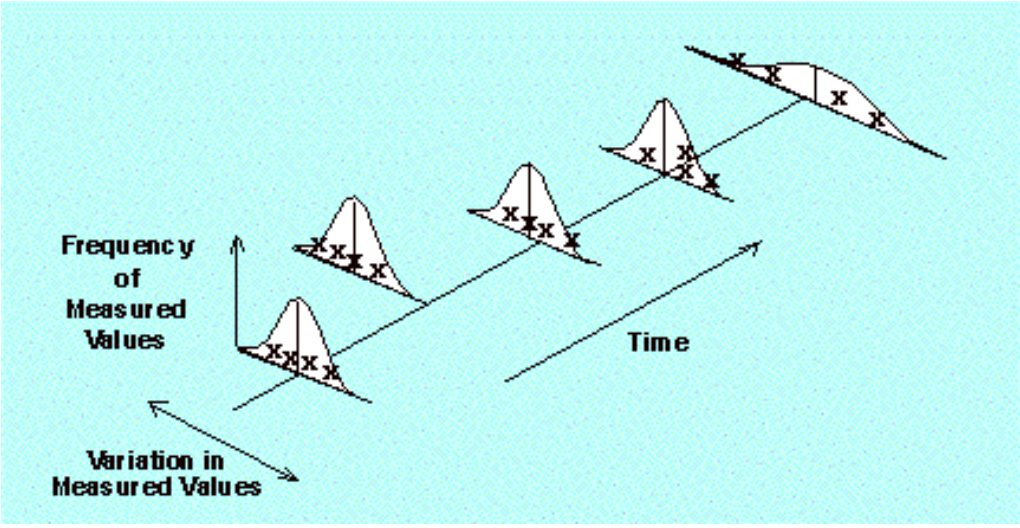


Figure 2: The Concept of Uncontrolled or Assignable Cause Variation

When all assignable causes have been removed and prevented from reoccurring in the future so that only a single, constant system of chance causes remains, we have a stable and predictable process.

Stability of a process with respect to any given attribute is determined by measuring the attribute and tracking the results over time. If one or more measurements fall outside the range of chance variation, or if systematic patterns are apparent, the process may not be stable. We must then look for the causes of deviation, and remove any that we find, if we want to achieve a stable and predictable state of operation.

When a process is stable, 99+% of process performance variation will fall within 3 sigma of the mean or average of the variation. When the process variation falls outside of the 3 sigma limits, the variation is very likely caused by an anomaly in the process.

When a process is stable, or nearly so, the 3 sigma limits determine the amount of variation that is normal or natural to the process. This is the "voice of the process" or the process telling us what it is capable of doing. This may or may not be satisfactory to the customer: if it is, it is "capable"; if it is not, the process must be changed since we know that the remaining variation is due to the process itself.

Three Important Factors

Before we look at an example, there are three important notions that should be discussed

1. the importance of operational definitions
2. homogeneity
3. issues of rational subgrouping

The need for operational definitions is fundamental to any measurement activity. It is not enough to identify measures. Measures must be defined in such a way as to tell others exactly how each measure is obtained so that they can collect and interpret the values correctly.

The primary issue is not whether a definition for a measure is correct, but that everyone understands, completely, what the measured values represent. Only then can people be expected to collect values consistently and have others interpret and apply the results to reach valid conclusions.

Communicating clear and unambiguous definitions is not easy. Having structured methods for identifying all the rules that are used to make and record measurements can be very helpful in ensuring that important information does not go unmentioned. When designing methods for defining measures, one should keep in mind that things that do not matter to one user are often important to another. This means that measurement definitions (and structures for recording the definitions) often become larger and more encompassing than the definitions most organizations have traditionally used. This is all the more reason to have a well-organized approach. Definition focuses on details, and structured methods help ensure that all details get identified, addressed, and recorded. They also help negotiating with people who believe that attention to detail is no longer their responsibility.

Operational definitions must satisfy two important criteria [Park 92]

1. communication. If someone uses the definition as a basis for measuring or describing a measurement result, will others know precisely what has been measured, how it was measured, and what has been included and excluded?
2. repeatability. Could others, armed with the definition, repeat the measurements and get the same results?

These criteria are closely related. In fact, if you can't communicate exactly what was done to collect a set of data, you are in no position to tell someone else how to do it. Far too many organizations propose measurement definitions without first determining what users of the data will need to know about the measured values in order to use them intelligently. It is no surprise, then, that measurements are often collected inconsistently and at odds with users' needs. When it comes to implementation, rules such as, "Count all noncomment, nonblank source statements" or "Count open problems" are open to far too many interpretations to provide repeatable results

Although communicating measurement definitions in clear, unambiguous terms requires effort, there is good news as well. When someone can exactly describe what has been collected, it is easy to turn the process around and say, "Please

do that again." Moreover, you can give the description to someone else and say, "Please use this as your definition, but with these changes." In short, when we can communicate clearly what we have measured, we have little trouble creating repeatable rules for collecting future data.

Next, the notions of homogeneity and rational subgrouping need to be understood and addressed. Homogeneity and rational subgrouping go hand in hand. Because of the non-repetitive nature of software products and processes, some believe it is difficult to achieve homogeneity with software data. The idea is to understand the theoretical issues and at the same time, work within some practical guidelines. We need to understand what conditions are necessary to consider the data homogeneous. When more than two data values are placed in a subgroup, we are making a judgement that these values are measurements taken under essentially the same conditions, and that any difference between them is due to natural or common variation. The primary purpose of homogeneity is to limit the amount of variability within the subgroup data. One way to satisfy the homogeneity principle is to measure the subgroup variables within a short time period. Since we are not talking about producing widgets but software products, the issue of homogeneity of subgroup data is a judgement call that must be made by one with extensive knowledge of the process being measured.

The principle of homogeneously subgrouped data is important when we consider the idea of rational subgrouping. That is, when we want to estimate process variability, we try to group the data so that assignable causes are more likely to occur between subgroups than within them. Control limits become wider and control charts less sensitive to assignable causes when containing non-homogeneous data. Creating rational subgroups that minimize variation within subgroups always takes precedence over issues of subgroup size.

Using Control Charts

Now let's examine how control charts can be used to investigate process stability and lead to process improvement. There are a number of different kinds of control charts (please see [Florac 99] for a more detailed discussion on this and other topics). In software environments, measurements often occur only as individual values. As a result, there may be a preference to using the individuals and moving range (XmR) charts to examine the time-sequenced behavior of process data.

For example, the figure below shows an XmR control chart for the number of reported but unresolved problems backlogged over the first 30 weeks of system testing. The chart indicates that the problem resolution process is stable, and that it is averaging about 20 backlogged problems (the center line, CL, equals 20.4), with an average change in backlog of 4.35 problems from week to week. The upper control limit (UCL) for backlogged problems is about 32, and the lower control limit (LCL) is about 8. If future backlogs were to exceed these limits or show other forms of nonrandom behavior, it would be likely that the process has become unstable. The causes should then be investigated. For instance, if the upper limit is exceeded at any point, this could be a signal that there are problems in the problem-resolution process. Perhaps a particularly thorny defect is consuming resources, causing problems to pile up. If so, corrective action

must be taken if the process is to be returned to its original (characteristic) behavior.

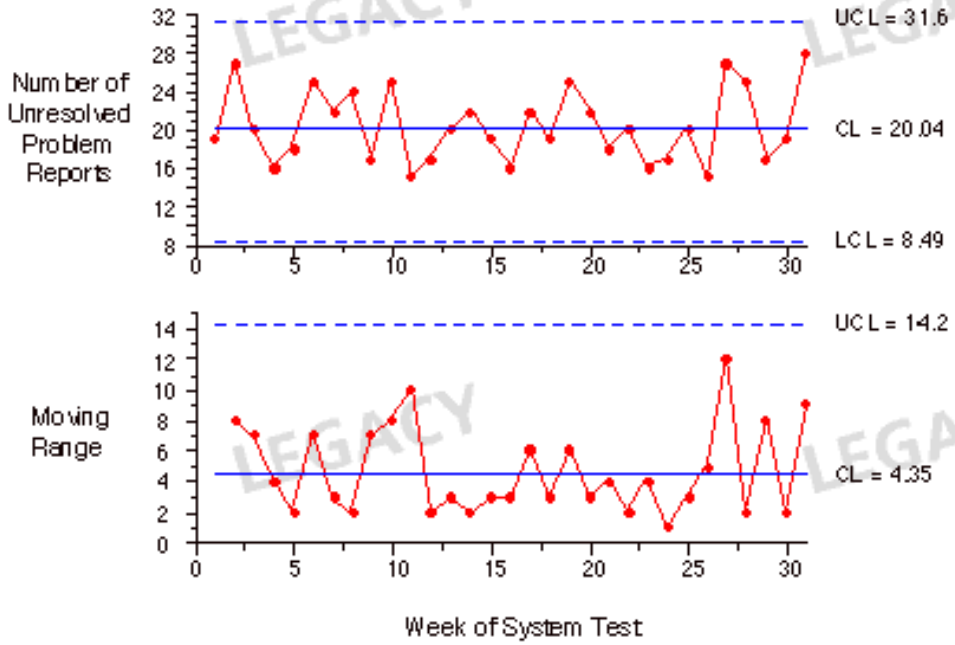


Figure 3: Control Chart for the Backlog of Unresolved Problems

We must be careful not to misinterpret the limits on the individual observations and moving ranges that are shown in the control chart. These limits are estimates for the limits of the process, based on measurements of the process performance. The process limits together with the center lines are sometimes referred to as the "voice of the process."

The performance indicated by the voice of the process is not necessarily the performance that needs to be provided to meet the customer's requirements. If the variability and location of the measured results are such that the processdoes not meet the customer requirement or specification (e.g., produces too many nonconforming products), the process must be improved. This means reducing the process performance variability, moving the average, or both.

Usage Considerations

1. When analyzing process performance data, all sources of variation in the process must be identified. If a conscious effort is not made to account for the potential sources of variation, variations that could help to improve the process might inadvertantly be hidden or obscured. Even worse, it could lead to a faulty analysis. When data are aggregated, the results will be particularly susceptible to overlooked or hidden sources of variation. Overly aggregated data come about in many ways, but the most common causes are

- inadequately formulated operational definitions of product and process measures
- inadequate description and recording of context information
- lack of traceability from data back to the context from where it originated
- working with data whose elements are combinations (mixtures) of values from non-homogeneous sources or different cause systems

Overly aggregated data easily lead to:

- difficulty in identifying instabilities in process performance
- difficulty in tracking instabilities to assignable causes
- using results from unstable processes to draw inferences or make predictions about capability or performance
- anomalous process behavior patterns

2. When measured values of continuous variables have insufficient granularity (i. e., are coarse and imprecise), the discreteness that results can mask the underlying process variation. Computations for $\overline{X}$ and sigma can then be affected, and individual values that are rounded or truncated in the direction of the nearest control limit can easily give false out-of-control signals.

There are four main causes of coarse data: inadequate measurement instruments, imprecise reading of the instruments, rounding, and taking measurements at intervals that are too short to permit detectable variation to occur. When measurements are not obtained and recorded with sufficient precision to describe the underlying variability, digits that contain useful information will be lost. If the truncation or rounding reduces the precision in recorded results to only one or two digits that change, the running record of measured values will show only a few levels of possible outcomes.

3. Control charts can be used to serve many different purposes. Control charts can be helpful for monitoring processes from release to release to compare overall performance. They can be used for making process adjustments to ensure that stability is maintained for a process on a daily or weekly basis. Most importantly control charts may be used for continuous improvement of a process that is stable and capable. It is important to keep in mind however, that the control charts provide the most value to the people or team where the process knowledge resides.

Management can also help set the example of how not to use the control charts. While the control charts can be used to improve personal performance, management should not misuse this tool or the data. Management has to remember that the old saw "we will continue the beatings until morale improves," comes into play whenever measurements are used as part of the "beating." Clearly, dysfunctional behavior is likely to occur if employees perceive that measurements are being used in this way

There is evidence that Shewhart's control charts can play a significant role in measuring process performance consistency, and process predictability . Successful implementers of this process recognize the importance to1) understand the concepts of variation, data homogeneity, common cause systems, and rational subgrouping, and 2) fully understand the process and subprocesses being measured. Furthermore, they have used the control charts to measure process performance at the subprocess (and lower) level realizing that there is far too much variation in the overall process to be helpful in identifying possible actions for improvement.

These software organizations have come to appreciate the value added when control charts are used to provide engineers and managers with quantitative insights into the behavior of their software development processes. In many ways the control chart is a form of instrumentation. Much like an oscilloscope, a temperature probe, or a pressure gauge, it provides data to guide decisions and judgements by process knowledgeable engineers and managers.

Maturity

While SPC is not a new technology, (i.e., this technique has been applied in manufacturing for years) it is just recently being applied to address software engineering improvement. Organizations are starting to become aware of SPC, getting appropriate training, and starting to apply SPC. To get started, many organizations are analyzing inspection data using SPC.

References and Information Sources

[Austin 96]	Robert D. Austin, *Measuring and Managing Performance in Organizations*, Dorset House Publishing, ISBN: 0-932633-36-6, New York, NY, 1996.
[Basili 92]	V.R. Basili, *Software Modeling and Measurement: The Goal/ Question/Metric Paradigm*, University of Maryland, CS-TR-2956, UMIACS-TR-92-96, 1992.
[Brassard 94]	Michael Brassard and Diane Ritter, *The Memory Jogger II*, GOAL/ QPC, Methuen, MA, 1994.
[Burr 96]	Adrian Burr and Mal Owen, *Statistical Methods for Software Quality*, ISBN 1-85032-171-X, International Thomson Computer Press, Boston, MA, 1996.
[Deming 86]	W. Edwards Deming, *Out of the Crisis*, MIT Center for Advanced Engineering Study, Cambridge, MA, 1986.
[Florac 99]	William A. Florac and Anita D. Carleton, *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, Addison &endash;Wesley, 1999.
[Hare 95]	Lynne B. Hare, Roger W. Hoerl, John D. Hromi, and Ronald D. Snee, *The Role of Statistical Thinking in Management*, ASQC Quality Progress, Vol. 28, No. 2, February 1995, pp. 53-60.

[Humphrey 95]	Watts S. Humphrey, *A Discipline for Software Engineering*, ISBN 0-201-54610-8, Addison-Wesley Publishing Company, Reading, MA, 1995.
[Ishikawa 86]	K. Ishikawa, *Guide to Quality Control*, Asian Productivity Organization, Tokyo, Japan, (available from Unipub - Kraus International Publications, White Plains, NY) 1986.
[Paulk 95]	Carnegie Mellon University, Software Engineering Institute (Principal Contributors and Editors: Mark C. Paulk, Charles V. Weber, Bill Curtis, and Mary Beth Chrissis), *The Capability Maturity Model: Guidelines for Improving the Software Process*, ISBN 0-201-54664-7, Addison-Wesley Publishing Company, Reading, MA, 1995.
[Wheeler 92]	Donald J. Wheeler and David S. Chambers, *Understanding Statistical Process Control*, Second Edition, SPC Press, Knoxville, TN, 1992.
[Wheeler 98]	Donald J. Wheeler and Sheila R. Poling, *Building Continual Improvement: A Guide for Business*, SPC Press, Knoxville, TN, 1998.

Current Author/Maintainer

Anita Carleton, SEI

External Reviewers

Bill Florac, SEI

Modifications

February 28, 2001: Original

URL: http://www.sei.cmu.edu/str/descriptions/spc_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

TAFIM Reference Model

Software Technology Roadmap

Status

Advanced

Note

We recommend Reference Models, Architectures, Implementations- An Overview as prerequisite reading for this technology.

Purpose and Origin

The Technical Architectural Framework for Information Management (TAFIM) reference model was developed by the Defense Information Systems Agency (DISA) to guide the evolution of Department of Defense (DoD) systems, including sustaining base, strategic, and tactical systems, as well as interfaces to weapon systems. Application of the TAFIM reference model is required on most DoD systems [Paige 93]. TAFIM is a set of services, standards, design components, and configurations that are used in design, implementation, and enhancement of information management system architectures. The intent is that the DoD infrastructure will have a common architecture that will, over time, be a fully *flexible* and *interoperable* enterprise. Details on the TAFIM model are available in a seven volume TAFIM document, but are primarily in Volume 3 [TAFIM 94].

Technical Detail

The TAFIM reference model (Figure 27) describes services (functionality) needed within each of the model's components. It contains a set of general principles on how components and component services relate to each other. This model is designed to enhance transition from legacy applications to a distributed environment. TAFIM addresses the following six software components:

1. *Application software*. Application software consists of mission area applications and support applications. Mission area applications may be custom-developed software, commercial-off-the-shelf (COTS) products, or Non-developmental items (NDI). Support applications are building blocks for mission area applications. They manage processing for the communication environment and can be shared by multiple mission and support applications. Common COTS support applications include multimedia, communications, business processing, environment management, database utilities, and engineering support (analysis, design, modeling, development, and simulation) capabilities.
2. *Application platform*. Application platform consists of hardware services and software

services, including operating system, real-time monitoring program, and peripheral drivers. Application software must access platform resources by a request across Application Programming Interfaces (APIs) to ensure integrity and consistency. A platform service may be realized by a single process shared by a group of applications, or by a distributed system with portions of an application operating on separate processors. Application platform services include software engineering, user interface, data management, data interchange, graphic, network, and operating system capabilities.

3. *Application platform cross-area services*. Application platform cross-area services are services that have a direct effect on the operation of one or more of the functional areas. Application platform cross-area services include culturally-related application environments, security, system administration and distributed computing capabilities.
4. *External environment*. The external environment supports system and application interoperability and user and data portability. The external environment interface specifies a complete interface between the application platform and underlying external environment. The external environment includes human-computer interaction, information services, and communication capabilities.
5. *TAFIM application program interface (API)*.The API is the interface between an application and a service that resides on a platform. The API specifies how a service is invoked- without specifying its implementation- so that the implementation may be changed without causing a change in the applications that use that API. The API makes the platform transparent to the application. A platform may be a single computer or a network of hosts, clients, and servers where distributed applications are implemented. A service invoked through an API can reside on the same platform as the requesting application, on a different platform, or on a remote platform. APIs are defined for mission and support applications and platform services. APIs are generally required for platform services such as compilers, window management, data dictionaries, database management systems, communication protocols, and system management utilities.
6. *TAFIM external environment interface*. The TAFIM external environment interface (which could be considered and API) is between the application platform and the external environment. This interface allows the exchange of information. It supports system and application software interoperability. User and data portability are directly provided by the external environment interface.

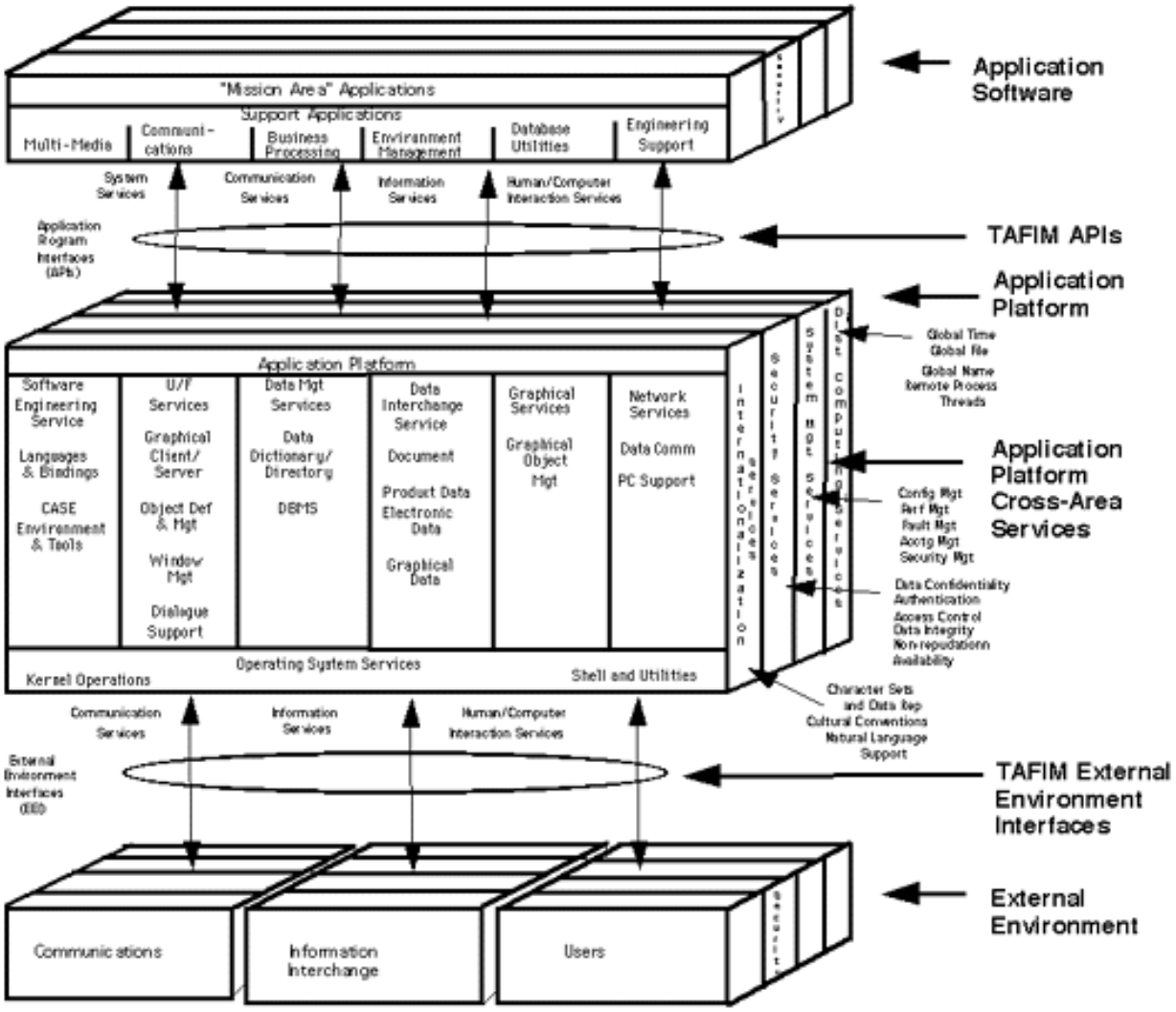


Figure 27: DoD TAFIM Technical Reference Model

Usage Considerations

The TAFIM reference model is applicable to most information systems, including sustaining base, strategic, and tactical systems, as well as interfaces to weapon systems [TAFIM 94]. It is mandatory for use on most DoD programs [Paige 93]. However, systems built using the reference model have been criticized by Rear Adm. John Gauss, the Interoperability Chief at DISA, when speaking on systems in the field in Bosnia: "We have built a bunch of state-of-the-art, open-systems, TAFIM-compliant stove-pipes" [Temin 96]. TAFIM-compliant means that the applicable standards and guidelines are met for the implemented component services. This suggests that even when complying with the TAFIM reference model, problems of interoperability are not necessarily resolved. The Joint Technical Architecture (JTA) provides a set of standards and guidelines for C4I systems, specifically in the area of interoperability, that supersedes TAFIM Volume 7 [JTA 96].

There are TAFIM-compliant software products available for use when implementing a TAFIM-based architecture in areas such as support applications, communication services, business process services, environment management, and engineering services. Additional products exist or are being developed in areas such as user interface, data management, data interchange, graphics, operating systems, internationalization, security system management,

and distributed computing.

Maturity

The latest version of TAFIM, Version 2.0, was published in 1994. DoD organizations and contractors have been applying this set of guidelines to current and future information systems. The Defense Information Infrastructure Common Operating Environment is an implementation of TAFIM. This COE is currently being used by the Global Command and Control System (GCCS) and the Global Combat Support System (GCSS). The Air Force Theater Battle Management Core System (TBMCS) is also required to comply with the TAFIM and use the COE. It may take several years, after multiple new TAFIM-compliant systems are in the field, to determine the effectiveness of the reference model with respect to achieving a common, flexible, and interoperable DoD infrastructure.

Costs and Limitations

The TAFIM reference model does not fully specify components and component connections [Clements 96]. It does not dictate the specific components for implementation. (No reference model prescribes implementation solutions.) TAFIM does provide the guidance necessary to improve commonality among DoD information technical architectures.

One contractor has found that there is no cost difference in using the TAFIM reference model (as compared to any other reference model) when designing and implementing a software architecture. This is based on the fact that application of a reference model is part of the standard design and implementation practice.

Dependencies

The TAFIM reference model is dependent on the evolution of component and service standards that apply specifically to software; it may be affected by computer platforms and network hardware as well.

Alternatives

Under conditions where the TAFIM reference model is not required, an alternative model would be the Reference Model for Frameworks of Software Engineering Environments (known as the ECMA model [ECMA 93]) that is promoted in Europe and used commercially and worldwide. Commercially-available Hewlett-Packard products use this model [HP 96]. Another alternative would be the Common Object Request Broker Architecture (CORBA) if the design called for object-oriented infrastructure .

Complementary Technologies

Open systems (see COTS and Open Systems-An Overview) would be a complementary technology to TAFIM because work done in open system supports the TAFIM goals of achieving interoperable systems.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	TAFIM Reference Model
Application category	Software Architecture Models (AP.2.1.1) Distributed Computing (AP.2.1.2)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1)
Computing reviews category	Distributed Systems (C.2.4) Software Engineering Design (D.2.10)

References and Information Sources

[Clements 96]	Clements, Paul C. & Northrop, Linda M. *Software Architecture: An Executive Overview* (CMU/SEI-96-TR-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
[ECMA 93]	*Reference Model for Frameworks of Software Engineering Environments*, 3rd Edition (NIST Special Publication 500-211/Technical Report ECMA TR/55). Prepared jointly by NIST and the European Computer Manufacturers Association (ECMA). Washington, DC: U.S. Government Printing Office, 1993.
[HP 96]	*Integrated Solutions Catalog for the SoftBench Product Family*. Palo Alto, CA: Hewlett-Packard, 1996.
[JTA 96]	U.S. Department of Defense. *Joint Technical Architecture (JTA)* [online]. Available WWW <URL: http://www-jta.itsi.disa.mil/>(1996).
[Paige 93]	Paige, Emmett. *Selection of Migration Systems* ASD (C3I) Memorandum. Washington, DC: Department of Defense, November 12, 1993.
[TAFIM 94]	U.S. Department Of Defense. *Technical Architecture Framework For Information Management (TAFIM)* Volumes 1-8, Version 2.0. Reston, VA: DISA Center for Architecture, 1994. Also available [online] WWW <URL: http://www-library.itsi.disa.mil/tafim/tafim.html> (1996).
[Temin 96]	Temin, Thomas, ed. "Mishmash at Work (DoD Systems in Bosnia are not Interoperable)." *Government Computer News 15, 7* (April 1996): 28.

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Peter Garrabrant, GTE
Tricia Oberndorf, SEI

Modifications

10 Jan 97 (original)

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/tafim_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Team Software Process (TSP)

Software Technology Roadmap

Status

Complete

Note

Recommended and additional and supplementary reading materials include:
The Team Software Process℠ (TSP℠) (CMU/SEI-2000-TR-023).
The Team Software Process℠ (TSP℠): An Overview and Preliminary Results of Using Disciplined Practices (CMU/SEI-2000-TR-015).
Building High Performance Teams Using Team Software Process℠ (TSP℠) and Personal Software Process℠ (PSP℠) - Home Page http://www.sei.cmu.edu/tsp

Purpose and Origin

Organizations that develop software recognize that controlling their software processes significantly affects their ability to be successful in business. However, organizations still struggle when trying to apply disciplined methods in software process. Historically, this struggle has resulted from a lack of operational procedures for use by teams and individuals in developing software in a disciplined fashion. The Team Software Process (TSP)[1] was designed to provide both a strategy and a set of operational procedures for using disciplined software process methods at the individual and team levels.

Watts Humphrey developed the Personal Software Process (PSP)[2] and the TSP as a follow-up to his work with the Capability Maturity Model (CMM)[3]. The PSP is a defined process for individuals [Humphrey 95] that operationally enacts the concepts and principles prescribed by the CMM. It is the foundation from which the TSP was developed for teams. The TSP represents an operational process for teams that may be used as a strategy for implementing the CMM framework on teams.

Technical Detail

The TSP is a fully defined and measured process that teams can use to plan their work, execute their plans, and continuously improve their software development processes. The TSP process is defined in a series of process

scripts that describe all aspects of project planning and product development. The process includes team role definitions, defined measures, and the postmortem process. Teams using the TSP practice those processes areas that are prescribed by the CMM Maturity Level 5. Their team processes are repeatable, defined, measured, and managed quantitatively. However, it should be noted that the TSP can and has been used successfully by organizations at levels 1, 2, 3, and 5. [McAndrews 00] In fact, the use of the TSP may even help low maturity organizations to have teams quickly start behaving like a level 5 organization. Also, using the TSP will help the organization recognize how the process areas operate at each maturity level.

Within the TSP scripts, there are operational definitions of the measures to be used as part of the process. These measures include basic size (thousands of lines of code [KLOC]), time (minutes and hours), and quality (defects), as well as derived measures for productivity (KLOC/hour), process yield (percentage of defects removed before a particular process phase), and defect densities (defects/KLOC) of finished products. The process establishes how these measures are defined, estimated, collected, reported, and analyzed. The process also makes use of the team's historical data, as well as industry planning and quality guidelines. Tools are available to help facilitate the TSP.

Usage Considerations

A typical software engineering team spends a great deal of time and creative energy struggling with questions concerning goals, team roles, quality, development, management, and multiple other issues. In fact, a team's ability to deal with these issues can affect their success. The TSP provides explicit guidance on how to answer these questions and accomplish the team's objectives. The TSP shows engineering teams how to produce quality products for planned costs and on aggressive schedules. It achieves this by showing teams how to manage their work and by making them owners of their plans and processes. The TSP also helps to accelerate software process improvement.

The TSP has been used with software-only teams and with mixed teams composed of hardware, software, systems, and test professionals. The TSP can be used on teams that typically range in size from 2 to about 150 individuals. The TSP has been used for both new development and enhancement, and on applications ranging from commercial software to embedded real-time systems. It is also applicable in maintenance and support environments.

Maturity

Since the TSP technology is new, it has not yet gained widespread use. It also takes time to obtain data on projects because software projects in industry typically take months or years to complete. Furthermore, because of competition in the software industry, and the resulting sensitivity about sharing data, it can be difficult to persuade organizations to release their data to the public and to participate in studies such as this. Consequently, there are limited data at this time on TSP application. However, there have been a few published results, and they are compelling. [McAndrews 00]

Costs and Limitations

Introducing the TSP into engineering organizations is the principal focus of the TSP effort at the Software Engineering Institute (SEI). The TSP was designed for engineering teams, and its introduction has been initially targeted at teams developing software-intensive products. To support use by industrial teams that include other than software specialties, the SEI has developed an introductory PSP course for professionals who are not software proficient. It has also introduced a series of training and qualification programs so that organizations can obtain their own PSP instructors. In addition, the SEI provides TSP coach training so that organizations can launch and coach their own TSP teams. The SEI has established relationships with a number of transition partners who are qualified to teach the PSP and to coach TSP teams.

Dependencies

The TSP requires careful introduction strategies that include PSP training. The initial reason for developing the TSP was to provide an environment where PSP-trained engineers would find it natural to use disciplined methods. PSP training by itself had not been found sufficient to get engineers to consistently use the methods [Ferguson 97]. There are several reasons why this is the case. First, without training, managers generally do not understand the PSP methods or appreciate their benefits. They then often object to their engineers spending time on planning, doing personal reviews, or gathering and analyzing data. Second, disciplined work is hard to do even with support and coaching. Without such help, long periods of sustained disciplined work are almost impossible. The initial motivation for the TSP design was to address these problems. [Humphrey 00]

Complementary Technologies

The TSP is a stand-alone technology used to produce effective teams. It can be used independent of, and in conjunction with, various development methodologies. TSP, like PSP, is complementary to organizational software process improvements efforts based on the CMM for Software [Paulk 95]. The CMM is an organization-focused process-improvement framework that provides a disciplined, efficient organizational environment for software engineering work. The PSP equips engineers with the personal skills and methods to do high-quality work and participate in organizational process improvement. Of the 18 key process areas in the CMM, PSP covers 12 of the 18, and the TSP covers 16.

The TSP is being developed for a wider range of project applications, including large multi-teams, geographically distributed teams, and functional teams.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Team Software Process
Application category	Detailed Design (AP.1.3.5) Code (AP.1.4.2) Unit Testing (AP.1.4.3.4) Component Testing (AP.1.4.3.5) Reapply Software Life Cycle (AP.1.9.3) Reengineering (AP.1.9.5)
Quality measures category	Reliability (QM.2.1.2) Availability (QM.2.1.1) Maintenance Control (QM.5.1.2.3) Productivity (QM.5.2)
Computing reviews category	Management (D.2.9)

References and Information Sources

[Ferguson 97] Ferguson, P.; Humphrey, W. S.; Khajenoori, S.; Macke, S.; and Matvya, A. "Introducing the Personal Software Process: Three Industry Case Studies." *IEEE Computer* 30 (May 1997): 24-31.

[Ferguson 99] Ferguson, P.; Leman, G.; Perini, P.; Renner, S.; and Seshagiri, G. Software Process Improvement Works! Advanced Information Services Inc. (CMU/SEI-99-TR-027, ADA371804). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1999.

[Humphrey 95] Humphrey, Watts. *A Discipline for Software Engineering.* Reading, MA: Addison-Wesley Publishing Company, 1995.

[Humphrey 00] Humphrey, Watts. *The Team Software Process^{SM} (TSP^{SM})* (CMU/SEI-2000-TR-023). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2000.

[McAndrews 00] McAndrews, Donald R. *The Team Software ProcessSM (TSP^{SM}): An Overview and Preliminary Results of Using Disciplined Practices* (CMU/SEI-2000-TR-015). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2000.

[Musson 99] Musson, R. "The Results of Using the TSP on Small Teams." Proceedings of the 1999 Software Engineering Symposium. (Pittsburgh, Pa., Software Engineering Institute, September 1999).

[Paulk 95] Paulk, Mark C. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley Publishing Company, 1995.

[Vu 00] Vu, J. "Process Improvement in the Boeing Company." Proceedings of the 2000 Software Engineering Process Group (SEPG) Conference. Seattle, WA, Software Engineering Process Group, March 2000.

[Webb 99] Webb, D. and Humphrey, W. Using the TSP on the TaskView Project. *Cross Talk: The Journal of Defense Software Engineering 12, 2* (February 1999).

[Webb 00] Webb, D. Managing Risk with the Team Software Process. *Cross Talk: The Journal of Defense Software Engineering 13, 6* (June 2000).

Current Author/Maintainer

Don McAndrews, SEI
Lauren Heinz, SEI

External Reviewers

Watts Humphrey, SEI

Modifications

1 May 2001 (original)

Footnotes

[1] Personal Software Process and PSP are service marks of Carnegie Mellon University.

[2] Capability Maturity Model and CMM are service marks of Carnegie Mellon University.

[3] Team Software Process and TSP are service marks of Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/tsp_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Three Tier Software Architectures

Software Technology Roadmap

Status

Complete

Note

We recommend Client/Server Software Architectures as prerequisite reading for this technology description.

Purpose and Origin

The three tier software architecture (a.k.a. three layer architectures) emerged in the 1990s to overcome the limitations of the two tier architecture (see Two Tier Software Architectures). The third tier (middle tier server) is between the user interface (client) and the data management (server) components. This middle tier provides process management where business logic and rules are executed and can accommodate hundreds of users (as compared to only 100 users with the two tier architecture) by providing functions such as queuing, application execution, and database staging. The three tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased *performance*, *flexibility*, *maintainability*, *reusability*, and *scalability*, while hiding the complexity of distributed processing from the user. For detailed information on three tier architectures see Schussel and Eckerson. Schussel provides a graphical history of the evolution of client/server architectures [Schussel 96, Eckerson 95].

The three tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. These characteristics have made three layer architectures a popular choice for Internet applications and net-centric information systems.

Technical Detail

A three tier distributed client/server architecture (as shown in Figure 28) includes

a user system interface top tier where user services (such as session, text input, dialog, and display management) reside.

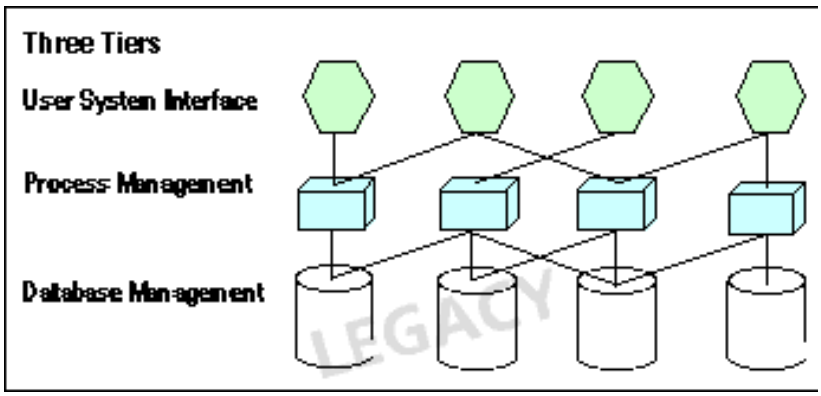


Figure 28: Three tier distributed client/server architecture depiction [Louis 95]

The third tier provides database management functionality and is dedicated to data and file services that can be optimized without using any proprietary database management system languages. The data management component ensures that the data is consistent throughout the distributed environment through the use of features such as data locking, consistency, and replication. It should be noted that connectivity between tiers can be dynamically changed depending upon the user's request for data and services.

The middle tier provides process management services (such as process development, process enactment, process monitoring, and process resourcing) that are shared by multiple applications.

The middle tier server (also referred to as the application server) improves performance, flexibility, maintainability, reusability, and scalability by centralizing process logic. Centralized process logic makes administration and change management easier by localizing system functionality so that changes must only be written once and placed on the middle tier server to be available throughout the systems. With other architectural designs, a change to a function (service) would need to be written into every application [Eckerson 95].

In addition, the middle process management tier controls transactions and asynchronous queuing to ensure reliable completion of transactions [Schussel 96]. The middle tier manages distributed database integrity by the two phase commit process (see Database Two Phase Commit). It provides access to resources based on names instead of locations, and thereby improves scalability and flexibility as system components are added or moved [Edelstein 95].

Sometimes, the middle tier is divided in two or more unit with different functions, in these cases the architecture is often referred as multi layer. This is the case, for example, of some Internet applications. These applications typically have light clients written in HTML and application servers written in C++ or Java, the gap between these two layers is too big to link them together. Instead, there is an intermediate layer (web server) implemented in a scripting language. This

layer receives requests from the Internet clients and generates html using the services provided by the business layer. This additional layer provides further isolation between the application layout and the application logic.

It should be noted that recently, mainframes have been combined as servers in distributed architectures to provide massive storage and improve security (see Distributed/Collaborative Enterprise Architectures).

Usage Considerations

Three tier architectures are used in commercial and military distributed client/server environments in which shared resources, such as heterogeneous databases and processing rules, are required [Edelstein 95]. The three tier architecture will support hundreds of users, making it more scalable than the two tier architecture (see Two Tier Software Architectures) [Schussel 96].

Three tier architectures facilitate software development because each tier can be built and executed on a separate platform, thus making it easier to organize the implementation. Also, three tier architectures readily allow different tiers to be developed in different languages, such as a graphical user interface language or light internet clients (HTML, applets) for the top tier; C, C++, SmallTalk, Basic, Ada 83, or Ada 95 for the middle tier; and SQL for much of the database tier [Edelstein 95].

Migrating a legacy system to a three tier architecture can be done in a manner that is low-risk and cost-effective. This is done by maintaining the old database and process management rules so that the old and new systems will run side by side until each application and data element or object is moved to the new design. This migration might require rebuilding legacy applications with new sets of tools and purchasing additional server platforms and service tools, such as transaction monitors (see Transaction Processing Monitor Technology) and Message-Oriented Middleware. The benefit is that three tier architectures hide the complexity of deploying and supporting underlying services and network communications.

Maturity

Three tier architectures have been used successfully since the early 1990s on thousands of systems of various types throughout the Department of Defense (DoD) and in commercial industry, where distributed information computing in a heterogeneous environment is required. An Air Force system that is evolving from a legacy architecture to a three tier architecture is Theater Battle Management Core System (TBMCS). Multi tier architectures have been widely and successfully applied in some of the biggest Internet servers.

Costs and Limitations

Building three tier architectures is complex work. Programming tools that support the design and deployment of three tier architectures do not yet provide all of the desired services needed to support a distributed computing environment.

A potential problem in designing three tier architectures is that separation of user interface logic, process management logic, and data logic is not always obvious. Some process management logic may appear on all three tiers. The placement of a particular function on a tier should be based on criteria such as the following [Edelstein 95]:

- ease of development and testing
- ease of administration
- scalability of servers
- performance (including both processing and network load)

Dependencies

Database management systems must conform to X/Open systems standards and XA Transaction protocols to ensure distributed database integrity when implementing a heterogeneous database two phase commit.

Alternatives

Two tier client server architectures (see Two Tier Software Architectures) are appropriate alternatives to the three tier architectures under the following circumstances:

- when the number of users is expect to be less than 100
- for non-real-time information processing in non-complex systems that requires minimal operator intervention

Distributed/collaborative enterprise computing (see Distributed/Collaborative Enterprise Architectures) is seen as a viable alternative, particularly if object-oriented technology on an enterprise-wide scale is desired. An enterprise-wide design is comprised of numerous smaller systems or subsystems.

Although three tier architecture has proven sound, the supporting products implementing the architecture are not as mature as other competing technologies. Transaction Monitors (TM) are a valid alternative when reliability and scalability requirements can not be fulfilled with existing multi layer technology. Although TMs don't support modern development paradigms like

Object Orientation (OO) they are still quite useful when massive scalability and robustness is needed.

Complementary Technologies

Complementary technologies to three tier architectures are Object-Oriented Design (to implement decomposable applications), three tier client/server architecture tools, and Database Two Phase Commit processing.

For communication between potentially distributed layers some middleware is needed. This middleware can be a Remote Procedure Call (RPC) mechanism or a Message-Oriented Middleware (MOM), depending on whether synchronous or asynchronous communication is preferred.

The middle tier encapsulates business logic. Some of this logic is application specific but a significant percentage is organization or even domain wide. Domain Engineering and Domain Analysis can be used to capture this inter-application commonality and create a set of assets that can be effectively reused in different application.

It should be noted that recently, mainframes have been combined as servers in distributed architectures to provide massive storage and improve security (see Distributed/Collaborative Enterprise Architectures).

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Three Tier Software Architectures
Application category	Client/Server (AP.2.1.2.1)
Quality measures category	Maintainability (QM.3.1) Scalability (QM.4.3) Reusability (QM.4.4) Reliability (QM.2.1.2)
Computing reviews category	Distributed Systems (C.2.4) Software Engineering Design (D.2.10)

References and Information Sources

[Dickman 95] Dickman, A. "Two-Tier Versus Three-Tier Apps." *Informationweek* 553 (November 13, 1995): 74-80.

[Eckerson 95] Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." *Open Information Systems 10*, 1 (January 1995): 3(20).

[Edelstein 95] Edelstein, Herb. "Unraveling Client Server Architectures." *DBMS 7*, 5 (May 1994): 34(7).

[Gallaugher 96] Gallaugher, J. & Ramanathan, S. "Choosing a Client/Server Architecture. A Comparison of Two-Tier and Three-Tier Systems." *Information Systems Management Magazine 13*, 2 (Spring 1996): 7-13.

[Louis 95] *Louis* [online]. Available WWW <URL: http://www.softis.is> (1995).

[Newell 95] Newell, D.; Jones, O.; & Machura, M. "Interoperable Object Models for Large Scale Distributed Systems," 30-31. *Proceedings. International Seminar on Client/Server Computing*. La Hulpe, Belgium, October 30-31, 1995. London, England: IEE, 1995.

[Schussel 96] Schussel, George. *Client/Server Past, Present, and Future* [online]. Formerly Available WWW <URL: http://news.dci.com/geos/dbsejava.htm> (1995).

Current Author/Maintainer

Darleen Sadoski, GTE
Santiago Comella-Dorda, SEI

External Reviewers

Paul Clements, SEI
Frank Rogers, GTE

Modifications

16 Feb 2000: Inclusion of multi-layer architectures and net-centric systems.

10 Jan 1997 (original)

URL: http://www.sei.cmu.edu/str/descriptions/threetier_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Transaction Processing Monitor Technology

Software Technology Roadmap

Status

Advanced

Note

We recommend Client/Server Software Architectures *as prerequisite reading for this technology description.*

Purpose and Origin

Transaction processing (TP) monitor technology provides the distributed client/server environment the capacity to *efficiently* and *reliably* develop, run, and manage transaction applications.

TP monitor technology controls transaction applications and performs business logic/rules computations and database updates. TP monitor technology emerged 25 years ago when Atlantic Power and Light created an online support environment to share concurrently applications services and information resources with the batch and time sharing operating systems environment. TP monitor technology is used in data management, network access, security systems, delivery order processing, airline reservations, and customer service. Use of TP monitor technology is a cost-effective alternative to upgrading database management systems or platform resources to provide this same functionality. Dickman and Hudson provide more details on TP monitor technology [Dickman 95, Hudson 94].

Technical Detail

TP monitor technology is software that is also referred to as Middleware. It can provide application services to thousands of clients in a distributed client/server environment. TP monitor technology does this by multiplexing client transaction requests (by type) onto a controlled number of processing routines that support particular services. These events are depicted in Figure 37.

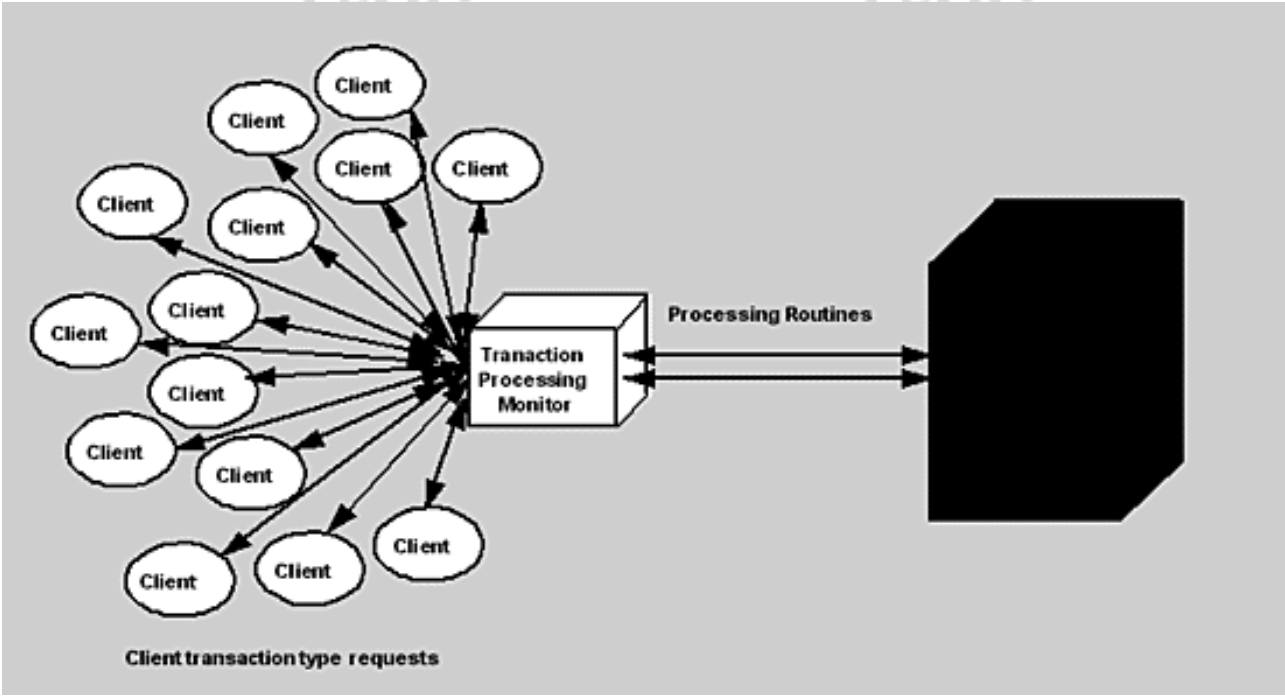


Figure 37: Transaction Processing Monitor Technology

Clients are bound, serviced, and released using stateless servers that minimize overhead. The database sees only the controlled set of processing routines as clients [Dickman 95, Hudson 94].

TP monitor technology maps numerous client requests through application services routines to improve system performance. The TP monitor technology (located as a server) can also take the application transitions logic from the client. This reduces the number of upgrades required by these client platforms. In addition, TP monitor technology includes numerous management features, such as restarting failed processes, dynamic load balancing, and enforcing consistency of distributed data. TP monitor technology is easily scalable by adding more servers to meet growing numbers of users [Dickman 95, Hudson 94].

TP monitor technology is independent of the database architecture. It supports flexible and robust business modeling and encourages modular, *reusable* procedures. TP monitor designs allow Application Programming Interfaces (APIs) to support components such as heterogeneous client libraries, databases and resource managers, and peer-level application systems. TP monitor technology supports architecture *flexibility because* each component in a distributed system is comprised of products that are designed to meet specific functionality, such as graphical user interface builders and database engines [Dickman 95, Hudson 94].

Usage Considerations

Within distributed client/server systems, each client that is supported adds overhead to system resources (such as memory). Responsiveness is improved and system resource overhead is reduced by using TP monitor technology to multiplex many clients onto a much smaller set of application service routines. TP monitor technology provides a highly active system that includes services for delivery order processing, terminal and forms management, data management, network access, authorization, and security.

TP monitor technology supports a number of program-to-program communication models, such as store-and-forward, asynchronous, Remote Procedure Call (RPC), and conversational. This improves interactions among application components. TP monitor technology provides the ability to construct complex business applications from modular, well-defined functional components. Because this technology is well-known and well-defined it should reduce program risk and associated costs [Dickman 95, Hudson 94].

Maturity

TP monitor technology has been used successfully in the field for 25 years. TP monitor technology is used for delivery order processing, hotel and airline reservations, electronic fund transfers, security trading, and manufacturing resource planning and control. It improves batch and time-sharing application effectiveness by creating online support to share application services and information resources [Dickman 95, Hudson 94].

Costs and Limitations

TP monitor technology makes database processing cost-effective for online applications. Spending relatively little money on TP monitor technology can result in significant savings compared to the resources required to improve database or platform resources to provide the same functionality [Dickman 95].

A limitation to TP technology is that the implementation code is usually written in a lower-level language (such as COBOL), and is not yet widely available in the popular visual toolsets [Schussel 96].

Alternatives

A variation of TP monitor technology is session based technology. In the TP monitor technology, transactions from the client are treated as messages. In the session based technology, a single server provides both database and transaction services. In session based technology, the server must be aware of clients in advance to maintain each client's processing thread. The session server must constantly send messages to the client (even when work is not being done in the client) to ensure that the client is still alive. Session based architectures are not as scalable because of the adverse effect on network performance as the number of clients grow.

Another alternative to TP monitor technology is remote data access (RDA). The RDA centers the application in a client computer, communicating with back-end database servers. Clients can be network-intensive, but scalability is limited.

A third alternative to TP monitor technology is the database server approach, which provides functions (usually specific to the database) and is architecturally locked to the specific database system [Dickman 95, Hudson 94].

Complementary Technologies

Complementary technologies include mainframe client/server software architectures (see Mainframe Server Software Architectures) and Three Tier Software Architectures; in both cases

the TP monitor technology could server as the middle tier.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Transaction Processing Monitor Technology
Application category	Client/Server (AP.2.1.2.1) Client/Server Communication (AP.2.2.1)
Quality measures category	Efficiency/ Resource Utilization (QM.2.2) Reusability (QM.4.4) Maintainability (QM.3.1)
Computing reviews category	Distributed Systems (C.2.4)

References and Information Sources

[Dickman 95]	Dickman, A. "Two-Tier Versus Three-Tier Apps." *Informationweek* 553 (November 13, 1995): 74-80.
[Hudson 94]	Hudson, D. & Johnson, J. *Client-Server Goes Business Critical*. Dennis, MA: The Standish Group International, 1994.
[Schussel 96]	Schussel, George. *Client/Server Past, Present, and Future* [online]. Available WWW <URL: http://www.dciexpo.com/geos/> (1995).
[TP 96]	*TP Lite vs. TP Heavy* [online]. Available WWW <URL: http://www.byte.com/art/9504/sec11/art4.htm> (1996).

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

David Altieri, GTE

Modifications

10 Jan 97 (original)

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Trusted Operating Systems

Software Technology Roadmap

Status

Advanced

Note

We recommend Computer System Security--An Overview as prerequisite reading for this technology description.

Purpose and Origin

Trusted operating systems provide the basic security mechanisms and services that allow a computer system to protect, distinguish, and separate classified data. Trusted operating systems have been developed since the early 1980s and began to receive National Security Agency (NSA) evaluation in 1984.

Technical Detail

Trusted operating systems lower the security risk of implementing a system that processes classified data. Trusted operating systems implement security policies and accountability mechanisms in an operating system package. A security policy is the rules and practices that determine how sensitive information is managed, protected, and distributed [Abrams 95]. Accountability mechanisms are the means of identifying and tracing who has had access to what data on the system so they can be held accountable for their actions.

Trusted operating systems are evaluated by the NSA National Computer Security Center (NCSC) against a series of six requirements-level classes listed in the table below. C1 systems have basic capabilities. A1 systems provide the most capability. The higher the rating level is, the wider the range of classified data is that may be processed.

Table 10 below shows the NCSC Evaluation Criteria Classes.

Table 10: NCSC Evaluation Criteria Classes

Class	Title	Number of Approved Operating Systems in this Class [TPEP 96]
A1	Verified Design	0
B3	Security Domains	1
B2	Structured Protection	1
B1	Labeled Security Protection	7
C2	Controlled Access Protection	5
C1	Discretionary Security Protection	No Longer Evaluated

A low level (C1 and C2) system provides limited discretionary access controls and identification and authentication mechanisms. Discretionary access controls identify who can have access to system data based on the need to know. Mandatory access controls identify who or what process can have access to data based on the requester having formal clearance for the security level of the data. A low-level system is used when the system only needs to be protected against human error and it is unlikely that a malicious user can gain access to the system.

A higher level (B2, B3, and A1) system provides complete mandatory and discretionary access control, thorough security identification of data devices, rigid control of transfer of data and access to devices, and complete auditing of access to the system and data. These higher level systems are used when the system must be protected against a malicious user's abuse of authority, direct probing, and human error [Abrams 95].

The portion of the trusted operating system that grants requesters access to data and records the action is frequently called the reference monitor because it refers to an authorization database to determine if access should be granted. Higher level trusted operating systems are used in MLS hosts and compartmented mode workstations (see Computer System Security- an Overview for overview information).

Usage Considerations

Trusted operating systems must be used to implement multi-level security systems and to build security guards that allow systems of different security levels to be connected to exchange data. Use of a trusted operating system may be the only way that a system can be networked with other high security

systems. Trusted operating systems may be required if a C4I system processes intelligence data and provides data to war fighters. Department of Defense (DoD) security regulations define what evaluation criteria must be satisfied for a multi-level system based on the lowest and highest classification of the data in a system and the clearance level of the users of the system. Using an NCSC-evaluated system reduces accreditation cost and risk. The security officer identified as the Designated Approving Authority (DAA) for secure computer systems has the responsibility and authority to review and approve the systems to process classified information. The DAA will require analysis and tests of the system to assure that it will operate securely. The DAA can accept the NCSC evaluation of a system rather than generating the data. For a B3 or A1 system, that can represent a savings of 1 to 2 years in schedule and the operating system will provide a proven set of functions.

Maturity

This technology has been implemented by several vendors for commercial-off-the-shelf (COTS) use in secure systems. As of September 1996, the NCSC Evaluated Product List indicated that fourteen operating systems have been evaluated as level C2, B1,B2, and B3 systems in the last three years [TPEP 96]. The number of operating systems evaluated by class (excluding evaluations of updated versions of operating systems) is included in the table. Use of one of the approved trusted operating systems can result in substantial cost and schedule reductions for a system development effort and provide assurance that the system can be operated securely.

Costs and Limitations

The heavy access control and accounting associated with high security systems can affect system performance; as such, higher performance processors, I/O, and interfaces may be required. Trusted operating systems have unique interfaces and operating controls that require special security knowledge to use and operate. Frequently COTS products that operate satisfactorily with a standard operating system must be replaced or augmented to operate with a trusted operating system.

Dependencies

Trusted operating systems at B2 and above enable the development of system interoperability for systems at different security levels and allow applications to perform data fusion. They are dependent on a trusted computing base that provides secure data paths and protected memory.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Trusted Operating Systems
Application category	Trusted Operating Systems (AP.2.4.1)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating System Security and Protection (D.4.6) Computer-Communications Network Security Protection (C.2.0)

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Russel 91] Russel, Deborah & Gangemi, G.T. Sr. *Computer Security Basics.* Sebastopol, CA: O'Reilly & Associates, Inc., 1991.

[TPEP 96] *Trusted Product Evaluation Program Evaluated Product List* [online]. Available WWW <URL: http://www.radium.ncsc.mil/tpep/index.html> (1996).

[White 96] White, Gregory B.; Fisch, Eric A.; & Pooch, Udo W. *Computer System and Network Security.* Boca Raton, FL: CRC Press, 1996.

Current Author/Maintainer

Tom Mills, Lockheed Martin

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/trusted_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Two Tier Software Architectures

Software Technology Roadmap

Status

Complete

Note

We recommend Client/Server Software Architectures*, as prerequisite reading for this technology description.*

Purpose and Origin

Two tier software architectures were developed in the 1980s from the file server software architecture design. The two tier architecture is intended to improve *usability* by supporting a forms-based, user-friendly interface. The two tier architecture improves *scalability* by accommodating up to 100 users (file server architectures only accommodate a dozen users), and improves *flexibility* by allowing data to be shared, usually within a homogeneous environment [Schussel 96]. The two tier architecture requires minimal operator intervention, and is frequently used in non-complex, non-time critical information processing systems. Detailed readings on two tier architectures can be found in Schussel and Edelstein [Schussel 96, Edelstein 94].

Technical Detail

Two tier architectures consist of three components distributed in two layers: client (requester of services) and server (provider of services). The three components are

1. User System Interface (such as session, text input, dialog, and display management services)
2. Processing Management (such as process development, process enactment, process monitoring, and process resource services)
3. Database Management (such as data and file services)

The two tier design allocates the user system interface exclusively to the client. It places database management on the server and splits the processing management between client and server, creating two layers. Figure 38 depicts the two tier software architecture.

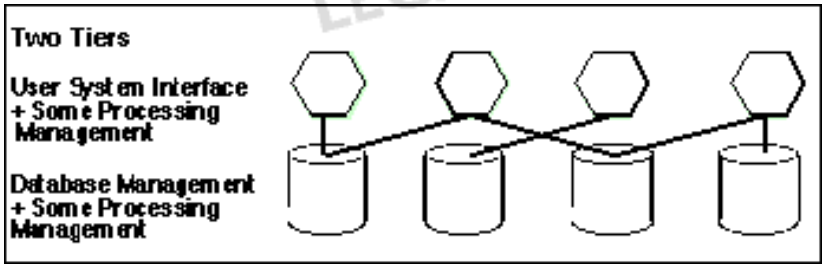


Figure 38: Two Tier Client Server Architecture Design [Louis 95]

In general, the user system interface client invokes services from the database management server. In many two tier designs, most of the application portion of processing is in the client environment. The database management server usually provides the portion of the processing related to accessing data (often implemented in store procedures). Clients commonly communicate with the server through SQL statements or a call-level interface. It should be noted that connectivity between tiers can be dynamically changed depending upon the user's request for data and services.

As compared to the file server software architecture (that also supports distributed systems), the two tier architecture improves flexibility and scalability by allocating the two tiers over the computer network. The two tier improves usability (compared to the file sever software architecture) because it makes it easier to provide a customized user system interface.

It is possible for a server to function as a client to a different server- in a hierarchical client/server architecture. This is known as a chained two tier architecture design.

Usage Considerations

Two tier software architectures are used extensively in non-time critical information processing where management and operations of the system are not complex. This design is used frequently in decision support systems where the transaction load is light. Two tier software architectures require minimal operator intervention. The two tier architecture works well in relatively homogeneous environments with processing rules (business rules) that do not change very often and when workgroup size is expected to be fewer than 100 users, such as in small businesses.

Maturity

Two tier client/server architectures have been built and fielded since the middle to late 1980s. The design is well known and used throughout industry. Two tier architecture development was enhanced by fourth generation languages.

Costs and Limitations

Scalability. The two tier design will scale-up to service 100 users on a network.

It appears that beyond this number of users, the performance capacity is exceeded. This is because the client and server exchange "keep alive" messages continuously, even when no work is being done, thereby saturating the network [Schussel 96].

Implementing business logic in stored procedures can limit scalability because as more application logic is moved to the database management server, the need for processing power grows. Each client uses the server to execute some part of its application code, and this will ultimately reduce the number of users that can be accommodated.

Interoperability. The two tier architecture limits interoperability by using stored procedures to implement complex processing logic (such as managing distributed database integrity) because stored procedures are normally implemented using a commercial database management system's proprietary language. This means that to change or interoperate with more than one type of database management system, applications may need to be rewritten. Moreover, database management system's proprietary languages are generally not as capable as standard programming languages in that they do not provide a robust programming environment with testing and debugging, version control, and library management capabilities.

System administration and configuration. Two tier architectures can be difficult to administer and maintain because when applications reside on the client, every upgrade must be delivered, installed, and tested on each client. The typical lack of uniformity in the client configurations and lack of control over subsequent configuration changes increase administrative workload.

Batch jobs. The two tiered architecture is not effective running batch programs. The client is typically tied up until the batch job finishes, even if the job executes on the server; thus, the batch job and client users are negatively affected [Edelstein 94].

Dependencies

Developing a two tier client/server architecture following an object-oriented methodology would be dependent on the CORBA standards for design implementation. See Common Object Request Broker Architecture.

Alternatives

Possible alternatives for two tier client server architectures are

- the three-tier architecture (see Three Tier Software Architectures) if there is a requirement to accommodate greater than 100 users
- distributed/collaborative architectures (see Distributed/Collaborative Enterprise Architectures) if there is a requirement to design on an enterprise-wide scale. An enterprise-wide design is comprised of numerous smaller systems or subsystems.

When preparing a two tier architecture for possible migration to an alternative three tier architecture, the following five steps will make the transition less costly and of lower risk [Dickman 95]:

1. Eliminate application diversity by ensuring a common, cross-hardware library and development tools.
2. Develop smaller, more comparable service elements, and allow access through clearly-defined interfaces.
3. Use an Interface Definition Language (IDL) to model service interfaces and build applications using header files generated when compiled.
4. Place service elements into separate directories or files in the source code.
5. Increase flexibility in distributed functionality by inserting service elements into Dynamic Linked Libraries (DLLs) so that they do not need to be complied into programs.

Complementary Technologies

Complementary technologies for two tier architectures are CASE (computer-aided software engineering) tools because they facilitate two tier architecture development, and open systems (see COTS and Open Systems-An Overview) because they facilitate developing architectures that improve scalability and flexibility.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Two Tier Software Architectures
Application category	Client/Server (AP.2.1.2.1)
Quality measures category	Usability (QM.2.3) Maintainability (QM.3.1) Scalability (QM.4.3)
Computing reviews category	Distributed Systems (C.2.4) Software Engineering Design (D.2.10)

References and Information Sources

[Dickman 95] Dickman, A. "Two-Tier Versus Three-Tier Apps." *Informationweek* 553 (November 13, 1995): 74-80.

[Edelstein 94] Edelstein, Herb. "Unraveling Client/Server Architecture." *DBMS 7*, 5 (May 1994): 34(7).

[Gallaugher 96] Gallaugher, J. & Ramanathan, S. "Choosing a Client/Server Architecture. A Comparison of Two-Tier and Three-Tier Systems." *Information Systems Management Magazine 13*, 2 (Spring 1996): 7-13.

[Louis 95] *Louis* [online]. Available WWW <URL: http://www.softis.is> (1995).

[Newell 95] Newell, D.; Jones, O.; & Machura, M. "Interoperable Object Models for Large Scale Distributed Systems," 30-31. *Proceedings. International Seminar on Client/Server Computing*. La Hulpe, Belgium, October 30-31, 1995. London, England: IEE, 1995.

[Schussel 96] Schussel, George. *Client/Server Past, Present, and Future* [online]. Available WWW <URL: http://www.dciexpo.com/geos/> (1995).

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Paul Clements, SEI
Frank Rogers, GTE

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/twotier_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections

- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Virus Detection

Software Technology Roadmap

Status

Advanced

Note

We recommend Computer System Security- an Overview, as prerequisite reading for this technology description.

Purpose and Origin

Technologies for Computer System Security in C4I Systems (see Computer System Security- an Overview) introduced virus detection software as one of the system security mechanisms included in Intranets used to support C4I systems. Viruses are malicious segments of code, inserted into legitimate programs, that execute when the legitimate program is executed. The primary characteristic of a virus is that it replicates itself when it is executed and inserts the replica into another program which will replicate the virus again when it executes. A computer is said to be infected if it contains a virus. Detecting that a computer is infected is the process of virus detection. Viruses have existed since the early 1980s and programs to detect them have been developed since then [Denning 90].

Technical Detail

Since viruses are executable code, they are written for a particular processor. They have been written for mainframes, for UNIX machines, and for personal computers (IBM PC compatibles and Apple Macintoshes). By far the most viruses have been developed to attack 80x86-based IBM PC compatible computers. By 1996, there have been over 2000 kinds of viruses developed that attack IBM PC compatible computers. The IBM PC compatible is a frequent target of viruses because there are so many of that type of computer in use and the operating system (DOS and Windows) has no provision to prevent code from being modified. A few viruses, written using word processing or spreadsheet macros, infect any processor that runs the word processor or spreadsheet program that can interpret those macros. There were some early, much publicized, viruses on UNIX machines, but they are rare. The 1988 Morris Worm was an early example of malicious code that attacked UNIX machines [Spafford

88]. Viruses are hard to write because they require detailed knowledge of how the operating system works; there are much easier ways to damage or copy information on a UNIX computer. There have been a few mainframe viruses but they are also rare because mainframe operating systems make it difficult for a program to gain access to and modify other programs.

Within some viruses is a portion of code called the payload. The payload is designed to do something malicious such as corrupt files, display a message on the screen, or prevent the computer from booting. When the virus executes or at some future execution after a trigger condition has been met, the virus will execute the payload. A favorite trigger condition is the occurrence of a particular date, such as Friday the 13th. A virus still causes harm, even if it does not contain a payload, by consuming processor and storage resources as it replicates itself.

The two general types of PC viruses are boot-record infectors and program file infectors. The type is determined by where the virus code copy is written when it is replicated.

Boot-record infectors, also called system infectors, infect the boot records on hard disks and floppy disks. When the system is booted, they are loaded into memory. They may execute and replicate themselves every time a disk is loaded. Once a hard disk boot record is infected the virus will be loaded into memory each time the system is booted from the hard disk.

The program file infectors attach their replicas to program file (.EXE or .COM files) hosts on disk whenever the virus is executed. When the host is executed the virus replicates itself again. When the virus is added to a file it makes the file larger. In order to not cause an obvious growth in a file, viruses include a signature pattern in the copy that it can recognize so that it will not add to a file again if the virus is there already.

There are three basic types of virus detection software:

- virus scanner
- activity monitor
- change detection

Virus scanner software looks for the virus signature in memory or in program files and looks for code in the boot record that is not boot code. Once suspicious code is found, a message is displayed to the operator that the system is infected. Some virus scanners have the capability to remove viruses as well as to detect them.

Activity monitors are memory resident programs that watch for suspicious activity such as a program other than the operating system trying to format disks, delete an executable file, or change the file allocation table on a disk. They also may look for programs trying to go memory resident, scanning for other program files, or trying to modify their own code [Slade 96b].

Change detection software scans the executable program files in the system

before a system is used and records vital statistics about each program, such as program file length or a calculated CRC or checksum. After the system is in operation, the change detection software periodically scans the program files looking for changes compared to the pre-stored data. These changes could have been caused by a virus.

Usage Considerations

Virus scanners are executed periodically, when the system is started up, or whenever a disk is initially put into the system. When new software (commercial, freeware, or downloaded) is added to the system, it should be checked with a virus scanner before the new software is executed to identify known viruses if they are present. Although virus scanners are very useful in finding known viruses they will not detect new kinds of viruses. They therefore must be updated frequently to include the "signatures" of new viruses.

Activity monitors are more likely to find new types of viruses than virus scanners since activity monitors are not limited to finding a known bit pattern in memory or on disk. Activity monitors have considerable performance overhead since they must be constantly scanning for unusual activity. Activity monitors also must be incorporated into software change processes so that its baseline of "correct" software files can be maintained.

Of the three types of virus detection software, change detection software has the best chance of detecting current and future virus types but is most likely to produce false alarms [Slade 96b]. The database for change detection software must be updated every time system files or executable program files are updated. This adds maintenance overhead to the system if the system is frequently modified.

Maturity

More than 100 virus detection products are listed on the National Institute of Standards and Technology (NIST) list of products reviewed [Slade 96a]. Most of those products are virus scanners. Virus scanners are also the most rapidly changing as they must be updated to check for new virus "signatures" as new viruses are identified. The challenge to virus detection product vendors is in the constant race to keep up with the host of smart computer hackers and malicious software developers creating new strains of viruses.

Costs and Limitations

Effective use of virus detection software requires system administrators familiar with virus types and their mode of attack, the operation of the virus detection software, the ability to evaluate the virus detection program output, and the ability to recognize a true attack versus a false alarm. This requires knowledge of the system and its normal operation, training in the use of the virus detection software, and frequent retraining as the virus detection software is routinely updated.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Virus Detection
Application category	Information Security (AP.2.4)
Quality measures category	Security (QM.2.1.5) Denial of Service (QM.2.1.4.1.3)
Computing reviews category	Operating Systems Security and Protection (D.4.6) Security and Protection (K.6.5)

References and Information Sources

[Abrams 95]	Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.
[Denning 90]	Denning, Peter J. *Computers Under Attack Intruders, Worms and Viruses*. New York, NY: ACM Press, 1990.
[Garfinkel 96]	Garfinkel, Simson & Spafford, Gene. *Practical UNIX and Internet Security* Second Edition. Sebastopol, CA: O'Reilly & Associates, Inc., 1996.
[Russel 91]	Russel, Deborah & Gangemi, G.T. Sr. *Computer Security Basics.* Sebastopol, CA: O'Reilly & Associates, Inc., 1991.
[Slade 96a]	Slade, Robert. *Quick Reference Antiviral Review Chart* [online]. Available WWW <URL: http://csrc.ncsl.nist.gov/virus/quickref.rvw> (1996).
[Slade 96b]	Slade, Robert. *Reviewing Anti-virus Products* [online]. Available WWW <URL: http://www.bocklabs.wisc.edu/~janda/sladerev.html> (1996).
[Spafford 88]	Spafford, Eugene H. *The Internet Worm Program: An Analysis* (CSD-TR-823). West Lafayette, IN: Purdue University, 1988.

Current Author/Maintainer

Tom Mills, Loral

Modifications

10 Jan 97 (original)

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Maintainability Index Technique for Measuring Program Maintainability

Software Technology Roadmap

Status

Complete

Purpose and Origin

Quantitative measurement of an operational system's *maintainability* is desirable both as an instantaneous measure and as a predictor of maintainability over time. Efforts to measure and track maintainability are intended to help reduce or reverse a system's tendency toward "code entropy" or degraded integrity, and to indicate when it becomes cheaper and/or less risky to rewrite the code than to change it. *Software Maintainability Metrics Models in Practice* is the latest report from an ongoing, multi-year joint effort (involving the Software Engineering Test Laboratory of the University of Idaho, the Idaho National Engineering Laboratory, Hewlett-Packard, and other companies) to quantify maintainability via a Maintainability Index (MI) [Welker 95]. Measurement and use of the MI is a process technology, facilitated by simple tools, that in implementation becomes part of the overall development or maintenance process. These efforts also indicate that MI measurement applied during software development can help reduce lifecycle costs. The developer can track and control the MI of code as it is developed, and then supply the measurement as part of code delivery to aid in the transition to maintenance.

Other studies to define code maintainability in various environments have been done [Peercy 81, Bennett 93], but the set of reports leading to the MI measurement technique offered by Welker [Welker 95] describes a method that appears to be very applicable to today's Department of Defense (DoD) systems.

Technical Detail

The literature of at least the last ten years shows that there have been several efforts to characterize and quantify software maintainability; Maintenance of Operational Systems--An Overview provides a broad overview of software maintenance issues. In this specific technology, a program's maintainability is calculated using a combination of widely-used and commonly-available measures to form a Maintainability Index (MI). The basic MI of a set of programs is a polynomial of the following form (all are based on average-per-code-module measurement):

171 - 5.2 * ln(aveV) - 0.23 * aveV(g') - 16.2 * ln (aveLOC) + 50 * sin (sqrt(2.4 * perCM))

The coefficients are derived from actual usage (see Usage Considerations). The terms are defined as follows:

aveV = average Halstead Volume V per module (see Halstead Complexity Measures)

aveV(g') = average extended cyclomatic complexity per module (see Cyclomatic Complexity)

aveLOC = the average count of lines of code (LOC) per module; and, optionally

perCM = average percent of lines of comments per module

Oman develops the MI equation forms and their rationale [Oman 92a]; the Oman study indicates that the above metrics are good and sufficient predictors of maintainability. Oman builds further on this work using a modification of the MI and describing how it was calibrated for a specific large suite of industrial-use operational code [Oman 94]. Oman describes a prototype tool that was developed specifically to support capture and use of maintainability measures for Pascal and C [Oman 91]. The aggregate strength of this work and the underlying simplicity of the concept make the MI technique potentially very useful for operational Department of Defense (DoD) systems.

Usage Considerations

Calibration of the equations. The coefficients shown in the equation are the result of calibration using data from numerous software systems being maintained by Hewlett-Packard. Detailed descriptions of how the MI equation was calibrated and used appear in Coleman, Pearse, and Welker [Coleman 94, Coleman, 95, Pearse 95, Welker 95]. The authors claim that follow-on efforts show that this form of the MI equation generally fits other industrial-sized software systems [Oman 94 and Welker 95], and the breadth of the work tends to support this claim. It is advisable to test the coefficients for proper fit with each major system to which the MI is applied.

Effects from comments in code. The user must analyze comment content and quality in the specific system to decide whether the comment term perCM is useful.

Ways of using MI

1. The system can be checked periodically for maintainability, which is also a way of calibrating the equations.

2. It can be integrated into a development effort to screen code quality as it is being built and modified; this could yield potentially significant life cycle cost savings.
3. It can be used to drive maintenance activities by evaluating modules either selectively or globally to find high-risk code.
4. MI can be used to compare or evaluate systems: Comparing the MIs of a known-quality system and a third-party system can provide key information in a make-or-buy decision.

Example of usage. Welker relates how a module containing a routine with some "very ugly" code was assessed as unmaintainable, when expressed in terms of the MI (note that just quantifying the problem is a step forward) [Welker 95]. The module was first redesigned, and then functionally enhanced. The measured results are shown in Table 7:

Table 7: Measured Results

Measure	Initial Code		Restructured Code		After Enhancement	
Code Unit	Routine	Module	Routine	Module	Routine	Module
MI (larger MI = more maintainable)	6.47	33.55	39.93	70.13	37.62	69.60
Halstead Effort[1]	2,216,499	2,233,072	182,216	480,261	201,429	499,474
Extended Cyclomatic Complexity[2]	45	49	18	64	21	67
Lines of Code	622	663	196	732	212	748

[1] Halstead Effort, rather than Halstead Volume, was used in this case study. See Halstead Complexity Measures for more information on both these measures. Generally, the lower a program's measure of effort, the simpler a change to the program will be (because Halstead measures are weighted toward measuring computational complexity, not all programs will behave this way).

[2] Note that a low Cyclomatic Complexity is generally indicative of a lower risk, hence more maintainable, program. In this case, restructuring increased the module complexity slightly (from 49 to 64), but reduced the "ugly" routine's complexity significantly. In both, the subsequent enhancement drove the complexity slightly higher.

If the enhancement had been made without first doing the restructuring, these figures indicate the change would have been much more risky.

Coleman, Pearse, and Welker provide detailed descriptions of how MI was calibrated and used at Hewlett-Packard [Coleman 94, Coleman 95, Pearse 95, Welker 95].

Maturity

Oman tested the MI approach by using production operational code containing around 50 KLOC to determine the metric parameters, and by checking the results against subjective data gathered using the 1989 AFOTEC maintainability evaluation questionnaire [AFOTEC 89, Oman 94]. Other production code of about half that size was used to check the results, with apparent consistency.

Welker applied the results to analyses of a US Air Force (USAF) system, the Improved Many-On-Many (IMOM) electronic combat modeling system. The original IMOM (in FORTRAN) was translated to C and the C version was later reengineered into Ada. The maintainability of both newer versions was measured over time using the MI approach [Welker 95]. Results were as follows:

- The *reengineered* version's MI was more than twice as high as the original code (larger MI = more maintainable), and declined only slightly over time (note that the original code was not measured over time for maintainability, so change in its MI could not be measured).
- The *translated* baseline's MI was *not* significantly different from the original. This is of special interest to those considering translation, because one of the primary objectives of translation is to reduce future maintenance costs. There was also evidence that the MI of translated code deteriorates more quickly than reengineered code.

Costs and Limitations

Calculating the MI is generally simple and straightforward, given that several commercially-available programming environments contain utilities to count code lines, comment lines, and even Cyclomatic Complexity. Other than the tool described in Oman [Oman 91], tools to calculate Halstead Complexity Measures are less common because the measure is not used as widely. However, once conventions for the counting have been established, it is generally not difficult to write language-specific code scanners to count the Halstead components (operators and operands) and calculate the E and V measures. In relating that removal of unused code in a single module did not affect the MI, Pearse highlights the fact that MI is a system measurement; its parameters are average values [Pearse 95]. However, measuring the MI of individual modules is useful because changes in either structural or computational complexity are reflected in a module's MI. A product/process measurement program not already gathering the metrics used in MI could find them useful additions. Those metrics already

being gathered may be useful in constructing a custom MI for the system. However, it would be advisable to consult the references for their findings on the effectiveness of metrics, other than Halstead E and V and cyclomatic complexity, in determining maintainability.

Dependencies

The MI method depends on the use of Cyclomatic Complexity and Halstead Complexity Measures. To realize the full benefit of MI, the maintenance environment must allow the rewriting of a module when it becomes measurably unmaintainable. The point of measuring the MI is to identify risk; when unacceptably risky code is identified, it should be rewritten.

Alternatives

The process described by Sittenauer is designed to assist in deciding whether or not to reengineer a system [Sittenauer 92]. There are also many research and analytic efforts that deal with maintainability as a function of program structure, design, and content, but none was found that was as clearly appropriate as MI to current DoD systems in the lifecycle phases described in Maintenance of Operational Systems--An Overview.

Complementary Technologies

The test in Sittenauer is meant to verify generally the condition of a system, and would be useful as a periodic check of a software system and to compare to the MI [Sittenauer 92].

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Maintainability Index Technique for Measuring Program Maintainability
Application category	Debugger (AP.1.4.2.4) Test (AP.1.4.3) Unit Testing (AP.1.4.3.4) Component Testing (AP.1.4.3.5) Reapply Software Life Cycle (AP.1.9.3) Reengineering (AP.1.9.5)

Quality measures category	Maintainability (QM.3.1) Testability (QM.1.4.1) Understandability (QM.3.2)
Computing reviews category	Software Engineering Distribution and Maintenance (D.2.7) Software Engineering Metrics (D.2.8) Complexity Classes (F.1.3) Tradeoffs Among Complexity Measures (F.2.3)

References and Information Sources

[AFOTEC 89] *Software Maintainability Evaluation Guide* 800-2, Volume 3. Kirtland AFB, NM: HQ Air Force Operational Test and Evaluation Center (AFOTEC), 1989.

[Ash 94] Ash, Dan, et al. "Using Software Maintainability Models to Track Code Health," 154-160. *Proceedings of the International Conference on Software Maintenance*. Victoria, BC, Canada, September 19-23, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[Bennett 93] Bennett, Brad & Satterthwaite, Paul. "A Maintainability Measure of Embedded Software," 560-565. *Proceedings of the IEEE 1993 National Aerospace and Electronics Conference*. Dayton, OH, May 24-28, 1993. New York, NY: IEEE, 1993.

[Coleman 94] Coleman, Don, et al. "Using Metrics to Evaluate Software System Maintainability." *Computer 27,* 8 (August 1994): 44-49.

[Coleman 95] Coleman, Don; Lowther, Bruce; & Oman, Paul. "The Application of Software Maintainability Models in Industrial Software Systems." *Journal of Systems Software 29,* 1 (April 1995): 3-16.

[Oman 91] Oman, P. *HP-MAS: A Tool for Software Maintainability, Software Engineering* (#91-08-TR). Moscow, ID: Test Laboratory, University of Idaho, 1991.

[Oman 92a] Oman, P. & Hagemeister, J. *Construction and Validation of Polynomials for Predicting Software Maintainability* (92-01TR). Moscow, ID: Software Engineering Test Lab, University of Idaho, 1992.

[Oman 92b] Oman, P. & Hagemeister, J. "Metrics for Assessing a Software System's Maintainability," 337-344. *Conference on Software Maintenance 1992*. Orlando, FL, November 9-12, 1992. Los Alamitos, CA: IEEE Computer Society Press, 1992.

[Oman 94] Oman, P. & Hagemeister, J. "Constructing and Testing of Polynomials Predicting Software Maintainability." *Journal of Systems and Software 24,* 3 (March 1994): 251-266.

[Pearse 95] Pearse, Troy & Oman, Paul. "Maintainability Measurements on Industrial Source Code Maintenance Activities," 295-303. *Proceedings. of the International Conference on Software Maintenance*. Opio, France, October 17-20, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Peercy 81] Peercy, David E. "A Software Maintainability Evaluation Methodology." *Transactions on Software Engineering 7*, 7 (July 1981): 343-351.

[Sittenauer 92] Sittenauer, Chris & Olsem, Mike. "Time to Reengineer?" *Crosstalk, Journal of Defense Software Engineering* 32 (March 1992): 7-10.

[Welker 95] Welker, Kurt D. & Oman, Paul W. "Software Maintainability Metrics Models in Practice." *Crosstalk, Journal of Defense Software Engineering 8*, 11 (November/December 1995): 19-23.

[Zhuo 93] Zhuo, Fang, et al. "Constructing and Testing Software Maintainability Assessment Models," 61-70. *Proceedings of the First International Software Metrics Symposium*. Baltimore, MD, May 21-22, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

Current Author/Maintainer

Edmond VanDoren, Kaman Sciences, Colorado Springs

External Reviewers

Paul W. Oman, Ph.D., Computer Science Department, University of Idaho, Moscow, ID
Kurt Welker, Lockheed Martin, Idaho Falls, ID

Modifications

10 Jan 97 (original)
12 Mar 02 Correction of Maintainability Index (MI) formula

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/mitmpm_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

About the Taxonomies

Software Technology Roadmap

Overview and Purpose

Some readers may not desire to read all of the technology descriptions or may not have a specific technology in mind when visiting this Web site. Instead a reader might be concerned about or interested in a particular software quality measure, a phase of the development process, or an operational function.

With this in mind, we created two taxonomies that serve as directories into the technology descriptions. This method is an effective way to lead readers to a set of possible technologies that address their software problem area. Each software technology description has been categorized into the following two taxonomies:

- **Application.** This taxonomy categorizes technologies by how they might be used in operational systems. A technology can fall into one of two major categories. It can be used to support an operational system or it can be used in an operational system.

- **Quality Measures.** This taxonomy categorizes technologies by the software quality characteristics or attributes that they influence, such as maintainability, expandability, reliability, trustworthiness, robustness, and cost of ownership.

The taxonomies serve other purposes as well. A taxonomy implies a hierarchical relationship of terms which are used for classifying items in a particular domain. It is this hierarchical relationship that we wanted to capture for the reader with the hope that each taxonomy would provide stand-alone utility. Additionally, this relationship of terms gives the reader an idea of alternative categories in which to look for technology descriptions.

General Taxonomy Structure

Both taxonomies are structured in a similar manner. Each term or category in a taxonomy has an index number. For the Application Taxonomy, the index numbers begin with AP; for the Quality Measures Taxonomy, the index numbers begin with QM. As mentioned before, a taxonomy is a hierarchical relationship. A category can be broken down into one or more subcategories with the subcategories beginning a new level in the hierarchy. Subcategories are indexed starting with the number 1. For example, index numbers that are subcategories to the first, or root level (AP or QM) would look like AP.2, QM.1, or QM.3.

Subcategories to AP.2, QM.1, or QM.3 would have index numbers like AP.2.4, QM.1.1, or QM.3.2, respectively; subcategories to these would have index numbers like AP.2.4.3, QM.1.1.2, or QM.3.2.1, respectively, and so on.

Some categories have hyphenated subcategories. These subcategories are terms that we feel are worth noting and help further define what type of technology descriptions the reader may find under the parent category. However, they are not sufficiently different from their parent category or in some cases from each other to warrant an index number.

Technology descriptions can be classified into more than one category, and these categories are usually three to four levels deep in the taxonomy.

Using the Taxonomies

When readers find a term within one of the taxonomies that leads them to a list of technology descriptions, they may want to examine the graphical representations of the taxonomies as well. By examining these, readers can identify other possible categories to look under that are related to their original term. For example, if a reader is concerned about reliability, the reader would look at one of the Quality Measures representations and notice that "correctness" and "completeness" are closely related to reliability. The reader could then look for technology descriptions under those categories. This method may give the reader a more complete solution set for their particular problem context.

Note: Within the technology descriptions, some software technologies that are mentioned or referenced do not yet have corresponding descriptions. However, we still indexed these into the Application Taxonomy. When these descriptions are written and more information is gathered, the categories into which these technologies are indexed may change. Thus technologies may appear in this taxonomy without corresponding URLs.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/taxonomies/about_tax.html
Last Modified: 24 July 2008

View the Application Taxonomy

Software Technology Roadmap

The following explains how to approach the graphical representations:

- There is always a two-level deep view from the root figure.
- Due to the structure of this taxonomy, it may take more than one figure to provide a complete two-level deep view.
- If further expansion of the taxonomy is needed (i.e., there is more detail at subordinate levels), the first level is marked with a number in a shaded box located in the lower, right-hand corner. That level is then further expanded (and rotated 90 degrees) in Figure X where X corresponds to the number that the level is marked with.
- You can display a list of technology descriptions categorized at a particular level of the taxonomy. Move to the appropriate figure (1-8), and use your mouse to select the taxonomy term of interest. The list of descriptions is then displayed in the lower frame of this document.

Root Figure: Application

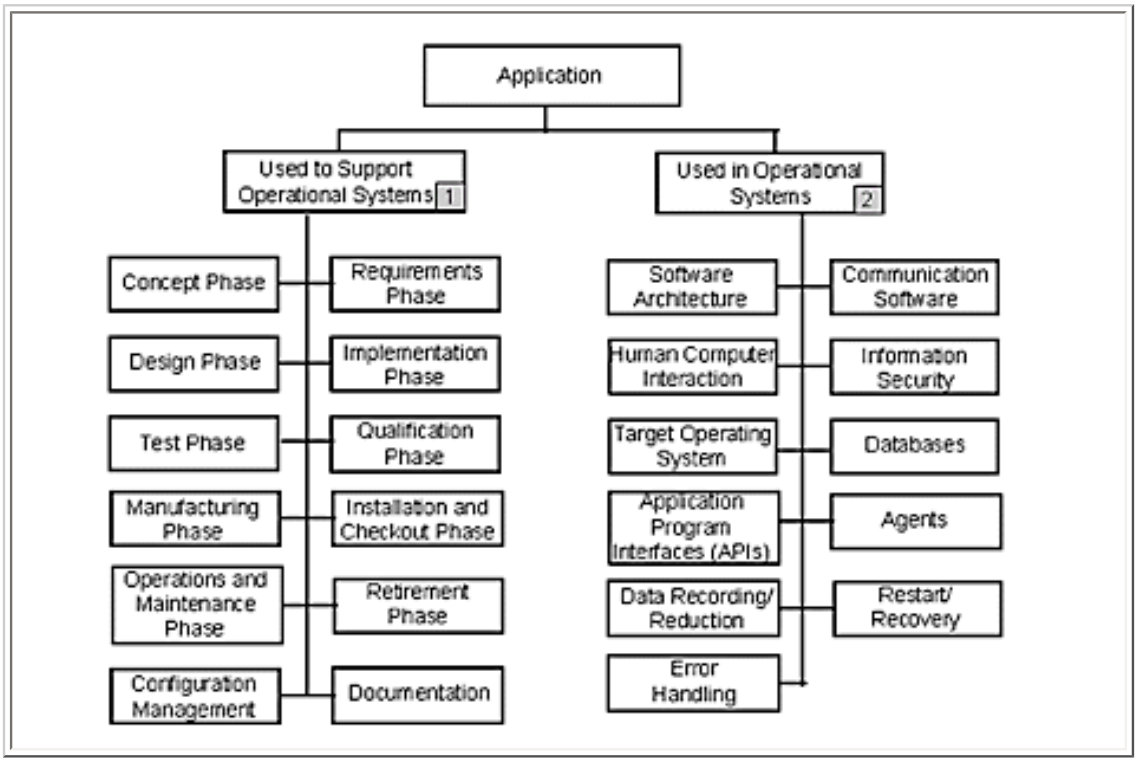


Figure 1a: Used to Support Operational Systems

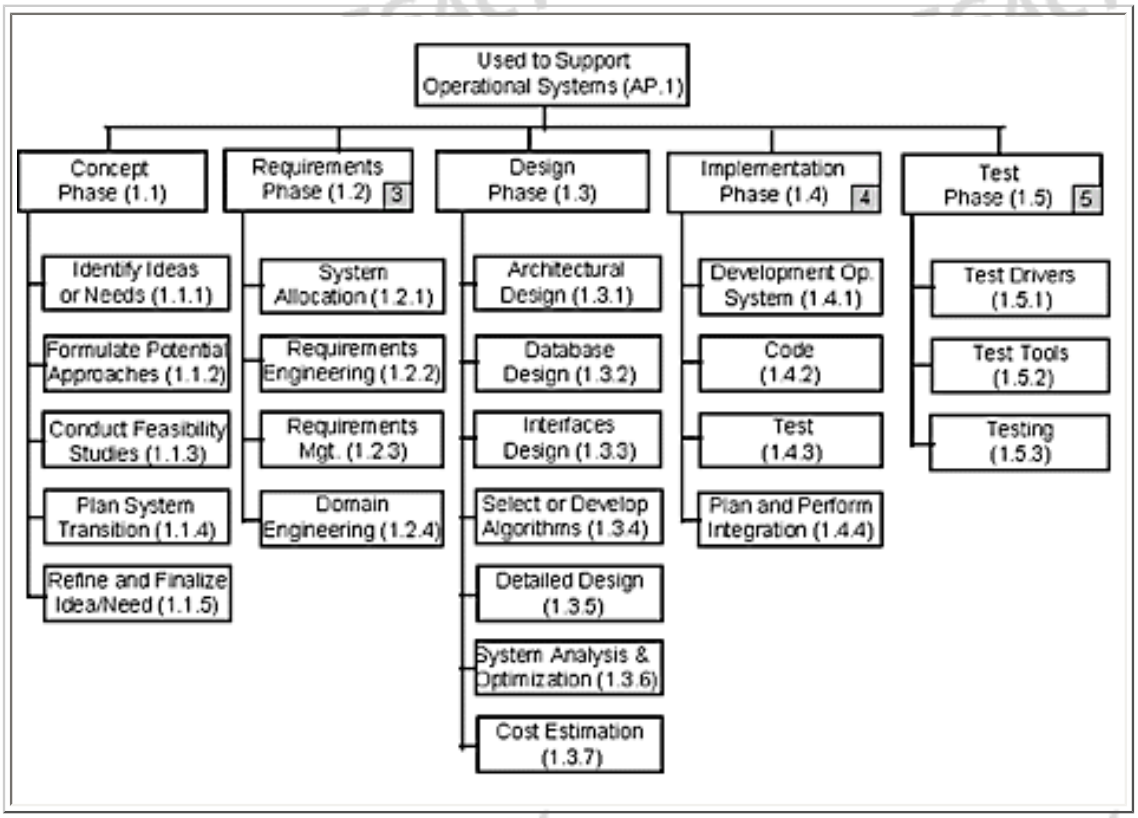


Figure 1b: Used to Support Operational Systems

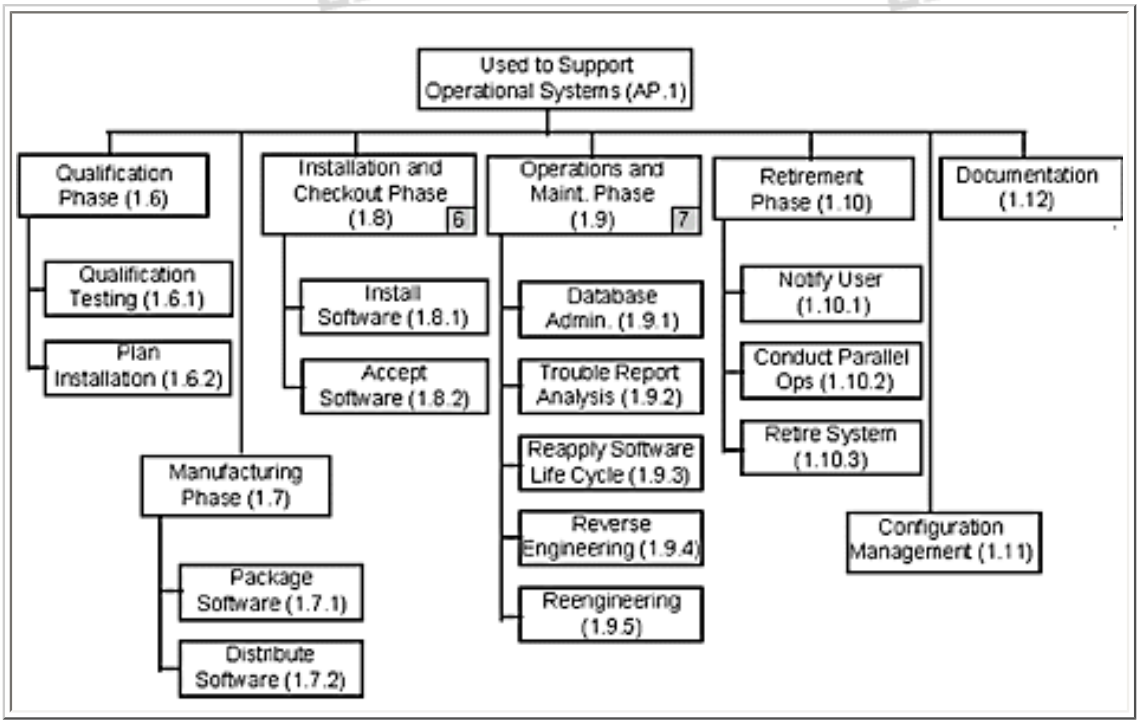


Figure 2a: Used in Operational Systems

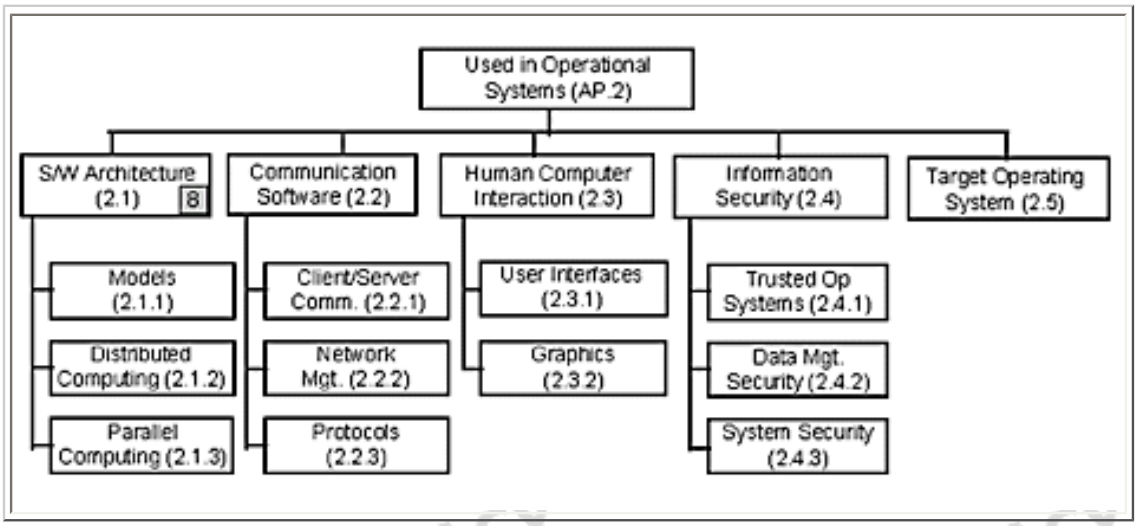


Figure 2b: Used in Operational Systems

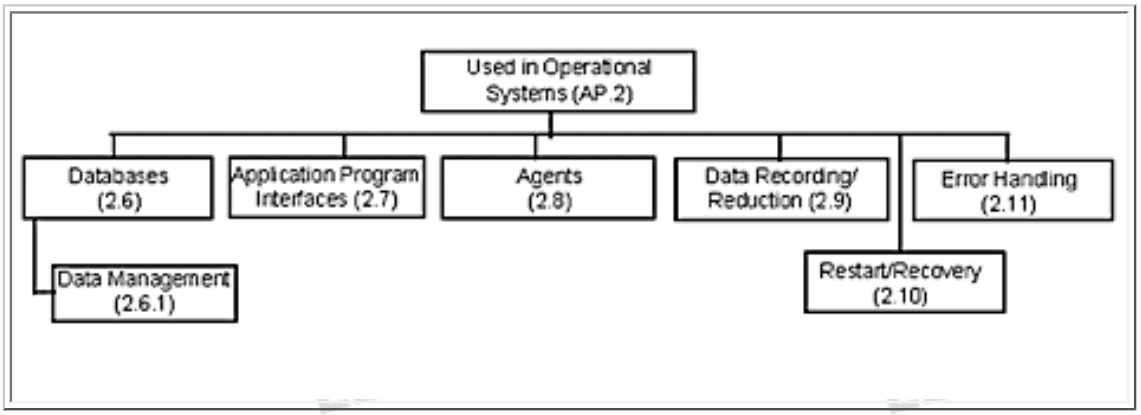


Figure 3: Requirements Phase

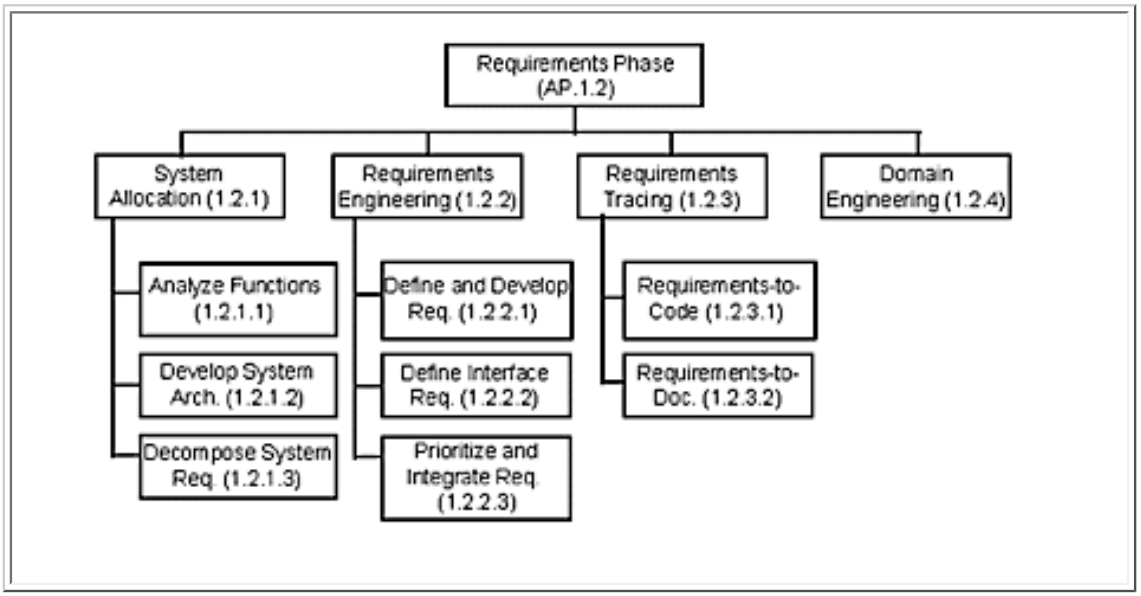


Figure 4: Implementation Phase

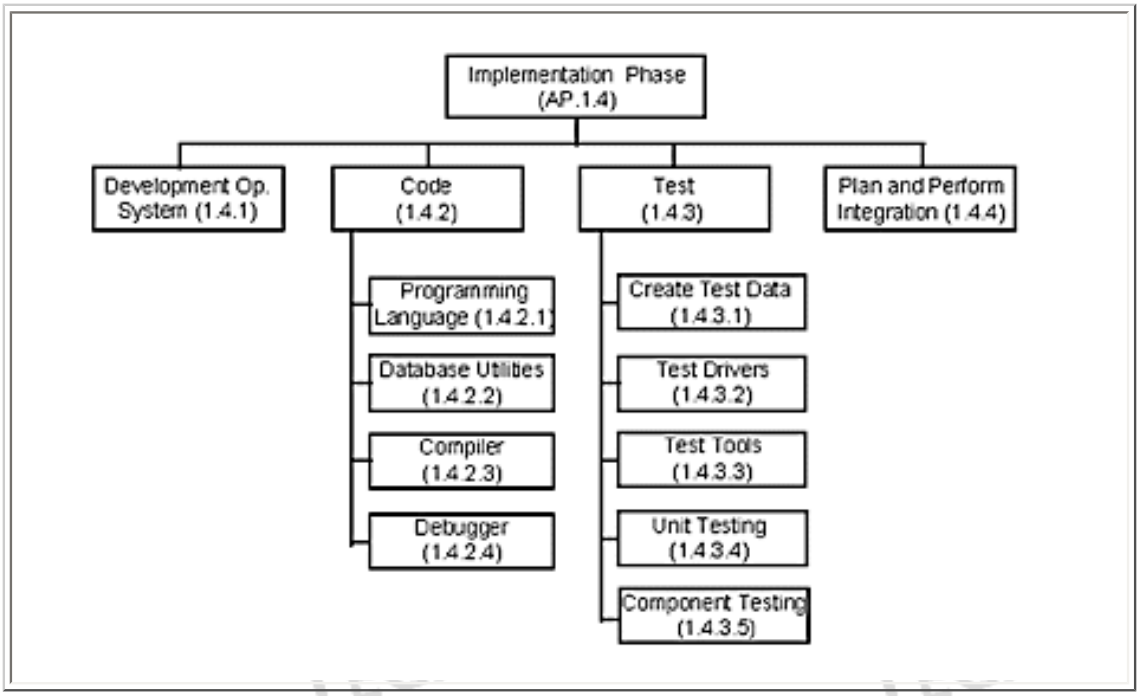


Figure 5: Test Phase

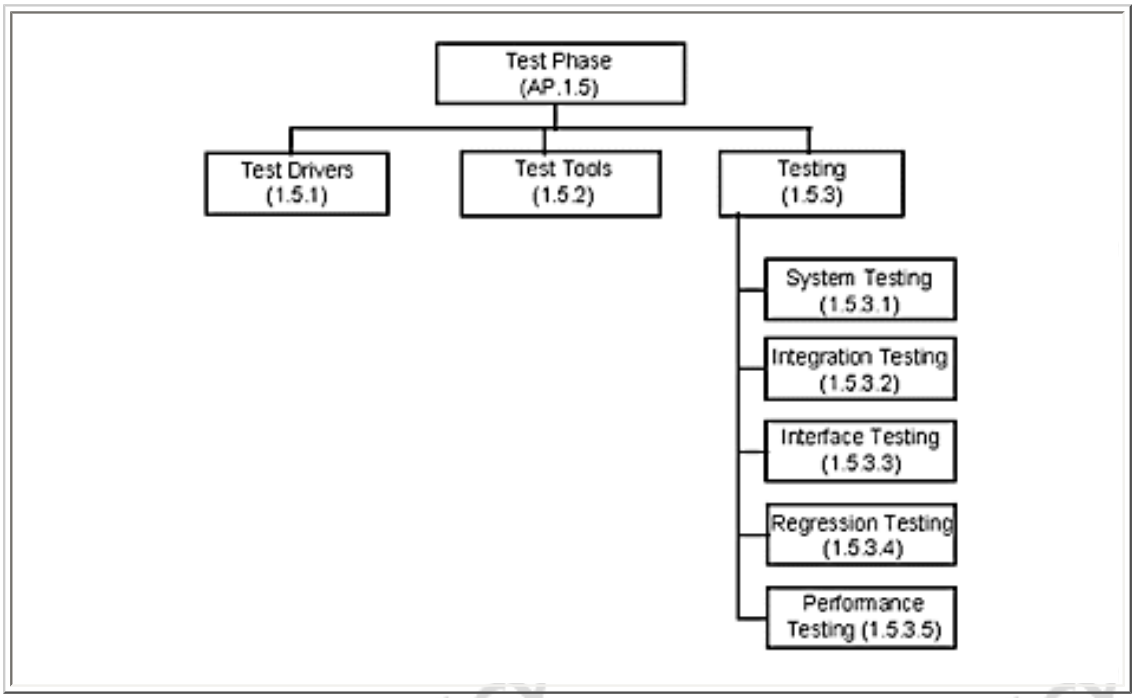


Figure 6: Installation and Checkout Phase

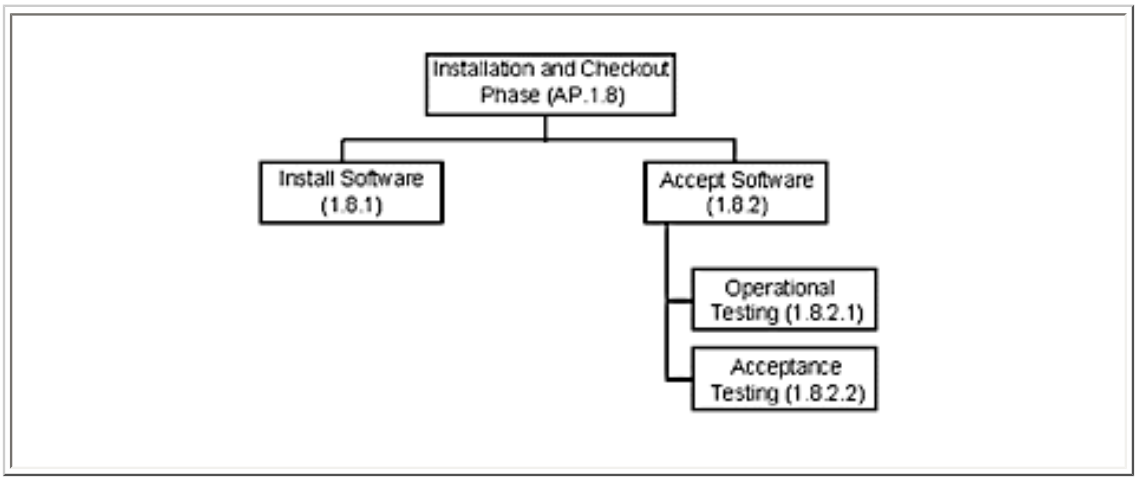


Figure 7: Operations and Maintenance Phase

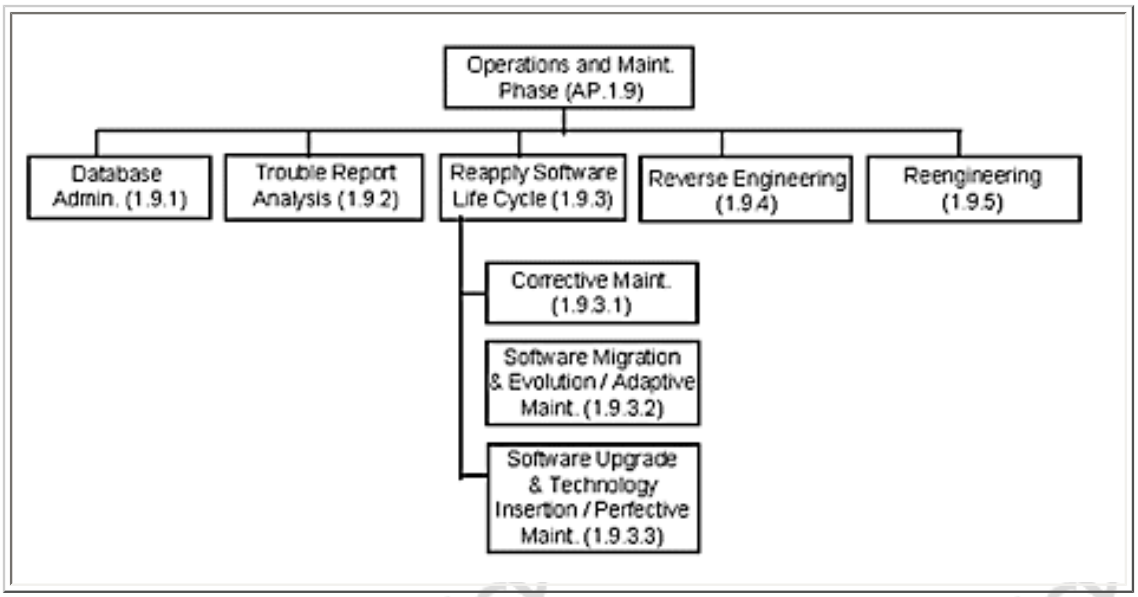
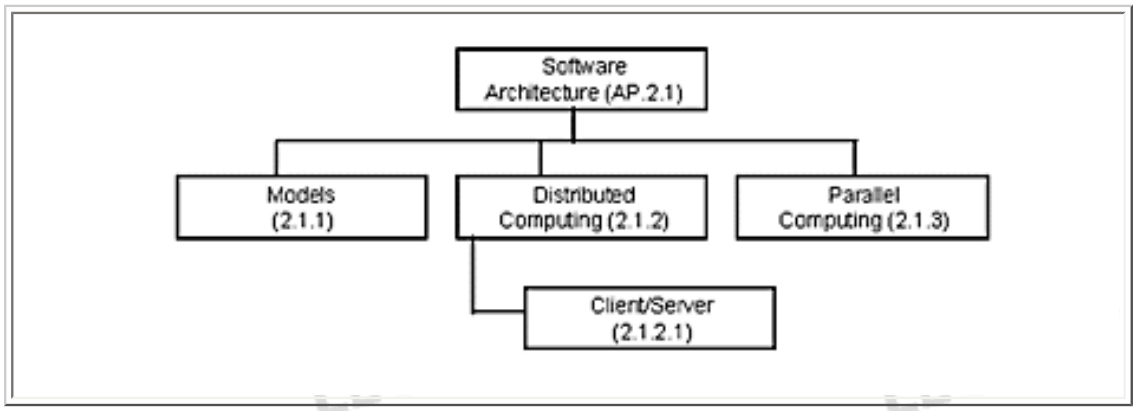


Figure 8: Software Architecture Phase



URL: http://www.sei.cmu.edu/str/taxonomies/view_ap_body.html
Last Modified: 24 July 2008

Technology Descriptions

When you select a taxonomy category from one of the images above, the list of technology descriptions in that category will appear in this frame.

View the Quality Measures Taxonomy

Software Technology Roadmap

The following explains how to approach the graphical representations:

- There is always a two-level deep view from the root figure.
- If further expansion of the taxonomy is needed (i.e., there is more detail at subordinate levels), the first level is marked with a number in a shaded box located in the lower, right-hand corner. That level is then further expanded (and rotated 90 degrees) in Figure X where X corresponds to the number that the level is marked with.
- You can display a list of technology descriptions categorized at a particular level of the taxonomy. Move to the appropriate figure (1-8), and use your mouse to select the taxonomy term of interest. The list of descriptions is then displayed in the lower frame of this document.

Root Figure: Quality Measures

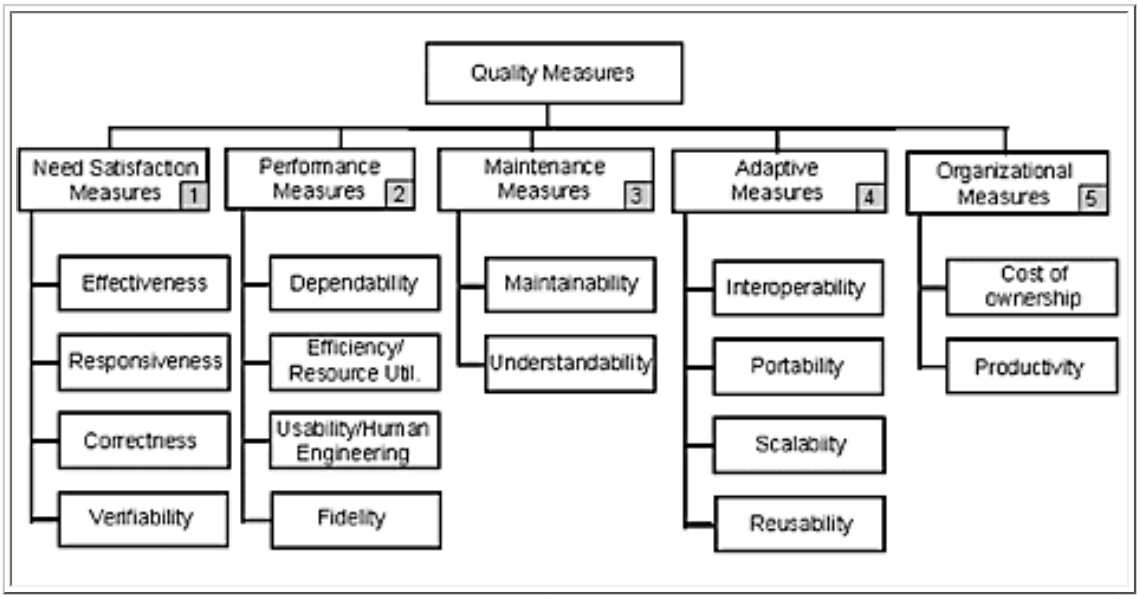


Figure 1: Needs Satisfaction Measures

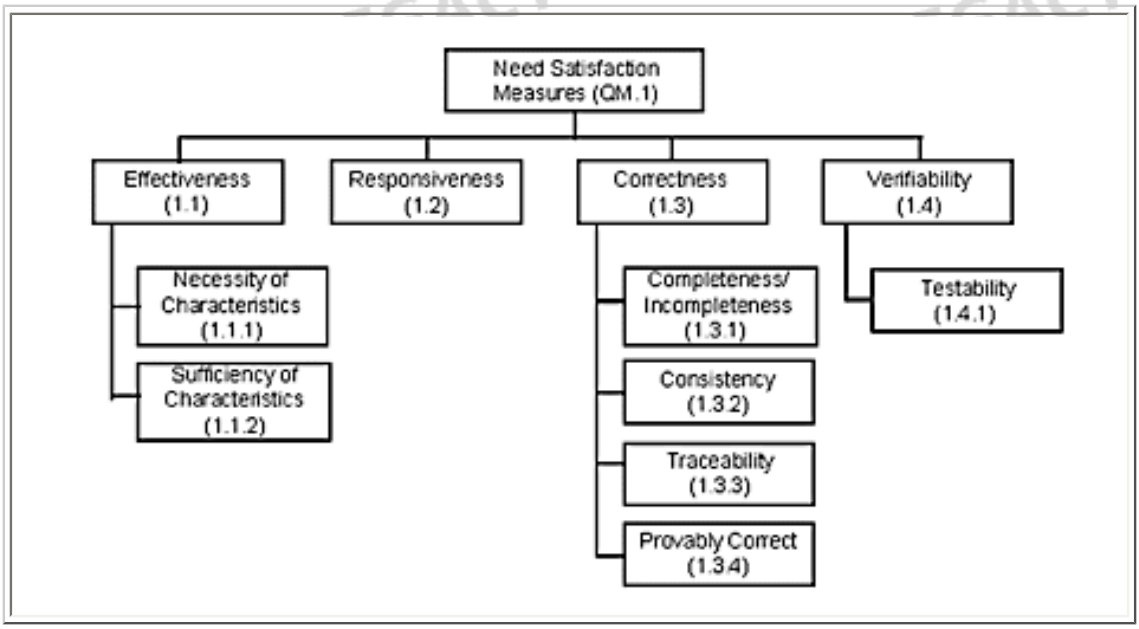


Figure 2: Performance Measures

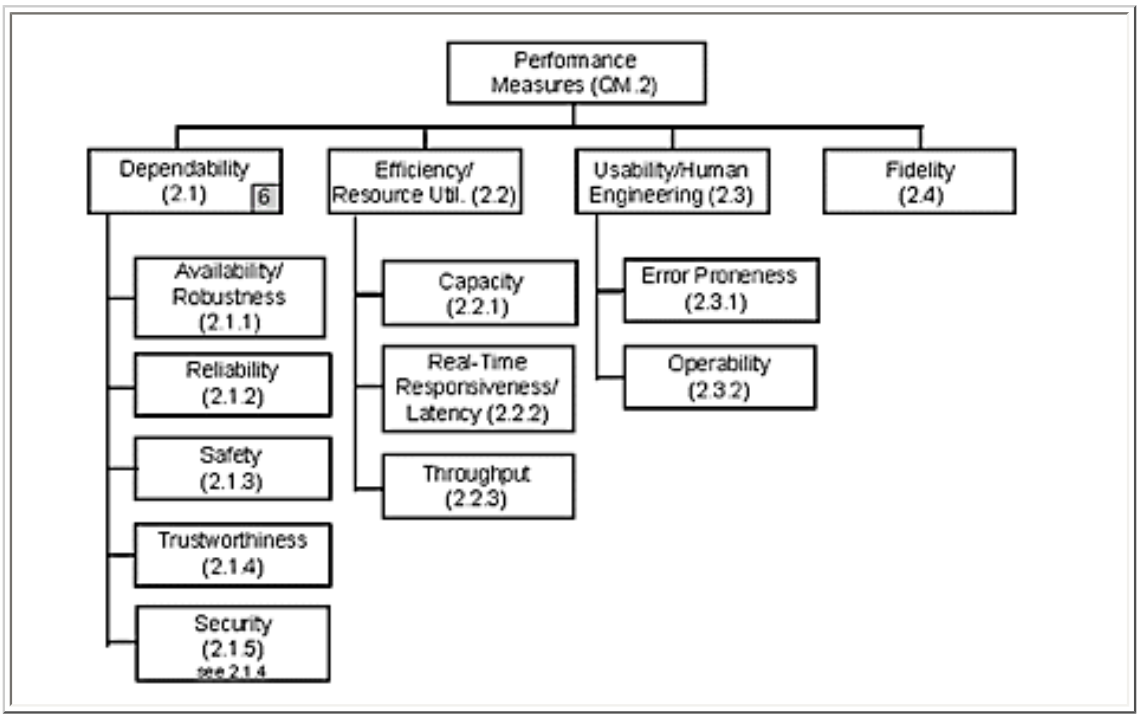


Figure 3: Maintenance Measures

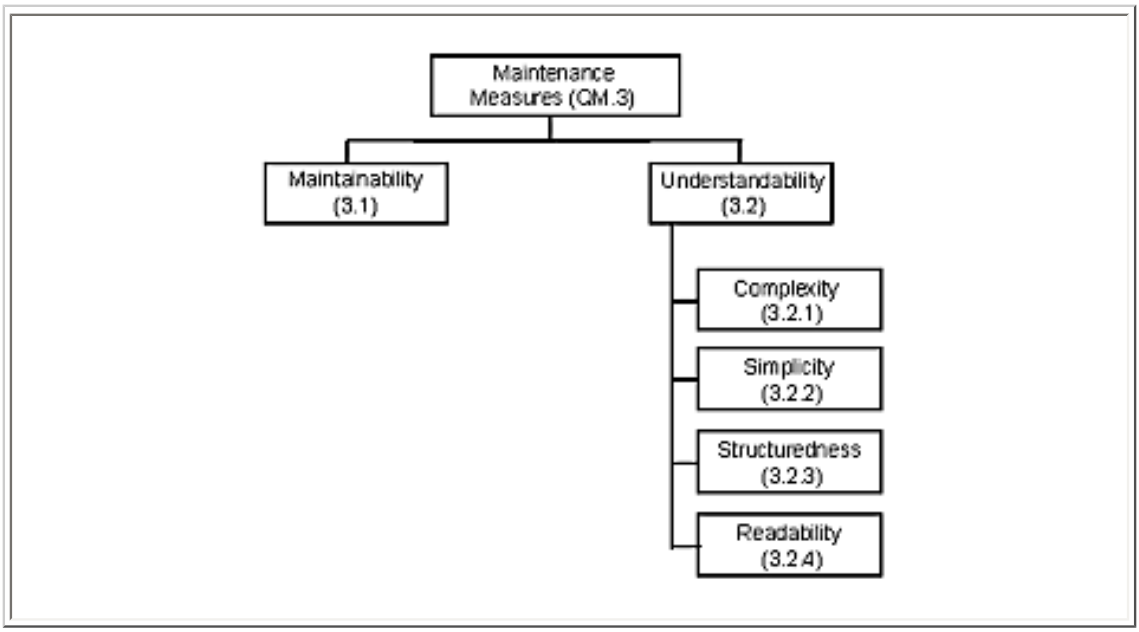


Figure 4: Adaptive Measures

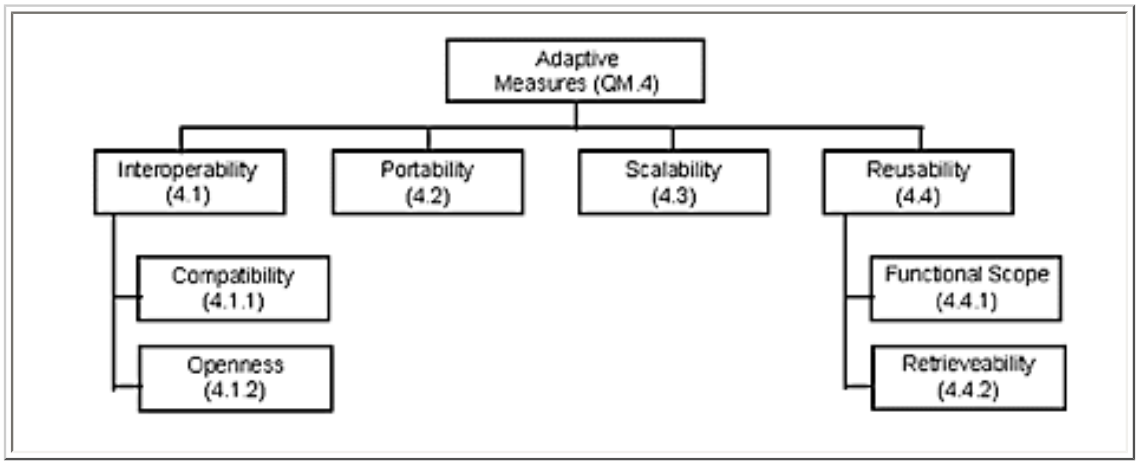


Figure 5: Organizational Measures

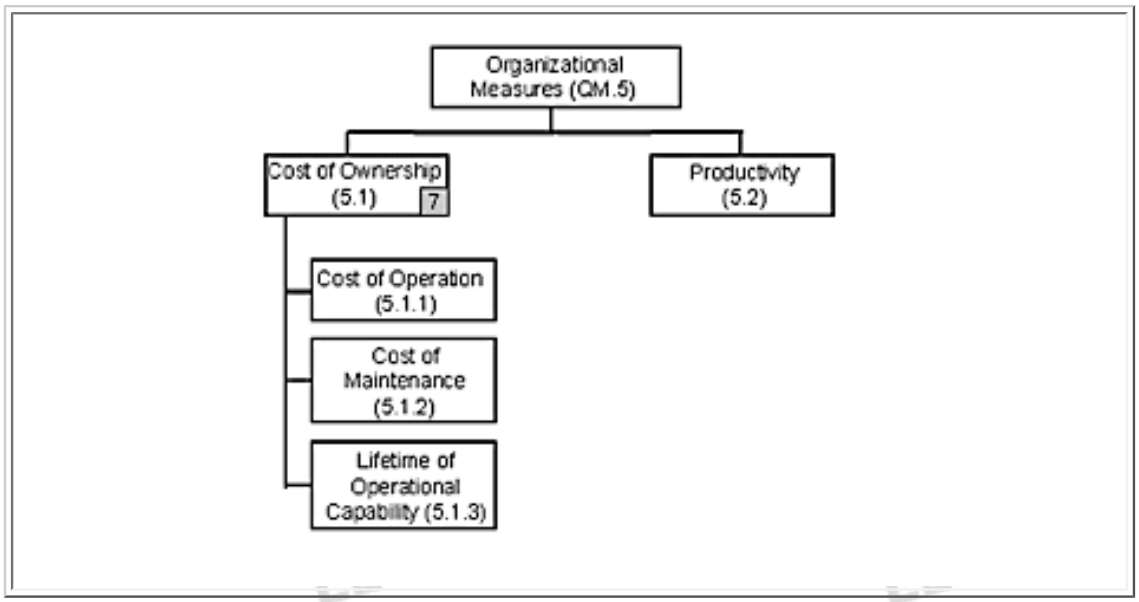


Figure 6: Dependability

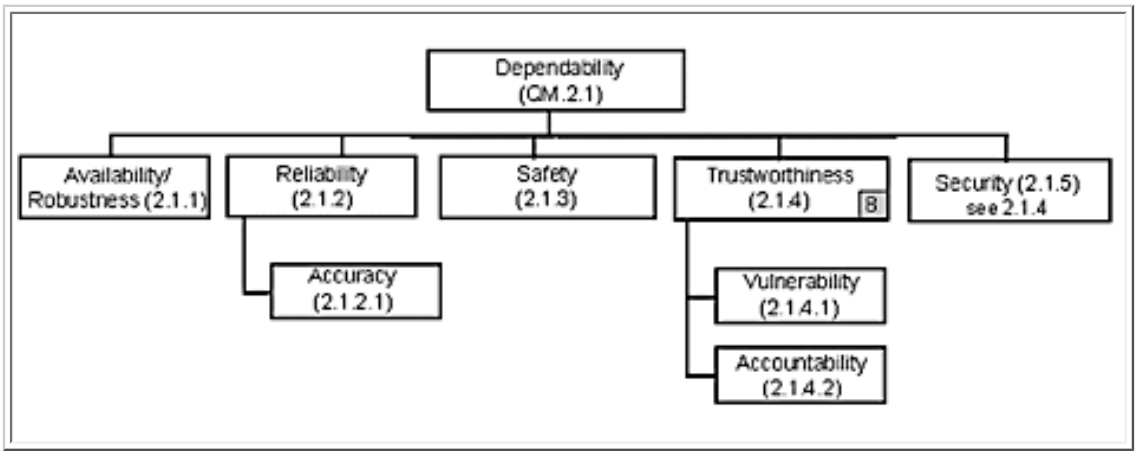


Figure 7: Cost of Ownership

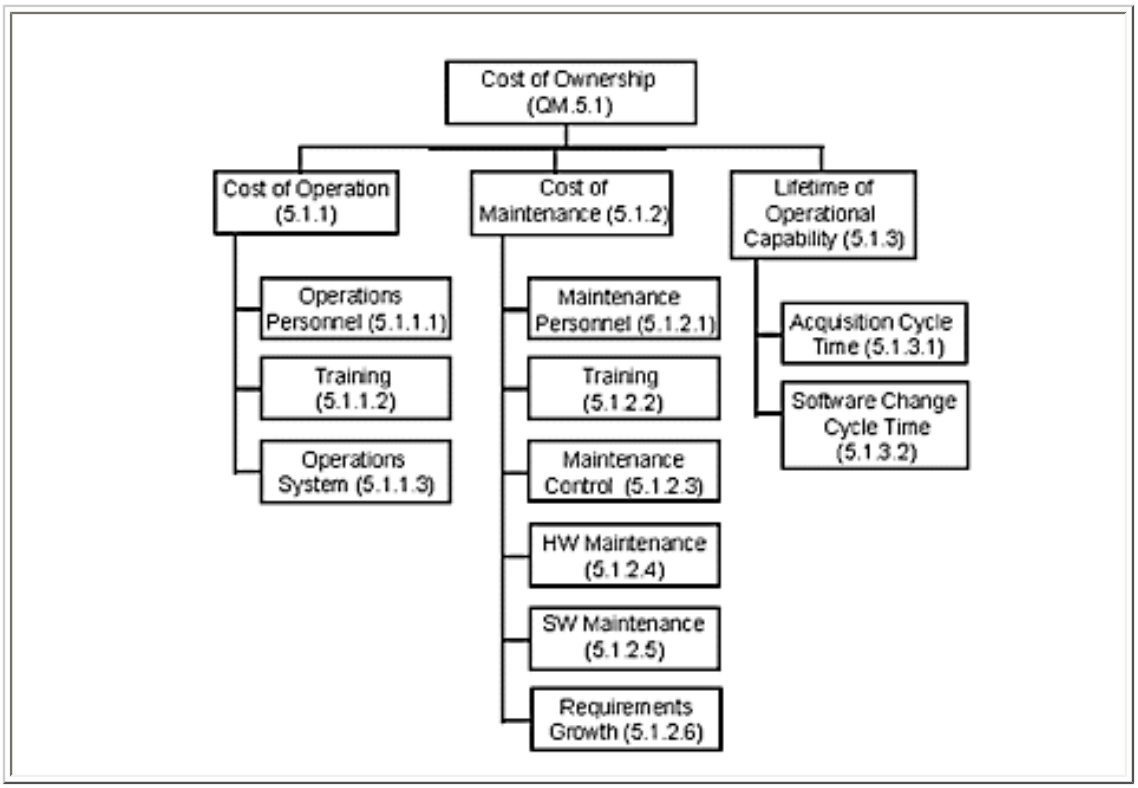
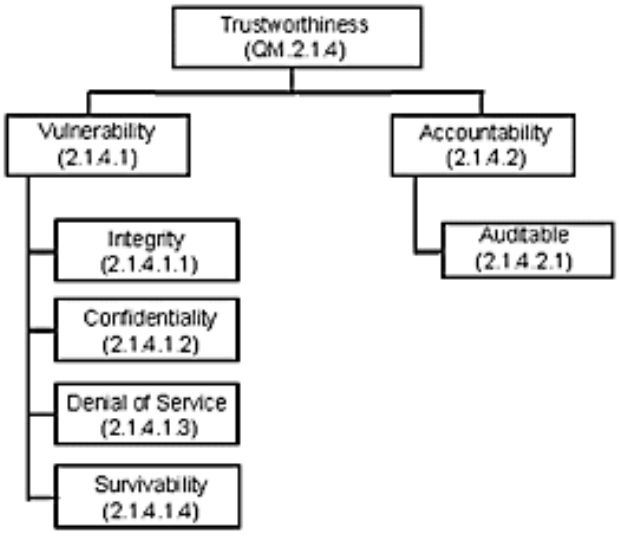


Figure 8: Trustworthiness



The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/taxonomies/view_qm_body.html
Last Modified: 24 July 2008

Technology Descriptions

When you select a taxonomy category from one of the images above, the list of technology descriptions in that category will appear in this frame.

Glossary

Software Technology Roadmap

A-H I-P Q-Z

Abstractness
: the degree to which a system or component performs only the necessary functions relevant to a particular purpose.

Acceptance testing
: formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system [IEEE 90].

Accessibility

1. (Denial of Service) the degree to which the software system protects system functions or service from being denied to the user
2. (Reusability) the degree to which a software system or component facilitates the selective use of its components [Boehm 78].

Accuracy
: a quantitative measure of the magnitude of error [IEEE 90].

Acquisition cycle time
: the period of time that starts when a system is conceived and ends when the product meets its initial operational capability.

Adaptability
: the ease with which software satisfies differing system constraints and user needs [Evans 87].

Adaptive maintenance
: software maintenance performed to make a computer program usable in a changed environment [IEEE 90].

Adaptive measures
: a category of quality measures that address how easily a system can evolve or migrate.

Agent
: a piece of software which acts to accomplish tasks on behalf of its user [McGill 96].

Anonymity

the degree to which a software system or component allows for or supports anonymous transactions.

ANSI

American National Standards Institute. This organization is responsible for approving U.S. standards in many areas, including computers and communications. Standards approved by this organization are often called ANSI standards (e.g., ANSI C is the version of the C language approved by ANSI). ANSI is a member of ISO. *See also:* International Organization for Standardization.

Application program interface

a formalized set of software calls and routines that can be referenced by an application program in order to access supporting system or network services [ITS 96].

Architectural design

the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system [IEEE 90].

Artificial intelligence

a subfield within computer science concerned with developing technology to enable computers to solve problems (or assist humans in solving problems) using explicit representations of knowledge and reasoning methods employing that knowledge [DoD 91].

Auditable

the degree to which a software system records information concerning transactions performed against the system.

Availability

the degree to which a system or component is operational and accessible when required for use [IEEE 90].

Capacity

a measure of the amount of work a system can perform [Barbacci 95].

Code

the transforming of logic and data from design specifications (design descriptions) into a programming language [IEEE 90].

Commonality

the degree to which standards are used to achieve interoperability.

Communication software

software concerned with the representation, transfer, interpretation, and processing of data among computer systems or networks. The meaning assigned to the data must be preserved during these operations.

Compactness

the degree to which a system or component makes efficient use of its data storage space- occupies a small volume.

Compatibility

the ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment [IEEE 90].

Completeness

the degree to which all the parts of a software system or component are present and each of its parts is fully specified and developed [Boehm 78].

Complexity

1. (Apparent) the degree to which a system or component has a design or implementation that is difficult to understand and verify [IEEE 90].
2. (Inherent) the degree of complication of a system or system component, determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, and the types of data structures [Evans 87].

Component testing

testing of individual hardware or software components or groups of related components [IEEE 90].

Concept phase

the initial phase of a software development project, in which the user needs are described and evaluated through documentation (for example, statement of needs, advance planning report, project initiation memo, feasibility studies, system definition, documentation, regulations, procedures, or policies relevant to the project) [IEEE 90].

Conciseness

the degree to which a software system or component has no excessive information present.

Confidentiality

the nonoccurrence of the unauthorized disclosure of information [Barbacci 95].

Consistency

the degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component [IEEE 90].

Corrective maintenance

maintenance performed to correct faults in hardware or software [IEEE 90].

Correctness

the degree to which a system or component is free from faults in its specification, design, and implementation [IEEE 90].

Cost estimation

the process of estimating the "costs" associated with software development projects, to include the effort, time, and labor required.

Cost of maintenance

the overall cost of maintaining a computer system to include the costs

associated with personnel, training, maintenance control, hardware and software maintenance, and requirements growth.

Cost of operation
: the overall cost of operating a computer system to include the costs associated with personnel, training, and system operations.

Cost of ownership
: the overall cost of a computer system to an organization to include the costs associated with operating and maintaining the system, and the lifetime of operational use of the system.

Data management security
: the protection of data from unauthorized (accidental or intentional) modification, destruction, or disclosure [ITS 96].

Data management
: the function that provides access to data, performs or monitors the storage of data, and controls input/output operations [McDaniel 94].

Data recording
: to register all or selected activities of a computer system. Can include both external and internal activity.

Data reduction
: any technique used to transform data from raw data into a more useful form of data. For example, grouping, summing, or averaging related data [IEEE 90].

Database administration
: the responsibility for the definition, operation, protection, performance, and recovery of a database [IEEE 90].

Database design
: the process of developing a database that will meet a user's requirements. The activity includes three separate but dependent steps: conceptual database design, logical database design, and physical database design [IEEE 91].

Database

1. a collection of logically related data stored together in one or more computerized files. Note: Each data item is identified by one or more keys [IEEE 90].
2. an electronic repository of information accessible via a query language interface [DoD 91].

Denial of service
: the degree to which a software system or component prevents the interference or disruption of system services to the user.

Dependability
: that property of a computer system such that reliance can justifiably be placed on the service it delivers [Barbacci 95].

Design phase
: the period of time in the software life cycle during which the designs for architecture, software components, interfaces, and data are created, documented, and verified to satisfy requirements [IEEE 90].

Detailed design
: the process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented [IEEE 90].

Distributed computing
: a computer system in which several interconnected computers share the computing tasks assigned to the system [IEEE 90].

Domain analysis
: the activity that determines the common requirements within a domain for the purpose of identifying reuse opportunities among the systems in the domain. It builds a domain architectural model representing the commonalities and differences in requirements within the domain (problem space) [ARC 96].

Domain design
: the activity that takes the results of domain analysis to identify and generalize solutions for those common requirements in the form of a Domain-Specific Software Architecture (DSSA). It focuses on the problem space, not just on a particular system's requirements, to design a solution (solution space) [ARC 96].

Domain engineering
: the process of analysis, specification and implementation of software assets in a domain which are used in the development of multiple software products [SEI 96]. The three main activities of domain engineering are: domain analysis, domain design, and domain implementation [ARC 96].

Domain implementation
: the activity that realizes the reuse opportunities identified during domain analysis and design in the form of common requirements and design solutions, respectively. It facilitates the integration of those reusable assets into a particular application [ARC 96].

Effectiveness
: the degree to which a system's features and capabilities meet the user's needs.

Efficiency
: the degree to which a system or component performs its designated functions with minimum consumption of resources (CPU, Memory, I/O, Peripherals, Networks) [IEEE 90].

Error handling
: the function of a computer system or component that identifies and responds to user or system errors to maintain normal or at the very least

degraded operations.

Error proneness
: the degree to which a system may allow the user to intentionally or unintentionally introduce errors into or misuse the system.

Error tolerance
: the ability of a system or component to continue normal operation despite the presence of erroneous inputs [IEEE 90].

Evolvability
: the ease with which a system or component can be modified to take advantage of new software or hardware technologies.

Expandability
: see Extendability [IEEE 90].

Extendability
: the ease with which a system or component can be modified to increase its storage or functional capacity [IEEE 90].

Fail safe
: pertaining to a system or component that automatically places itself in a safe operating mode in the event of a failure [IEEE 90].

Fail soft
: pertaining to a system or component that continues to provide partial operational capability in the event of certain failures [IEEE 90].

Fault tolerance
: the ability of a system or component to continue normal operation despite the presence of hardware or software faults [IEEE 90].

Fault
: an incorrect step, process, or data definition in a computer program [IEEE 90].

Fidelity
: the degree of similarity between a model and the system properties being modeled [IEEE 90].

Flexibility
: the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed [IEEE 90].

Functional scope
: the range or scope to which a system component is capable of being applied.

Functional testing
: testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. Synonym: black-box testing [IEEE 90].

Generality
: the degree to which a system or component performs a broad range of functions [IEEE 90].

Graphics
: methods and techniques for converting data to or from graphic display via computers [McDaniel 94].

Hardware maintenance
: the cost associated with the process of retaining a hardware system or component in, or restoring it to, a state in which it can perform its required functions.

Human Computer Interaction
: a subfield within computer science concerned with the design, evaluation, and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them [Toronto 95].

Human engineering
: the extent to which a software product fulfills its purpose without wasting user's time and energy or degrading their morale [Boehm 78].

Implementation phase
: the period of time in the software life cycle during which a software product is created from design documentation and debugged [IEEE 90].

Incompleteness
: the degree to which all the parts of a software system or component are not present and each of its parts is not fully specified or developed.

Information Security
: the concepts, techniques, technical measures, and administrative measures used to protect information assets from deliberate or inadvertent unauthorized acquisition, damage, disclosure, manipulation, modification, loss, or use [McDaniel 94].

Installation and checkout phase
: the period of time in the software life cycle during which a software product is integrated into its operational environment and tested in this environment to ensure it performs as required [IEEE 90].

Integration testing
: testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them [IEEE 90].

Integrity
: the degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data [IEEE 90].

Interface testing
: testing conducted to evaluate whether systems or components pass data and control correctly to one another [IEEE 90].

Interfaces design
: the activity concerned with the interfaces of the software system contained in the software requirements and software interface requirements documentation. Consolidates the interface descriptions into a single interface description of the software system [IEEE 91].

Interoperability
: the ability of two or more systems or components to exchange information and to use the information that has been exchanged [IEEE 90].

ISO
: International Organization for Standardization. A voluntary, non-treaty organization founded in 1946 which is responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of the 89 member countries, including ANSI for the U.S.

Latency
: the length of time it takes to respond to an event [Barbacci 95].

Lifetime of operational capability
: the total period of time in a system's life that it is operational and meeting the user's needs.

Maintainability
: the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment [IEEE 90].

Maintenance control
: the cost of planning and scheduling hardware preventive maintenance, and software maintenance and upgrades, managing the hardware and software baselines, and providing response for hardware corrective maintenance.

Maintenance measures
: a category of quality measures that address how easily a system can be repaired or changed.

Maintenance personnel
: the number of personnel needed to maintain all aspects of a computer system, including the support personnel and facilities needed to support that activity.

Managed device
: any type of node residing on a network, such as a computer, printer or routers that contain a management agent.

Managed object
: a characteristic of a managed device that can be monitored, modified or controlled.

Management agent
: software that resides in a managed device that allows the device to be monitored and/or controlled by a network management application.

Manufacturing phase
: the period of time in the software life cycle during which the basic version of a software product is adapted to a specified set of operational environments and is distributed to a customer base [IEEE 90].

Model
: an approximation, representation, or idealization of selected aspects of the structure, behavior, operation, or other characteristics of a real-world process, concept, or system. Note: Models may have other models as components [IEEE 90].

Modifiability
: the degree to which a system or component facilitates the incorporation of changes, once the nature of the desired change has been determined [Boehm 78].

Necessity of characteristics
: the degree to which all of the necessary features and capabilities are present in the software system.

Need satisfaction measures
: a category of quality measures that address how well a system meets the user's needs and requirements.

Network management
: the execution of the set of functions required for controlling, planning, allocating, deploying, coordinating, and monitoring the resources of a computer network [ITS 96].

Network management application
: application that provides the ability to monitor and control the network.

Network management information
: information that is exchanged between the network management station (s) and the management agents that allows the monitoring and control of a managed device.

Network management protocol
: protocol used by the network management station(s) and the management agent to exchange management information.

Network management station
: system that hosts the network management application.

Openness
: the degree to which a system or component complies with standards.

Operability
: the ease of operating the software [Deutsch 88].

Operational testing
: testing conducted to evaluate a system or component in its operational environment [IEEE 90].

Operations and maintenance phase
: the period of time in the software life cycle during which a software product is employed in its operational environment, monitored for satisfactory performance, and modified as necessary to correct problems or to respond to changing requirements [IEEE 90].

Operations personnel
: the number of personnel needed to operate all aspects of a computer system, including the support personnel and facilities needed to support that activity.

Operations system
: the cost of environmentals, communication, licenses, expendables, and documentation maintenance for an operational system.

Organizational measures
: a category of quality measures that address how costly a system is to operate and maintain.

Parallel computing
: a computer system in which interconnected processors perform concurrent or simultaneous execution of two or more processes [McDaniel 94].

Perfective maintenance
: software maintenance performed to improve the performance, maintainability, or other attributes of a computer program [IEEE 90].

Performance measures
: a category of quality measures that address how well a system functions.

Performance testing
: testing conducted to evaluate the compliance of a system or component with specified performance requirements [IEEE 90].

Portability
: the ease with which a system or component can be transferred from one hardware or software environment to another [IEEE 90].

Productivity
: the quality or state of being productive [Webster 87].

Protocol
: a set of conventions that govern the interaction of processes, devices, and other components within a system [IEEE 90].

Provably correct
: the ability to mathematically verify the correctness of a system or component.

Qualification phase
: the period of time in the software life cycle during which it is determined whether a system or component is suitable for operational use.

Qualification testing

testing conducted to determine whether a system or component is suitable for operational use [IEEE 90].

Quality measure
: a software feature or characteristic used to assess the quality of a system or component.

Readability
: the degree to which a system's functions and those of its component statements can be easily discerned by reading the associated source code.

Real-time responsiveness
: the ability of a system or component to respond to an inquiry or demand within a prescribed time frame.

Recovery
: the restoration of a system, program, database, or other system resource to a prior state following a failure or externally caused disaster; for example, the restoration of a database to a point at which processing can be resumed following a system failure [IEEE 90].

Reengineering
: rebuilding a software system or component to suit some new purpose; for example to work on a different platform, to switch to another language, to make it more maintainable.

Regression testing
: selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements [IEEE 90].

Reliability
: the ability of a system or component to perform its required functions under stated conditions for a specified period of time [IEEE 90].

Requirements engineering
: involves all life-cycle activities devoted to identification of user requirements, analysis of the requirements to derive additional requirements, documentation of the requirements as a specification, and validation of the documented requirements against user needs, as well as processes that support these activities [DoD 91].

Requirements growth
: the rate at which the requirements change for an operational system. The rate can be positive or negative.

Requirements phase
: the period of time in the software life cycle during which the requirements for a software product are defined and documented [IEEE 90].

Requirements tracing
: describing and following the life of a requirement in both forwards and backwards direction (i.e., from its origins, through its development and

specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases) [Gotel 95].

Resource utilization
: the percentage of time a resource (CPU, Memory, I/O, Peripheral, Network) is busy [Barbacci 95].

Responsiveness
: the degree to which a software system or component has incorporated the user's requirements.

Restart
: to cause a computer program to resume execution after a failure, using status and results recorded at a checkpoint [IEEE 90].

Retirement phase
: the period of time in the software life cycle during which support for a software product is terminated [IEEE 90].

Reusability
: the degree to which a software module or other work product can be used in more than one computing program or software system [IEEE 90].

Reverse engineering
: the process of analyzing a system's code, documentation, and behavior to identify its current components and their dependencies to extract and create system abstractions and design information. The subject system is not altered; however, additional knowledge about the system is produced.

Robustness
: the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions [IEEE 90].

Safety
: a measure of the absence of unsafe software conditions. The absence of catastrophic consequences to the environment [Barbacci 95].

Scalability
: the ease with which a system or component can be modified to fit the problem area.

Security
: the ability of a system to manage, protect, and distribute sensitive information.

Select or develop algorithms
: the activity concerned with selecting or developing a procedural representation of the functions in the software requirements documentation for each software component and data structure. The algorithms shall completely satisfy the applicable functional and/or mathematical specifications [IEEE 91].

Self-descriptiveness
: the degree to which a system or component contains enough information

to explain its objectives and properties [IEEE 90].

Simplicity
: the degree to which a system or component has a design and implementation that is straightforward and easy to understand [IEEE 90].

Software architecture
: the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time [Clements 96].

Software change cycle time
: the period of time that starts when a new system requirement is identified and ends when the requirement has been incorporated into the system and delivered for operational use.

Software life cycle
: the period of time that begins when a software product is conceived and ends when the software is no longer available for use. The life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and sometimes, retirement phase. These phases may overlap or be performed iteratively, depending on the software development approach used [IEEE 90].

Software maintenance
: the cost associated with modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment.

Software migration and evolution
: see Adaptive maintenance.

Software upgrade and technology insertion
: see Perfective maintenance.

Speed
: the rate at which a software system or component performs its functions.

Statistical testing
: employing statistical science to evaluate a system or component. Used to demonstrate a system's fitness for use, to predict the reliability of a system in an operational environment, to efficiently allocate testing resources, to predict the amount of testing required after a system change, to qualify components for reuse, and to identify when enough testing has been accomplished [Poore 96].

Structural testing
: testing that takes into account the internal mechanism of a system or component. Types include branch testing, path testing, statement testing. Synonym: white-box testing [IEEE 90].

Structuredness
: the degree to which a system or component possesses a definite pattern

of organization of its interdependent parts [Boehm 78].

Sufficiency of characteristics
: the degree to which the features and capabilities of a software system adequately meet the user's needs.

Survivability
: the degree to which essential functions are still available even though some part of the system is down [Deutsch 88].

System allocation
: mapping the required functions to software and hardware. This activity is the bridge between concept exploration and the definition of software requirements [IEEE 91].

System analysis and optimization
: a systematic investigation of a real or planned system to determine the information requirements and processes of the system and how these relate to each other and to any other system, and to make improvements to the system where possible.

System security
: a system function that restricts the use of objects to certain users [McDaniel 94].

System testing
: testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements [IEEE 90].

Taxonomy
: a scheme that partitions a body of knowledge and defines the relationships among the pieces. It is used for classifying and understanding the body of knowledge [IEEE 90].

Test drivers
: software modules used to invoke a module(s) under test and, often, provide test inputs, control and monitor execution, and report test results [IEEE 90].

Test phase
: the period of time in the software life cycle during which the components of a software product are evaluated and integrated, and the software product is evaluated to determine whether or not requirements have been satisfied [IEEE 90].

Test tools
: computer programs used in the testing of a system, a component of the system, or its documentation. Examples include monitor, test case generator, timing analyzer [IEEE 90].

Test
: an activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component [IEEE 90].

Testability
: the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met [IEEE 90]. Note: Not only is testability a measurement for software, it can also apply to the testing scheme.

Testing
: the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component [IEEE 90].

Throughput
: the amount of work that can be performed by a computer system or component in a given period of time [IEEE 90].

Traceability
: the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another [IEEE 90].

Training
: Provisions to learn how to develop, maintain, or use the software system.

Trouble report analysis
: the methodical investigation of a reported operational system deficiency to determine what, if any, corrective action needs to be taken.

Trustworthiness
: the degree to which a system or component avoids compromising, corrupting, or delaying sensitive information.

Understandability
: the degree to which the purpose of the system or component is clear to the evaluator [Boehm 78].

Unit testing
: testing of individual hardware or software units or groups of related units [IEEE 90].

Upgradeability
: see Evolvability.

Usability
: the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component [IEEE 90].

User interface
: an interface that enables information to be passed between a human user and hardware or software components of a computer system [IEEE 90].

Verifiability
: the relative effort to verify the specified software operation and

performance [Evans 87].

Vulnerability

the degree to which a software system or component is open to unauthorized access, change, or disclosure of information and is susceptible to interference or disruption of system services.

URL: http://www.sei.cmu.edu/str/indexes/glossary/index_body.html
Last Modified: 24 July 2008

References and Information Sources

This frame provides full citations for references used in definitions above.

Keyword Index

Software Technology Roadmap

The Keyword Index is structured as in any typical document; a few nuances of our index include the following:

- If the keyword is defined in the glossary, it is linked to that glossary definition.
- If the keyword is the name of a technology, it appears in bold type and is linked to that technology description.
- If the keyword is a category in one of the taxonomies, it is followed by that category's index label in parenthesis. AP and QM labels are linked to a list of technology descriptions included in the category.

Each keyword is followed by a list of the technology descriptions that reference it. After selecting one of the descriptions, use your browser's "find" capabilities to locate instances of the keyword.

A| B| C| D| E| F| G| H| I| J| K| L| M| N| O| P| Q| R| S| T| U| V| W| X| Y| Z

A

abstraction
    Object-Oriented Analysis

abstractness (QM.4.4.1.x)

acceptance testing (AP.1.8.2.2)

accessibility (QM.2.1.4.1.3.x), (QM.4.4.1.x)

accountability (QM.2.1.4.2)

accuracy (QM.2.1.2.1)
    Database Two Phase Commit

acquisition cycle time

Active Group
    Component Object Model (COM), DCOM, and Related Capabilities

ActiveX
    Component Object Model (COM), DCOM, and Related Capabilities

Ada 83
    Ada 95

Ada 95

Ada 83
Common Object Request Broker Architecture
Rate Monotonic Analysis
adaptability (QM.3.1.x)
adaptive-maintenance (AP.1.9.3.2)
adaptive-measures (QM.4)
ADL. see architecture description languages
agents (AP.2.8)
Algorithm Formalization
American National Standards Institute
Ada 83
anonymity (QM.2.1.4.1.2.x)
ANSI. see American National Standards Institute
aperiodic task/process
Rate Monotonic Analysis
API. see application program interfaces
application engineering
Domain Analysis and Domain Engineering
application program interfaces (AP.2.7)
Application Program Interface
COTS and Open Systems--An Overview
Java
Message-Oriented Middleware
Middleware
private
Defense Information Infrastructure Common Operating Environment
public
Defense Information Infrastructure Common Operating Environment
application server
Three Tier Software Architectures
applications
event-driven
Message-Oriented Middleware
architectural design (AP.1.3.1)
architecture
Architecture Description Languages
Cleanroom Software Engineering
description languages
Architecture Description Languages
Module Interconnection Languages
modeling
Feature-Oriented Domain Analysis
Reference Models, Architectures, Implementations--An Overview
Argument-Based Design Rationale Capture Methods for Requirements

Tracing
artificial intelligence
asynchronous
processing
Distributed Computing Environment
auditable (QM.2.1.4.2.1)
Authenticode
Component Object Model (COM), DCOM, and Related Capabilities
automatic programming
availability (QM.2.1.1)
Intrusion Detection
Software Inspections
Statistical-Based Intrusion Detection

B

backfiring
Function Point Analysis
Bang measure
Function Point Analysis
binary large objects
Object-Oriented Database
black-box testing (AP.1.4.3.4.x)
BLOBS. see binary large objects
Bowles metrics
Cyclomatic Complexity
box structure method
Cleanroom Software Engineering
browsers
Computer System Security--An Overview

C

C
Ada 83
Common Object Request Broker Architecture
Distributed Computing Environment
C++
Ada 83
Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures
C4I
Computer System Security--An Overview
Capability Maturity Model

Cleanroom Software Engineering
Personal Software Process for Module-Level Development
Software Inspections

capacity (QM.2.2.1)

cell
in distributed computing
Distributed Computing Environment

Cleanroom Software Engineering
Object-Oriented Analysis
Object-Oriented Design

client
Two Tier Software Architectures

client/server (AP.2.1.2.1)
communication (AP.2.2.1)
Distributed Computing Environment
Mainframe Server Software Architectures
Message-Oriented Middleware
Object Request Broker
Remote Procedure Call
Software Architectures

CMIP. see Common Management Information Protocol

CMM. see Capability Maturity Model

Coad-Yourdan
Object-Oriented Analysis

COCOMO. see constructive cost model

code (AP.1.4.2)
analyzers (AP.1.4.3.4.x)
complexity
Halstead Complexity Measures
entropy
Maintenance of Operational Systems--An Overview
generator
Graphical User Interface Builders

COE. see Common Operating Environment

commercial-off-the-shelf
Application Programming Interface
Component-Based Software Development/COTS Integration
COTS and Open Systems--An Overview
integration
Application Programming Interface

commit phase
Database Two Phase Commit

Common Management Information Protocol
Simple Network Management Protocol

Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures
Distributed Computing Environment

Middleware
Object Request Broker
compliance
Object Request Broker
implementations
Object Request Broker

Common Operating Environment
TAFIM Reference Model
architecture
Defense Information Infrastructure Common Operating Environment
compliance levels
Defense Information Infrastructure Common Operating Environment
component segments
Defense Information Infrastructure Common Operating Environment
Information Server
Defense Information Infrastructure Common Operating Environment
Software Repository System
Defense Information Infrastructure Common Operating Environment

commonality (QM.4.1.2.x)
Domain Engineering and Domain Analysis

communication software (AP.2.2)

compactness (QM.2.2.x)

compartmented mode workstations
Computer System Security--An Overview
Trusted Operating Systems

compatibility (QM.4.1.1)
Graphic Tools for Legacy Database Migration

compiler (AP.1.4.2.3)
Ada 83
Ada 95

completeness (QM.1.3.1)
Requirements Tracing

complexity (QM.3.2.1)
Halstead Complexity Measures
Message-Oriented Middleware
Remote Procedure Call
analysis
Maintenance of Operational Systems--An Overview
apparent (QM.3.2.1.x)
inherent (QM.3.2.1.x)

compliance (standalone)
Common Object Request Broker Architecture

Defense Information Infrastructure Common Operating Environment
component
adaptation
Component-Based Software Development/COTS Integration
assembly
Component-Based Software Development/COTS Integration
selection and evaluation
Component-Based Software Development/COTS Integration
testing (AP.1.4.3.5)
Component Object Model
Distributed Computing Environment
Middleware
Object Request Broker
Component-Based Software Development/COTS Intergration
component-based software engineering
Component-Based Software Development/COTS Intergration
computational complexity
Halstead Complexity Measures
Computer System Security--An Overview
concept phase (AP.1.1)
conciseness (QM.3.2.4.x)
concurrent engineering
Cleanroom Software Engineering
confidentiality (QM.2.1.4.1.2)
Intrusion Detection
conformance
COTS and Open Systems--An Overview
connected graph
Cyclomatic Complexity
connectivity software
Middleware
consistency (QM.1.3.2)
Algorithm Formalization
Requirements Tracing
constructive cost model
Function Point Analysis
context analysis
Feature-Oriented Domain Analysis
CORBA. see Common Object Request Broker Architecture
corrective maintenance (AP.1.9.3.1)
correctness (QM.1.3)
Cleanroom Software Engineering
cost estimation (AP.1.3.7)
Function Point Analysis
cost of maintenance (QM.5.1.2)
cost of operation (QM.5.1.1)

cost of ownership (QM.5.1)
COTS and Open Systems--An Overview
COTS. see commercial-off-the-shelf
cycle time
Cleanroom Software Engineering
Cyclomatic Complexity

D

data
analyzers (AP.1.4.3.4.x)
complexity
Cyclomatic Complexity
exchange
Object Request Broker
integrity
Database Two Phase Commit
Public Key Digital Signatures
management (AP.2.6.1)
management security (AP.2.4.2)
mining
Mainframe Server Software Architectures
recording (AP.2.9)
reduction (AP.2.9)
sharing
Application Programming Interface
visualization
Graphic Tools for Legacy Database Migration
warehouses
Mainframe Server Software Architectures
Data Encryption Standard
Simple Network Management Protocol
databases (AP.2.6)
Graphic Tools for Legacy Database Migration
administration (AP.1.9.1)
design (AP.1.3.2)
management
Three Tier Software Architecture
Two Tier Software Architectures
management system
Multi-Level Secure Database Management Schemes
server
Transaction Processing Monitor Technology
two phase commit
Database Two Phase Commit
utilities (AP.1.4.2.2)
DBMS. see database management system

debugger (AP.1.4.2.4.x)
decision support systems
Two Tier Software Architectures
defect
detection
Software Inspections
leakage
Software Inspections
management
Personal Software Process for Module-Level Development
prevention
Cleanroom Software Engineering
Defense Information Infrastructure Common Operating Environment. see Common Operating Environment
Defense Information Systems Agency
COTS and Open Systems--An Overview
TAFIM Reference Model
denial of service (QM.2.1.4.1.3)
Department of Defense systems
evolution of
TAFIM Reference Model
dependability (QM.2.1)
Rate Monotonic Analysis
DES. see Data Encryption Standard
design
Cleanroom Software Engineering
architectural (AP.1.3.1)
complexity
Cyclomatic Complexity
database (AP.1.3.2)
decision
history
Feature-Based Design Rationale Capture Method for Requirements Tracing
decisions
Feature-Based Design Rationale Capture Method for Requirements Tracing
detailed (AP.1.3.5)
interface (AP.1.3.3)
phase (AP.1.3)
rationale
Argument-Based Design Rationale Capture Methods for Requirements Tracing
Feature-Based Design Rationale Capture Method for Requirements Tracing
Requirements Tracing
capture
Argument-Based Design Rationale Capture Methods for

Requirements Tracing
history
Argument-Based Design Rationale Capture Methods for Requirements Tracing
detailed design (AP.1.3.5)
development phase
Cleanroom Software Engineering
Maintenance of Operational Systems--An Overview
digital signatures
Computer System Security--An Overview
Public Key Digital Signatures
DII COE. see Defense Information Infrastructure Common Operating Environment
directory services
Distributed Computing Environment
DISA. see Defense Information Systems Agency
diskless support
Distributed Computing Environment
distributed
business models
Distributed/Collaborative Enterprise Architectures
client/server architecture
Three Tier Software Architecture
computing (AP.2.1.2)
database system
Database Two Phase Commit
environment
TAFIM Reference Model
system
Distributed Computing Environment
Object Request Broker
Remote Procedure Call
services
Middleware
Distributed/Collaborative Enterprise Architectures
Client/Server Software Architectures
Distributed Computing Environment
Common Object Request Broker Architecture
Middleware
domain
Domain Engineering and Domain Analysis
analysis
Cleanroom Software Engineering
Domain Engineering and Domain Analysis
Feature-Oriented Domain Analysis
Organization Domain Modeling
design

engineering (AP.1.2.4)
Domain Engineering and Domain Analysis
Organization Domain Modeling
implementation
modeling
Feature-Oriented Domain Analysis
Organization Domain Modeling
Domain Engineering and Domain Analysis
dynamic binding
Object-Oriented Programming Languages

E

early operational phase
Maintenance of Operational Systems--An Overview
effectiveness (QM.1.1)
efficiency (QM.2.2)
Algorithm Formalization
Transaction Processing Monitor Technology
electronic encryption key distribution cryptography
Computer System Security--An Overview
end-to-end encryption
Computer System Security--An Overview
engineering function points
Function Point Analysis
Halstead Complexity Measures
entropy
Maintenance of Operational Systems--An Overview
error
handling (AP.2.11)
proneness (QM.2.3.1)
tolerance (QM.2.1.1.x)
essential complexity
Cyclomatic Complexity
estimating
Personal Software Process for Module-Level Development
event-driven applications
Message-Oriented Middleware
evolution/replacement phase
Maintenance of Operational Systems--An Overview
evolvability (QM.3.1.x)
expandability (QM.3.1.x)
extendability (QM.3.1.x)

F

fail safe (QM.2.1.1.x)
fail soft (QM.2.1.1.x)
FARS. see Federal Acquisition Regulations
FASA. see Federal Acquisition Streamlining Acts
fault
fault tolerance (QM.2.1.1.x)
feature analysis
    Feature-Oriented Domain Analysis
feature points
    Function Point Analysis
Feature-Based Design Rationale Capture Method for Requirements Tracing
Feature-Oriented Domain Analysis
Federal Acquisition Regulations
    COTS and Open Systems--An Overview
Federal Acquisition Streamlining Acts
    COTS and Open Systems--An Overview
fidelity (QM.2.4)
file systems
    Distributed Computing Environment
    support for
        Remote Procedure Call
file transfer
    Application Programming Interface
firewalls
    Computer System Security--An Overview
    Firewalls and Proxies
    proxies, and
        Firewalls and Proxies
fixed priority
    Rate Monotonic Analysis
flexibility (QM.3.1.x)
    Ada 83
    Ada 95
    Client/Server Software Architectures
    Component-Based Software Development/COTS Integration
    Distributed/Collaborative Enterprise Architectures
    Distributed Computing Environment
    Mainframe Server Software Architectures
    Message-Oriented Middleware
    Remote Procedure Call
    TAFIM Reference Model
    Three Tier Software Architecture
    Transaction Processing Monitor Technology
    Two Tier Software Architectures
FODA. see Feature-Oriented Domain Analysis

FORTEZZA
Computer System Security--An Overview
function call
Remote Procedure Call
Function Point Analysis
function points
early and easy
Function Point Analysis
functional scope (QM.4.4.1)
functional size measurement
Function Point Analysis
functional testing (AP.1.4.3.4.x)
functionality analysis
Maintenance of Operational Systems--An Overview
fundamental distributed services
Distributed Computing Environment
Futurebus+
Rate Monotonic Analysis

G

GCCS. see Global Command and Control System
GCSS. see Global Combat Support System
generality (QM.4.4.1.x)
Global Combat Support System
Defense Information Infrastructure Common Operating Environment
TAFIM Reference Model
Global Command and Control System
Defense Information Infrastructure Common Operating Environent
TAFIM Reference Model
Graphic Tools for Legacy Database Migration
graphical user interface
Graphical User Interface Builders
builders
Graphical User Interface Builders
graphics (AP.2.3.2)
GUI builders. see graphical user interface builders

H

Halstead complexity measures
Cyclomatic Complexity
Function Point Analysis
hardware maintenance

hardware-software co-design (AP.1.3.1.x)
Henry metrics
Cyclomatic Complexity
heterogeneous databases
Three Tier Software Architecture
homogeneous environment
Two Tier Software Architectures
human computer interaction (AP.2.3)
human engineering

I

IDTs. see interface development tools
IFPUG. see International Function Point User Group
implementation phase (AP.1.4)
implementations
overview of
Reference Models, Architectures, Implementations--An Overview
incompleteness (QM.1.3.1)
incremental development
Cleanroom Software Engineering
independence
Distributed Computing Environment
information
analysis
Feature-Oriented Domain Analysis
hiding
Object-Oriented Programming Languages
security (AP.2.4)
warfare
Intrusion Detection
Rule-Based Intrusion Detection
Statistical-Based Intrusion Detection
inheritance
Object-Oriented Programming Languages
installation and checkout phase (AP.1.8)
integration testing (AP.1.5.3.2)
integrity (QM.2.1.4.1.1)
Distributed Computing Environment
Intrusion Detection
interface
definition language
Common Object Request Broker Architecture
Distributed Computing Environment
design (AP.1.3.3)

development tools
Graphical User Interface Builders
specification
COTS and Open Systems--An Overview
standards
COTS and Open Systems--An Overview
testing (AP.1.5.3.3)
International Function Point User Group
Function Point Analysis
International Standards Organization
Ada 83
Function Point Analysis
standards
Distributed Computing Environment
internet
Firewalls and Proxies
Object Request Broker
standards
Distributed Computing Environment
interoperability (QM.4.1)
Ada 83
Ada 95
Application Programming Interface
Client/Server Software Architectures
Defense Information Infrastructure Common Operating Environment
Distributed Computing Environment
Message-Oriented Middleware
Middleware
Remote Procedure Call
TAFIM Reference Model
intranet
Computer System Security--An Overview
Firewalls and Proxies
Object Request Broker
Intrusion Detection
Computer System Security--An Overview
model-based
Rule-Based Intrusion Detection
rule-based
Rule-Based Intrusion Detection
statistical-based
Statistical-Based Intrusion Detection
ISO see International Standards Organization

J

Jacobson
Object-Oriented Analysis
Java
Ada 95
Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures
Object Request Broker
Joint Technical Architecture
Defense Information Infrastructure Common Operating Environment
JTA. see Joint Technical Architecture

K

Kafura metrics
Cyclomatic Complexity
kernel COE
Defense Information Infrastructure Common Operating Environment

L

latency (QM.2.2.2)
legacy systems
COTS and Open Systems--An Overview
Distributed Computing Environment
Domain Engineering and Domain Analysis
lifetime of operational capability
Ligier metrics
Cyclomatic Complexity
lines of code
Function Point Analysis
metrics
Halstead Complexity Measures
LOC. see lines of code

M

MAC. see message authentication code
Mainframe Server Software Architectures
maintainability (QM.3.1)
Ada 83
Ada 95
Cyclomatic Complexity
Domain Engineering and Domain Analysis

Feature-Oriented Domain Analysis
Graphic Tools for Legacy Database Migration
Maintainability Index Technique for Measuring Program Maintainability
Module Interconnection Languages
Object-Oriented Analysis
Object-Oriented Database
Object-Oriented Design
Object-Oriented Programming Languages
Organization Domain Modeling
Software Inspections
Three Tier Software Architecture

Maintainability Index Technique for Measuring Program Maintainability

maintenance
adaptive (AP.1.9.3.2)
control
corrective (AP.1.9.3.1)
costs
Maintenance of Operational Systems--An Overview
documentation
Maintenance of Operational Systems--An Overview
measures
metric
Halstead Complexity Measures
perfective (AP.1.9.3.3)
personnel

Maintenance of Operational Systems--An Overview

managed device
managed object
managed objects
Simple Network Management Protocol
management agent
management information base
Simple Network Management Protocol
manufacturing phase (AP.1.7)
marshalling
Component Object Model (COM), DCOM, and Related Capabilities
mature operational phase
Maintenance of Operational Systems--An Overview
mature systems
Maintenance of Operational Systems--An Overview
McCabe's complexity
Cyclomatic Complexity
message authentication code
Public Key Digital Signatures
message delivery
Application Programming Interface
message digest function

Public Key Digital Signatures

Message-Oriented Middleware

Middleware

Remote Procedure Call

metrics

Cyclomatic Complexity

Halstead

Cyclomatic Complexity

Function Point Analysis

Henry

Cyclomatic Complexity

McCabe

Cyclomatic Complexity

Troy

Cyclomatic Complexity

Zweben

Cyclomatic Complexity

MIB. see management information base

middle tier server

Three Tier Software Architecture

Middleware

Application Programming Interface

Message-Oriented Middleware

Object Request Broker

Transaction Processing Monitor Technology

minimal operator intervention

Two Tier Software Architecture

MISSI. see Multilevel Information Systems Security Initiative

MLS Host

Computer System Security--An Overview

MLS Operating System

Computer System Security--An Overview

MLS. see multi-level security

models (AP.2.1.1)

modifiability (QM.3.1.x)

Application Programming Interface

Cleanroom Software Engineering

Module Interconnection Languages

module-level development

Personal Software Process for Module-Level Development

MOM. see message-oriented middleware

moniker

Component Object Model (COM), DCOM, and Related Capabilities

Morris Worm

Virus Detection

Motif

Graphical User Interface Builders

Multilevel Information Systems Security Initiative
Computer System Security--An Overview
multi-level secure
database management schemes
Computer System Security--An Overview
Multi-Level Secure Database Management Schemes
guard
Computer System Security--An Overview
one way guard with random acknowledgement
Multi-Level Secure One Way Guard with Random Acknowledgment
systems
Computer System Security--An Overview
multi-level security
Multi-Level Secure Database Management Schemes
Trusted Operating System
multiplexing client transaction requests
Transaction Processing Monitor Technology

N

NDI. see non-developmental items
necessity of characteristics (QM.1.1.1)
need satisfaction measures
network
Application Programming Interface
Distributed Computing Environment
architecture
Middleware
hardware
Distributed Computing Environment
management (AP.2.2.2)
manager
Message-Oriented Middleware
overhead
Distributed Computing Environment
performance of
Remote Procedure Call
protocols
interface to
Remote Procedure Call
security
Firewalls and Proxies
network management
network management application
network management information
network management protocol

network management station
Network Management--An Overview
non-developmental items
COTS and Open Systems--An Overview
Nonrepudiation in Network Communications

O

object activation
Object Request Broker
Object Linking and Embedding
Component Object Model (COM), DCOM, and Related Capabilities
Object Management Architecture
Common Object Request Broker Architecture
Object Management Group
Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures
object model
Object-Oriented Analysis
Object-Oriented Database
object orientation
Object Request Broker
object-oriented
Cleanroom Software Engineering
Distributed Computing Environment
Remote Procedure Call
analysis
Object-Oriented Analysis
database
Object-Oriented Database
design
Object-Oriented Design
programming
Ada 83
Ada 95
programming language
Object-Oriented Programming Languages
systems
Message-Oriented Middleware
Object Request Broker
Client/Server Software Architectures
Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures
Middleware
objects
Object-Oriented Analysis

Object Request Broker

ODM. see organization domain modeling

one way guards
Computer System Security--An Overview

OOA. see object-oriented analysis

OOD. see object-oriented design

OODB. see object-oriented database

OOPL. see object-oriented programming languages

Open Group
Component Object Model (COM), DCOM, and Related Capabilities

open systems
Application Programming Interface
COTS and Open Systems--An Overview
Mainframe Server Software Architectures
cost
COTS and Open Systems--An Overview
COTS, and
COTS and Open Systems--An Overview
interconnect standards
Distributed Computing Environment

openness (QM.4.1.2)

operability (QM.2.3.2)

operational analysis
Feature-Oriented Domain Analysis

operational testing (AP.1.8.2.1)

operations
personnel
system

operations and maintenance phase (AP.1.9)

opportunistic reuse
Domain Engineering and Domain Analysis

ORB. see object request broker

Organization Domain Modeling

organizational measures

overview of reference models, architectures, implementations
Reference Models, Architectures, Implementations--An Overview

P

parallel computing (AP.2.1.3)

payload
Virus Detection

peer reviews
Software Inspections

perfective maintenance (AP.1.9.3.3)

performance
Graphic Tools for Legacy Database Migration
Rate Monotonic Analysis
Three Tier Software Architecture
measures
testing (AP.1.5.3.5)
periodic task/process
Rate Monotonic Analysis
persistent
data
Object-Oriented Database
objects
Object-Oriented Database
Personal Software Process
Personal Software Process for Module-Level Development
for module-level development
Personal Software Process for Module-Level Development
piecewise reengineering
Maintenance of Operational Systems--An Overview
pilot project
Cleanroom Software Engineering
plug-and-play
COTS and Open Systems--An Overview
polymorphism
Object-Oriented Programming Languages
portability (QM.4.2)
Ada 83
Ada 95
Defense Information Infrastructure Common Operating Environment
Distributed Computing Environment
Graphic Tools for Legacy Database Migration
POSIX
Rate Monotonic Analysis
pre-delivery phase
Maintenance of Operational Systems--An Overview
prepare phase
Database Two Phase Commit
priority inheritance
Rate Monotonic Analysis
priority inversion
Rate Monotonic Analysis
process management services
Three Tier Software Architecture
processing management

Two Tier Software Architectures
product line
Component-Based Software Development/COTS Integration
productivity (QM.5.2)
Function Point Analysis
Object-Oriented Analysis
rates
Function Point Analysis
profiles
Statistical-Based Intrusion Detection
programming language (AP.1.4.2.1)
proprietary interfaces
COTS and Open Systems--An Overview
protocols (AP.2.2.3)
COTS and Open Systems--An Overview
support of
Message-Oriented Middleware
provably correct (QM.1.3.4)
proxies
Computer System Security--An Overview
Firewalls and Proxies
PSP. see Personal Software Process
public key cryptography
Computer System Security--An Overview
Public Key Digital Signatures
Public Key Digital Signatures

Q

qualification phase (AP.1.6)
qualification testing (AP.1.6.1)
quality
Cleanroom Software Engineering
quality measures
Component-Based Software Development/COTS Integration
Personal Software Process for Module-Level Development
queuing theory
Rate Monotonic Analysis

R

Rate Monotonic Analysis

rate monotonic scheduling
- Rate Monotonic Analysis

rationale capture
- Argument-Based Design Rationale Capture Methods for Requirements Tracing
- Feature-Based Design Rationale Capture Method for Requirements Tracing

RBID. see Rule-Based Intrusion Detection

RDA. see remote data access

readability (QM.3.2.4)

real-time
- Rate Monotonic Analysis
- responsiveness
  - responsiveness (QM.2.2.2)
- systems
  - COTS and Open Systems--An Overview
  - Rate Monotonic Analysis

recovery (AP.2.10)

reengineering (AP.1.9.5)
- Cyclomatic Complexity
- Graphic Tools for Legacy Database Migration
- Graphical User Interface Builders
- Maintenance of Operational Systems--An Overview

reference models
- overview of
  - **Reference Models, Architectures, Implementations--An Overview**

regression testing (AP.1.5.3.4)

reliability (QM.2.1.2)
- Ada 83
- Ada 95
- Cleanroom Software Engineering
- Distributed/Collaborative Enterprise Architectures
- Software Inspections
- Transaction Processing Monitor Technology

remote data access
- Transaction Processing Monitor Technology

remote method invocation
- Object Request Broker

Remote Procedure Call
- Application Programming Interface
- Distributed Computing Environment

Message-Oriented Middleware
Middleware
Transaction Processing Monitor Technology

requirements
cross referencing
Requirements Tracing
engineering (AP.1.2.2)
growth (QM.5.1.2.6)
phase (AP.1.2)
tracing (AP.1.2.3)
Maintenance of Operational Systems--An Overview
Requirements Tracing

requirements-to-code (AP.1.2.3.1)
resource utilization (QM.2.2)
responsiveness (QM.1.2)
restart (AP.2.10)
restructuring
Maintenance of Operational Systems--An Overview
retirement phase (AP.1.10)
retrievability (QM.4.4.2)
reusability (QM.4.4)
Ada 83
Ada 95
Architecture Description Languages
Defense Information Infrastructure Common Operating Environment
Domain Engineering and Domain Analysis
Feature-Based Design Rationale Capture Method for Requirements Tracing
Feature-Oriented Domain Analysis
Mainframe Server Software Architectures
Object-Oriented Analysis
Object-Oriented Design
Organization Domain Modeling
Three Tier Software Architecture
Transaction Processing Monitor Technology

reuse
Component-Based Software Development/COTS Integration
Module Interconnection Languages

reverse-engineering (AP.1.9.4)
Maintenance of Operational Systems--An Overview
design recovery
Maintenance of Operational Systems--An Overview

REVIC. see revised intermediate COCOMO

revised intermediate COCOMO
Function Point Analysis
risk analysis
Cyclomatic Complexity
RMA. see rate monotonic analysis
RMI. see remote method invocation
robustness (QM.2.1.1)
RPC. see remote procedure call
Rule-Based Intrusion Detection
Rumbaugh
Object-Oriented Analysis
runtime environment
Defense Information Infrastructure Common Operating Environment

S

safety (QM.2.1.3)
scalability (QM.4.3)
Distributed/Collaborative Enterprise Architectures
Distributed Computing Environment
Mainframe Server Software Architectures
Three Tier Software Architecture
Two Tier Software Architectures
schedulability analysis
Rate Monotonic Analysis
scheduling
Rate Monotonic Analysis
security (QM.2.1.5)
Distributed Computing Environment
Firewalls and Proxies
Intrusion Detection
Multi-Level Secure Database Management Schemes
Public Key Digital Signatures
Trusted Operating Systems
security services
Distributed Computing Environment
segments
Defense Information Infrastructure Common Operating Environment
select or develop algorithms
self descriptiveness (QM.3.2.4.x)
server
Two Tier Software Architecture

session based technology
Transaction Processing Monitor Technology
sharing services
Distributed Computing Environment
Shlaer-Mellor
Object-Oriented Analysis
Simple Network Management Protocol
secure SNMP
Simple Network Management Protocol
SNMPv1
Simple Network Management Protocol
SNMPv2
Simple Network Management Protocol
Simplex Architecture
simplicity (QM.3.2.2)
Smalltalk
Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures
SNMP. see Simple Network Management Protocol
software
architecture (AP.2.1)
change cycle time
complexity
Maintenance of Operational Systems--An Overview
engineering
Personal Software Process for Module-Level Development
engineering tools
Graphical User Interface Builders
entropy
Maintenance of Operational Systems--An Overview
generation
Algorithm Formalization
inspections
Software Inspections
life cycle
Cleanroom Software Engineering
maintainability
Maintenance of Operational Systems--An Overview
maintenance (QM.5.1.2.5)
metrics
Cyclomatic Complexity
migration and evolution (AP.1.9.3.2)
process improvement
Personal Software Process for Module-Level Development
productivity

Function Point Analysis
synthesis
Algorithm Formalization
upgrade and technology insertion (AP.1.9.3.3)
Software Technology for Adaptable Reliable Systems
Cleanroom Software Engineering
Organization Domain Modeling
speed (QM.2.2.x)
SQL. see standard query language
standard query language
Application Programming Interface
STARS. see Software Technology for Adaptable Reliable Systems
static metrics
Cyclomatic Complexity
statistical quality control
Cleanroom Software Engineering
statistical testing (AP.1.5.3.5.x)
Cleanroom Software Engineering
Statistical-Based Intrusion Detection
structural complexity
Cyclomatic Complexity
structural testing (AP.1.4.3.4.x)
structuredness (QM.3.2.3)
sufficiency of characteristics (QM.1.1.2)
support requirements
Function Point Analysis
survivability (QM.2.1.4.1.4)
synchronous mechanism
Message-Oriented Middleware
Remote Procedure Call
synchronous processing
Distributed Computing Environment
system
administrators
Distributed Computing Environment
allocation (AP.1.2.1)
analysis and optimization (AP.1.3.6)
Cleanroom Software Engineering
change costs
Function Point Analysis
evolution
Component-Based Software Development/COTS Integration
integration
Component-Based Software Development/COTS Integration

lifecycle
Maintenance of Operational Systems--An Overview
migration
Graphic Tools for Legacy Database Migration
security (AP.2.4.3)
testing (AP.1.5.3.1)
system engineering
Domain Engineering and Domain Analysis
systematic reuse
Domain Engineering and Domain Analysis
Organization Domain Modeling

T

TAFIM
Defense Information Infrastructure Common Operating Environment
Application Program Interface
TAFIM Reference Model
External Environment Interface
TAFIM Reference Model
reference model
TAFIM Reference Model
tasks
Rate Monotonic Analysis
taxonomy
test (AP.1.4.3)
drivers (AP.1.4.3.2, AP.1.5.1)
generation
Maintenance of Operational Systems--An Overview
optimization
Maintenance of Operational Systems--An Overview
phase (AP.1.5)
planning
Cyclomatic Complexity
tools (AP.1.4.3.3, AP.1.5.2)
testability (QM.1.4.1)
testing (AP.1.5.3)
acceptance (AP.1.8.2.2)
black-box (AP.1.4.3.4.x)
component (AP.1.4.3.5)
functional (AP.1.4.3.4.x)
integration (AP.1.5.3.2)
interface (AP.1.5.3.3)

operational (AP.1.8.2.1)
performance (AP.1.5.3.5)
qualification (AP.1.6.1)
regression (AP.1.5.3.4)
statistical (AP.1.5.3.5.x)
Cleanroom Software Engineering
structural (AP.1.4.3.4.x)
system (AP.1.5.3.1)
unit (AP.1.4.3.4)
white-box (AP.1.4.3.4.x)
threads
Distributed Computing Environment
Rate Monotonic Analysis
services
Distributed Computing Environment
three tier
architecture
Mainframe Server Software Architectures
Three Tier Software Architecture
software architectures
Three Tier Software Architecture
client/server
Message-Oriented Middleware
with application server
Client/Server Software Architectures
with message server
Client/Server Software Architectures
with ORB architecture
Client/Server Software Architectures
throughput (QM.2.2.3)
Intrusion Detection
time services
Distributed Computing Environment
TP Heavy
Client/Server Software Architectures
TP Lite
Client/Server Software Architectures
TP monitor. see transaction processing monitor technology.
traceability (QM.1.3.3)
Requirements Tracing
training (QM.5.1.1.2), QM.5.1.2.2)
transaction applications
Transaction Processing Monitor Technology
Transaction Processing Monitor Technology

Client/Server Software Architectures
Middleware
translation
Maintainability Index Technique for Measuring Program Maintainability
Maintenance of Operational Systems--An Overview
restructuring/modularizing
Maintenance of Operational Systems--An Overview
transport software
Distributed Computing Environment
trouble report analysis (AP.1.9.2)
Troy metrics
Cyclomatic Complexity
Trusted Operating Systems (AP.2.4.1)
Multi-Level Secure Database Management Schemes
trustworthiness (QM.2.1.4)
Public Key Digital Signatures
two life cycle model
Domain Engineering and Domain Analysis
two phase commit technology
Database Two Phase Commit
two tier
architecture
Mainframe Server Software Architectures
software architectures
Two Tier Software Architectures

U

UDP. see user datagram protocol
UIL. see user interface language
UIMS. see user interface management system
understandability (QM.3.2)
Architecture Description Languages
Cleanroom Software Engineering
Domain Engineering and Domain Analysis
Graphic Tools for Legacy Database Migration
Module Interconnection Languages
Organization Domain Modeling
unit testing (AP.1.4.3.4)
UNIX
Mainframe Server Software Architectures
unmarshalling

Component Object Model (COM), DCOM, and Related Capabilities
upgradeability (QM.3.1.x)
usability (QM.2.3)
Client/Server Software Architectures
Domain Engineering and Domain Analysis
Graphical User Interface Builders
Two Tier Software Architectures
user datagram protocol
Simple Network Management Protocol
user interfaces (AP.2.3.1)
development tools
Graphical User Interface Builders
language
Graphical User Interface Builders
management system
Graphical User Interface Builders
user services
Three Tier Software Architecture
user system interface
Two Tier Software Architectures
user friendly interface
Two Tier Software Architectures

V

validation suite
Ada
Ada 83
Ada 95
variability
Domain Engineering and Domain Analysis
vendor-driven upgrades
Component-Based Software Development/COTS Integration
verifiability (QM.1.4)
Cleanroom Software Engineering
VHDL. see VHSIC Hardware Description Language
VHSIC Hardware Description Language
Architecture Description Languages
virus
Virus Detection
Virus Detection
Computer System Security--An Overview
visualization tool

Graphic Tools for Legacy Database Migration

vulnerability (QM.2.1.4.1)

W

walkthroughs

Software Inspections

white-box testing (AP.1.4.3.4.x)

widgets

Graphical User Interface Builders

workstation compliance level three

Defense Information Infrastructure Common Operating Environment

World Wide Web

Firewalls and Proxies

X

Y

Z

Zweben metrics

Cyclomatic Complexity

A| B| C| D| E| F| G| H| I| J| K| L| M| N| O| P| Q| R| S| T| U| V| W| X| Y| Z

URL: http://www.sei.cmu.edu/str/indexes/keywords/index_body.html

Last Modified: 24 July 2008

Reference and Glossary Items

Reference and glossary items are displayed in this frame.

Notes

[1] This spectrum of technologies includes past, present, under-used, and emerging technologies.

Feedback and Participation

Software Technology Roadmap

Thank you for taking time to provide information for the Software Technology Review!

Our goal is to keep this document current and to continually reflect the latest information. As a reader of this document, you can play a significant role in updating and expanding the information contained here. You can also help us by letting us know how you use the document and how it could be improved.

You Can Participate By...

- Critiquing or adding information to an existing technology description
- Commenting on the overall STR effort
- Submitting a new technology description

  Includes a list of potential topics for which authors are sought.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/feedback/index.html
Last Modified: 24 July 2008

Notes

[2] As an example of balanced coverage, let's briefly look at information hiding of object-oriented inheritance, which reduces the amount of information a software developer must understand. Substantial evidence exists that such object-oriented technologies significantly increase productivity in the early stages of software development; however, there is also growing recognition that these same technologies may also encourage larger and less efficient implementations, extend development schedules beyond the "90% complete" point, undermine maintainability, and preclude error free implementations.

Notes

[3] Similar to a roadmap for highways, the review prescribes neither the destination nor the most appropriate route. Instead, it identifies a variety of alternative routes that are available, gives an indication of their condition, and describes where they may lead. Specific DoD applications must chart their own route through the technological advances.

Section Explanation

Status. The status indicator provides an assessment of the overall quality and maturity of the technology description. One of four indicators is assigned by the STR staff: **Draft**, **In Review**, **Advanced**, or **Complete**.

Draft technology descriptions have the following attributes:

- They need more work.
- They have generally not been reviewed.
- Overall assessment: While technology descriptions labeled "Draft" will contain some useful information, readers should not rely on these descriptions as their only source of information about the topic. Readers should consider these descriptions as starting points for conducting their own research about the technology.

In Review technology descriptions have the following attributes:

- They are thought to be in fair technical shape.
- They have begun an internal review cycle[1].
- They may have major issues that must be resolved, or some sections that may require additional text.
- Relevant keywords have been added to the Keyword Index.
- Overall assessment: Readers can get some quality information from these, but because these descriptions have not been completely reviewed, readers should explore some of the references for additional information and consider conducting their own research about the technology.

Advanced technology descriptions have the following attributes:

- They are in good technical shape.
- Internal review has occurred.
- There are minor issues to be worked, but it is generally polished.
- They are subject to additional review by external reviewers.
- Relevant keywords have been added to the Keyword Index.
- Overall assessment: These descriptions are in rather good shape, but because they have not been through external review, readers should exercise some caution.
- Note: We encourage readers to critique Advanced technology descriptions, especially for content accuracy. Please see the feedback section for more details.

Complete technology descriptions have the following attributes:

- At least one expert external review has occurred, and issues from that review have been resolved.
- Relevant keywords have been added to the Keyword Index.
- No additional work is necessary at this time.
- Overall assessment: These technology descriptions are believed to be complete and correct. They would be revised in the future based on additional external reviewers, new information, and public feedback.
- Note: We encourage readers to critique Complete technology descriptions, especially for content accuracy. Please see the feedback section for more details.

[1] Internal review cycle refers to the review process that takes place within the development/editorial team.

Section Explanation

Note. This section appears if prerequisite or follow-on reading is recommended. The prerequisites are usually overviews of the general topic area that establish a context for the different technologies in the area.

Section Explanation

Purpose and Origin. This section provides a general description and brief background of the technology. It describes what capability or benefit was anticipated for the technology when originally conceived. It cites quality measures that are significantly influenced by the technology (these quality measures are italicized), and it identifies common aliases for the technology as well as its originator(s) or key developer(s) (if known).

Section Explanation

Technical Detail. This section answers -- succinctly -- the question "what does the technology do?" It describes the salient quality measures (see the Quality Measures Taxonomy) that are influenced by the technology in all situations and describes the tradeoffs that are enabled by the technology. It may also provide some insight into why the technology works and what advances are expected. Since the STR is not a "how-to" manual, no implementation details are provided.

Section Explanation

Usage Considerations. This section provides insight for the use of the technology. Issues that are addressed include

- example applications into which this technology may be incorporated (or should not be incorporated); for instance, "this technology, because of its emphasis on synchronized processing, is particularly suited for real-time applications"
- quality measures that may be influenced by this technology, depending on the particular context in which the application is employed

Section Explanation

Maturity. The purpose of this section is to provide an indication as to how well-developed the technology is. A technology that was developed a year or two ago and is still in the experimental stage (or still being developed at the university research level) will likely be more difficult to adopt than one that has been in use in many systems for a decade. It is not the intent of this document to provide an absolute measure of maturity, but to provide enough information *to allow the reader to make an informed judgment as to the technology's maturity for their application area*. Details that will help in this determination include

- the extent to which the technology has been incorporated into real systems, tools, or commercial products
- the success that developers have had in adopting and using the technology
- notable failures of the technology (if any)

Other information that might appear in this section includes trend information, such as a projection of the technology's long term potential, observations about the rate of maturation, and implications of rapid maturation.

Section Explanation

Costs and Limitations. No technology is right for every situation, and each technology has associated costs (monetary and otherwise). This section points out these limitations and costs. Some examples of the kinds of costs and limitations that a technology may possess are the following: a technology may impose an otherwise unnecessary interface standard; it might require investment in other technologies (see "Dependencies" below); it might require investment of time or money; or it may directly conflict with security or real-time requirements. Specific items of discussion include

- what is needed to adopt this technology (this could mean training requirements, skill levels needed, programming languages, or specific architectures)
- how long it takes to incorporate or implement this technology
- barriers to the use of this technology
- reasons why this technology would not be used

Section Explanation

Dependencies. This section identifies other technologies that influence or are influenced by the technology being described. The only dependencies mentioned are those where significant influence in either direction is expected. An indication as to why the dependency exists (usually in terms of quality measure or usage consideration) is also provided. If the dependent technology appears in the document, a cross-reference is provided. This paragraph is omitted if no dependencies are known.

Section Explanation

Alternatives. An alternative technology is one that could be used for the same purposes as the technology being described. A technology is an alternative if there is any situation or purpose for which both technologies are viable or likely to be considered candidates. Alternatives may represent a simple choice among technologies that achieve the same solution to a problem, or they may represent completely different approaches to the problem being addressed by the technology.

For each alternative technology, this section provides a concise description of the situations for which it provides an alternative. Also provided are any special considerations that could help in selecting among alternatives. If the alternative technology appears in the document, a cross-reference is provided.

Alternative technologies are distinct from dependent or complementary technologies, which must be used in combination with the technology being described to achieve the given purpose.

Section Explanation

Complementary Technologies. A complementary technology is one that enhances or is enhanced by the technology being described, but for which neither is critical to the development or use of the other (if it were critical, then it would appear in the "Dependencies" section above). Typically, a complementary technology is one that in combination with this technology will achieve benefits or capabilities that are not obvious when the technologies are considered separately. For each complementary technology, this section provides a concise description of the conditions under which it is complementary and the additional benefits that are provided by the combination. If the complementary technology appears in the document, a cross-reference is provided.

Section Explanation

Index Categories. This section provides keywords on which this technology may be indexed. Beside providing the name of the technology, it provides keywords in the following categories:

- *Application category.* This category refers to how this technology would be employed, either in support of operational systems (perhaps in a particular phase of the life cycle) or in actual operation of systems (for example, to provide system security).
- *Quality measures category.* This is a list of those quality attributes (e.g., reliability or responsiveness) that are influenced in some way by the application of this technology.
- *Computing Reviews category.* This category describes the technical subdiscipline within Computer Science into which the technology falls. The category is based on the *ACM Computing Reviews* Classification System developed in 1991 (and currently undergoing revision). A complete description of the Classification System and its contents can be found in any January issue of *Computing Surveys* or in the annual *ACM Guide to Computing Literature*.

Section Explanation

References and Information Sources. This section provides bibliographic information. We include sources cited in the technology description, as well as pointers to additional resources that a reader can go to for additional information. Typically several references are designated as key references by an asterisk (*). Key references are those that will, in our opinion, best assist one in learning more about the technology.

Section Explanation

Current Author/Maintainer. The author(s)/maintainer(s) of the current version of the technology description are listed in this section. The only exceptions are *Draft* technology descriptions, which will not have an author's name.

Section Explanation

External Reviewer. This section contains names of external experts who have reviewed this technology description. If no "External Reviewer(s)" heading is present, then an external review has not occurred.

Section Explanation

Modifications. This area lists the modification history of the technology description and includes the names of contributing authors from earlier versions.

Section Explanation

Pending. A known item that needs to be addressed in future versions of the description. These are posted (when known) so that the reader can pursue these items on their own if necessary.

Glossary Term

Portability

the ease with which a system or component can be transferred from one hardware or software environment to another [IEEE 90].

Glossary Term

Maintainability

the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment [IEEE 90].

Glossary Term

Flexibility

the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed [IEEE 90].

Glossary Term

Reliability

the ability of a system or component to perform its required functions under stated conditions for a specified period of time [IEEE 90].

Glossary Term

Interoperability

the ability of two or more systems or components to exchange information and to use the information that has been exchanged [IEEE 90].

References and Information Sources

[Lawlis 96] Lawlis, Patricia K. *Guidelines for Choosing a Computer Language: Support for the Visionary Organization* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/> (1996).

References and Information Sources

[AdaLRM 95] *Ada95 Language Reference Manual*, International Standard ISO/IEC 8652: 1995(E), Version 6.0 [online]. Available WWW <URL: http://www.adahome.com/rm95/> (1995).

Ada 83

Software Technology Roadmap

Status

COMPLETE

Purpose and Origin

Ada is a general-purpose, internationally-standardized computer programming language developed by the U.S. Department of Defense to help software designers and programmers develop large, reliable applications. The Ada language enhances *portability*, *maintainability*, *flexibility*, *reliability*, and provides *interoperability* by standardization. The Ada 83 (1983) version [ANSI 83] (international standard: ISO/IEC 8652: 1987) is considered object-based as opposed to object-oriented (see Object-Oriented Programming Languages) because it does not fully support inheritance or polymorphism [Lawlis 96].

Technical Detail

The Ada language supports principles of good software engineering and discourages poor practices by prohibiting them when possible. Features that support code clarity and encapsulation (use of packages, use of generic packages and subprograms with generic parameters, and private and limited private types) provide support for maintenance and *reusability*. Ada also features strong typing--stronger than C or C++.

The Ada 83 language is independent of any particular hardware or operating system; the interface to any given platform is defined in a specific "System" package. Ada features that support portability include the ability to define numerical types using system-independent declarations and the ability to encapsulate dependencies.

Ada compilers are validated against established written standards- all standard language features exist in every validated Ada compiler. To become validated, a compiler must comply with the Ada Compiler Validation Capability (ACVC) suite of tests. Because of language standardization and required compiler validation, Ada provides an extremely high degree of support for interoperability and portability.

Ada 83 includes features that can be used for object-based programming, but it stops short of providing full support for object-oriented programming (OOP); this

is partly because of concerns regarding runtime performance during Ada's development.

By requiring specifications such as type specifications, by performing consistency checks across separately compiled units, and by providing exception handling facilities, Ada 83 provides a high degree of reliability compared to other programming languages.

Ada 83 provides features such as tasking, type declarations, and low-level language features to give explicit support of concurrency and real-time processing. However, Ada 83 does not specify tasking and type declarations in such a way that the resulting performance can always be predicted; this has been a criticism of the language in certain application areas such as embedded, real-time systems.

Usage Considerations

Ada was originally developed to support embedded software systems, but it has proven to provide good support for real-time, computationally-intensive, communication, and information system domains [Lawlis 96].

When combined with static code analysis or formal proofs, Ada can be used in safety-critical systems. For example, Ada has successfully been used in the development of the control systems for the safety-critical Boeing 777 Aircraft [AdaIC 96].

When considering performance, benchmarks performed on both Ada and C software with language toolsets of equal quality and maturity found that the two languages execute equally efficiently- with Ada versions having a slight edge over C versions [Syiek 95]. The quality of the compiled code is determined mostly by the quality of the compiler and not by the language. The burden of optimization is somewhat automated in Ada, as opposed to languages like C, where it is manually performed by the programmer.

When attempting to interface Ada 83 with other languages, several technical issues must be addressed. In order for Ada to call subroutines written in another language, an Ada compiler must support the pragma interface for the other language and its compiler. Similarly, if another language must call Ada subroutines, that language's compiler may also need modifications. The data representation between Ada and the other language must be compatible. Also, the system runtime support environment may need to be modified so that space is not wasted by functionally redundant support software [Hefley 92].

Ada 83 has recently been superseded by Ada 95 (see Ada 95). This new version places the software community into a transition period. Among the issues to be considered in transitioning from Ada 83 to Ada 95 are the following:

- Ada 83 compiler validation status. Validation certificates for all validated Ada 83 compilers expire at the end of March 1998; this may affect maintenance on existing systems written in Ada 83.

- Ada 95 compiler capabilities and availability
- the developmental status of a particular system

The current "philosophy" is that unless a demonstrated need exists, current operational systems or systems currently in development using Ada 83 do not need to transition to Ada 95 [Engle 96]. Refer to the Ada 95 technology description for more information on transitioning from Ada 83 to Ada 95.

A significant resource that addresses management and technical issues surrounding the adoption of Ada is the Ada Adoption Handbook [Hefley 92].

Maturity

Ada 83, with over 700 validated compilers [Compilers 96], has been used on a wide variety of programs in embedded, real-time, communication, and information system domains. It is supported by many development environments. Over 4 million lines of Ada code were successfully used in developing the AN/BSY-2 and AN/BQG-5 systems of the Seawolf submarine- a large, extensive, embedded system [Holzer 96]. Ada has become the standard programming language for airborne systems at Boeing Commercial Airplane Group (BCAG). Boeing used Ada to build 60 percent of the systems on the Boeing 777, which represents 70% of the 2.5 million lines of developed code [Pehrson 96, ReuseIC 95].

Ada is increasingly being taught in schools- approximately 323 institutions and companies are teaching Ada- a trend of 25% growth per year in schools and courses; this indicates increased and continued acceptance of Ada as a programming language [AdaIC 96].

Costs and Limitations

In a study performed in 1994, it was found that for life-cycle costs, Ada was twice as cost effective as C [Zeigler 95].

Common perceptions and conventional wisdom regarding Ada 83 (and Ada 95) have been shown to be incorrect or only partially correct. These perceptions include the following:

- Ada is far too complex.
- Ada is too difficult to teach, to learn, to use.
- Ada is too expensive.
- Using Ada causes inefficiencies.
- Training in Ada is too expensive.
- Ada is old-fashioned.
- Ada is not object-oriented.
- Ada does not fit into COTS software.

Mangold examines these perceptions in some detail [Mangold 96].

Alternatives

Other programming languages to consider are Ada 95, C, C++, FORTRAN, COBOL, Pascal, Assembly Language, LISP, or Smalltalk.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Ada 83
Application category	Programming Language (AP.1.4.2.1), Compiler (AP.1.4.2.3)
Quality measures category	Reliability (QM.2.1.2), Maintainability (QM.3.1), Interoperability (QM.4.1), Portability (QM.4.2), Scalability (QM.4.3), Reusability (QM.4.4)
Computing reviews category	Programming Languages (D.3)

References and Information Sources

[AdaIC 96] *AdaIC NEWS* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/news/> (1996).

[ANSI 83] ANSI/MIL-STD-1815A-1983. *Reference Manual for the Ada Programming Language.* New York, NY: American National Standards Institute, Inc., 1983.

[Compilers 96] *Ada 83 Validated Compilers List* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/compilers/83val/83vcl.txt> (August 1996).

[Engle 96] Engle, Chuck. *Re[2]: Ada 83/Ada 95* [email to Gary Haines], [online]. Available email: ghaines@spacecom.af.mil (August 19, 1996).

[Halang 90] Halang, W.A. & Stoyenko, A.D. "Comparative Evaluation of High-Level Real-Time Programming Languages." *Real-Time Systems 2,* 4 (November 1990): 365-82.

[HBAP 96] *Ada Home: The Home of the Brave Ada Programmers (HBAP)* [online]. Available WWW <URL: http://lglwww.epfl.ch:80/Ada/> (1996).

[Hefley 92] Hefley, W.; Foreman, J.; Engle, C.; & Goodenough, J. *Ada Adoption Handbook: A Program Manager's Guide* Version 2.0 (CMU/SEI-92-TR-29, ADA258937). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.

[Holzer 96] Holzer, Robert. "Sea Trials Prompt U.S. Navy to Tout Seawolf Sub's Virtues," *Defense News 11,* 28 (July 15-20, 1996): 12.

[Lawlis 96] Lawlis, Patricia K. *Guidelines for Choosing a Computer Language: Support for the Visionary Organization* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/> (1996).

[Mangold 96] Mangold, K. "Ada95-An Approach to Overcome the Software Crisis?" 4-10. *Proceedings of Ada in Europe 1995.* Frankfurt, Germany, October 2-6, 1995. Berlin, Germany: Springer-Verlag, 1996.

[Pehrson 96] Pehrson, Ron J. *Software Development for the Boeing 777* [online]. Available WWW <URL: http://www.stsc.hill.af.mil/CrossTalk/1996/jan/Boein777.html> (1996).

[Poza 90] Poza, Hugo B. & Cupak Jr., John J. "Ada: The Better Language for Embedded Applications." *Journal of Electronic Defense 13,* 1 (January 1990): 47.

[ReuseIC 95] *Boeing 777: Flying High with Ada and Reuse* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/ReuseIC/pubs/flyers/boe-reus.shtml> (1995).

[Syiek 95] Syiek, David. "C vs. Ada: Arguing Performance Religion." *ACM Ada Letters 15,* 6 (November/December 1995): 67-9.

[Tang 92] Tang, L.S. "A Comparison of Ada and C++," 338-49. *Proceedings of TRI-Ada `92.* Orlando, FL, November 16-20, 1992. New York, NY: Association for Computing Machinery, 1992.

[Zeigler 95] Zeigler, Stephen F. *Comparing Development Costs of C and Ada* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/> (1995).

Current Author/Maintainer

Cory Vondrak, TRW, Redondo Beach, CA
Capt Gary Haines, AFMC SSSG

External Reviewers

John Goodenough, SEI

Modifications

20 June 97: updated URLs for [AdaIC 96] and [Pehrson 96].
10 Jan 97 (original)

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/ada83_body.html
Last Modified: 24 July 2008

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

References and Information Sources

[AdaIC 97a] *Validation and Evaluation Test Suites: The Ada compiler certification process* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/testing/> (1997).

References and Information Sources

[AdaIC 97b] Ada Compiler Validation Capability, Version 2.1 (ACVC 2.1) [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/compilers/acvc/95acvc/acvc2_1> (1997).

Glossary Term

Reusability

the degree to which a software module or other work product can be used in more than one computing program or software system [IEEE 90].

References and Information Sources

[Brosgol 93] Brosgol, Benjamin. "Object-Oriented Programming in Ada 9X." *Object Magazine 2*, 6 (March-April 1993): 64-65.

References and Information Sources

[Taylor 95] Taylor, B. *Ada Compatibility Guide* Version 6.0 [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/docs/compat-guide/compat-guide6-0.txt> (1995).

Notes

[1] From John Goodenough, SEI, in email to John Foreman, Re: Ada 95, August 16, 1996.

References and Information Sources

[AJPO 95] Ada Joint Program Office. *Ada 95 Adoption Handbook: A Guide to Investigating Ada 95 Adoption* Version 1.1. Falls Church, VA: Ada Joint Program Office, 1995.

References and Information Sources

[AJPO 94] Ada Joint Program Office. *Ada 9X Transition Planning Guide: A Living Document and Working Guide for PEOs and PMs* Version 1.0. Falls Church, VA: Ada Joint Program Office, 1994.

References and Information Sources

[Patton 95] Patton II, I. Lee. "Early Experiences Adopting Ada 95," 426-34. *Proceedings of TRI-Ada '95*. Anaheim, CA, November 5-10, 1995. New York, NY: Association for Computing Machinery, 1995.

References and Information Sources

[AdaIC 95] AdaIC News Brief: November 3, 1995 [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/news/weekly/1995/95-11-03.html > (1995).

References and Information Sources

[AdaIC 96a] *AdaIC NEWS* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/news/> (1996).

References and Information Sources

[Wheeler 96] Wheeler, David A. *Java and Ada* [online]. Available WWW <URL: http://www.adahome.com/Tutorials/Lovelace/java.htm> (1996).

References and Information Sources

[Compilers 97] *Ada 95 Validated Compilers List* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/cgi-bin/vcl/report95.pl> (1997).

References and Information Sources

[Mangold 96] Mangold, K. "Ada95-An Approach to Overcome the Software Crisis?" 4-10. *Proceedings of Ada in Europe 1995*. Frankfurt, Germany, October 2-6, 1995. Berlin, Germany: Springer-Verlag, 1996.

Ada 95

Software Technology Roadmap

Status

Complete

Purpose and Origin

Ada is a general-purpose, internationally-standardized computer programming language developed by the U.S. Department of Defense (DoD) to help software designers and programmers develop large, reliable applications. The Ada language enhances *portability*, *maintainability*, *flexibility*, *reliability*, and provides *interoperability* by standardization [Lawlis 96].

The Ada 95 (1995) version [AdaLRM 95] supersedes the 1983 standard Ada 83. It corrects some shortcomings uncovered from nearly a decade of using Ada 83, and exploits developments in software technology that were not sufficiently mature at the time of Ada's original design. Specifically, Ada 95 provides extensive support for object-oriented programming (OOP) (see Object-Oriented Programming Languages), efficient real-time concurrent programming, improved facilities for programming in the large, and increased ability to interface with code written in other languages.

When distinguishing between the two versions of the language, the 1983 version is referred to as Ada 83, and the revised version is referred to as Ada or Ada 95.

Technical Detail

Ada 95 consists of a core language that must be supported by all validated compilers, and a set of specialized needs annexes that may or may not be implemented by a specific compiler. However, if a compiler supports a special needs annex, all features of the annex must be supported. The following is the set of annexes [AdaLRM 95]:

Required annexes (i.e., part of core language)
A. Predefined Language Environment
B. Interface to Other Languages
J. Obsolescent Features

Optional special needs annexes

C. Systems Programming
D. Real-time Programming
E. Distributed Systems
F. Information Systems
G. Numerics
H. Safety and Security

Annexes K - P are for informational purposes only and are not part of the standard.

As in Ada 83, Ada 95 compilers are validated against established written standards- all standard language features exist in every validated Ada compiler. To become validated, a compiler must comply with the Ada Compiler Validation Capability (ACVC) suite of tests [AdaIC 97a, 97b]. Because of language standardization and required compiler validation, Ada provides an extremely high degree of support for interoperability and portability.

Like Ada 83, the Ada 95 language is independent of any particular hardware or operating system; the interface to any given platform is defined in a specific "System" package. Ada 95 improves on the Ada 83 features that support portability, which include the ability to define numerical types using system-independent declarations and the ability to encapsulate dependencies.

By requiring specifications such as type specifications, by performing consistency checks across separately compiled units, and by providing exception handling facilities, Ada 95, like Ada 83, provides a high degree of reliability when compared to other programming languages.

The Ada language was developed explicitly to support software engineering- it supports principles of good software engineering and discourages poor practices by prohibiting them where possible. Features supporting code clarity and encapsulation (use of packages, use of generic packages and subprograms with generic parameters, and private and limited private types) provide support for maintenance and *reusability*. Ada 95 also provides full support for object-oriented programming, which allows for a high level of reusability:

- encapsulation of objects and their operations
- OOP inheritance- allowing new abstractions to be built from existing ones by inheriting their properties at either compile time or runtime
- an explicit pointer approach to polymorphism- the programmer must decide to use pointers to represent objects [Brosgol 93]
- dynamic binding

Ada 95 also provides special features (hierarchical libraries and partitions) to assist in the development of very large and distributed software components and systems.

Ada 95 improves on the flexibility provided by Ada 83 for interfacing with other programming languages by better standardizing the interface mechanism and

providing an Interface to Other Languages Annex.

Ada 95 improves the specification of previous Ada features that explicitly support concurrency and real-time processing, such as tasking, type declarations, and low-level language features. A Real-Time Programming Annex has been added to better specify the language definition and model for concurrency. Ada 95 has paid careful attention to avoid runtime overhead for the new object-oriented programming (OOP) features and incurs runtime costs commensurate with the generality actually used. Ada 95 also provides the flexibility for the programmer to specify the desired storage reclamation technique that is desired for the application.

Usage Considerations

Ada 95 is essentially an upwardly-compatible extension to Ada 83 with improved support for embedded software systems, real-time systems, computationally-intensive systems, communication systems, and information systems [Lawlis 96]. In revising Ada 83 to Ada 95, incompatibilities were catalogued, tracked, and assessed by the standard revision committee [Taylor 95]. These incompatibilities have proven to be mostly of academic interest, and they have not been a problem in practice.[1]

Combined with at least static code analysis or formal proofs, Ada 95, like Ada 83, is particularly appropriate for use in safety-critical systems.

The Ada Joint Program Office (AJPO) supports Ada 95 by providing an Ada 95 Adoption Handbook [AJPO 95] and an Ada 95 Transition Planning Guide [AJPO 94], and helping form Ada 95 early adoption partnerships with DoD and commercial organizations. The Handbook helps managers understand and assess the transition from Ada 83 to Ada 95 and the Transition Guide is designed to assist managers in developing a transition plan tailored for individual projects [Patton 95]. Another valuable source for Ada 95 training is a multimedia CD-ROM titled *Discovering Ada*. This CD-ROM contains tutorial information, demo programs, and video clips [AdaIC 95].

Ada 95 is the standard programming language for new DoD systems; the use of any other language would require a waiver. Early DoD adoption partnerships who are working Ada 95 projects include the Marine Corps Tactical Systems Support Activity (MCTSSA), Naval Research and Development (NRAD), and the Joint Strike Fighter (JSF) aircraft program [AdaIC 96a].

The AJPO supported the creation of an Ada 95-to-Java J-code compiler. This means that Java programs can be created by using Ada. The compiler generates Java "class" files just as a Java language compiler does. Ada and Java components can even call each other [Wheeler 96]. This capability gives Ada, like Java, extensive portability across platforms and allows Internet programmers to take advantage of Ada 95 features unavailable in Java.

Maturity

On February 15, 1995, Ada 95 became the first internationally-standardized object-oriented programming language. As of April 1997, 51 validated compilers were available [Compilers 97]. The current validation suite (Version 2.1) provides the capability to validate the core language as well as the additional features in the annexes [AdaIC 97b].

Results from early projects, such as the Joint Automated Message Editing Software (JAMES) and Airfields [AdaIC 96a], indicate that Ada 95 is upwardly-compatible with Ada 83 and that some Ada 95 compilers are mature and stable enough to use on fielded projects [Patton 95]. However, as of the spring of 1996, Ada 95 tool sets and development environments were, in general, still rather immature as compared to Ada 83 versions. As such, platform compatibility, bindings (i.e., database, user interface, network interface) availability, and tool support should be closely evaluated when considering Ada 95 compilers.

Costs and Limitations

Common perceptions and conventional wisdom regarding Ada 83 and Ada 95 have been shown to be incorrect or only partially correct. These perceptions include the following:

- Ada is far too complex.
- Ada is too difficult to teach, to learn, to use.
- Ada is too expensive.
- Using Ada causes inefficiencies.
- Training in Ada is too expensive.
- Ada is old-fashioned.
- Ada is not object-oriented.
- Ada does not fit into COTS software.

Mangold examines these perceptions in some detail [Mangold 96].

Alternatives

Other programming languages to consider are Ada 83, C, C++, FORTRAN, COBOL, Pascal, Assembly Language, LISP, Smalltalk, or Java.

Complementary Technologies

The Ada-95-to-Java J-code compiler (discussed in Usage Considerations) enables applications for the Internet to be developed in Ada 95.

Index Categories

This technology is classified under the following categories. Select a category for

a list of related topics.

Name of technology	Ada 95
Application category	Programming Language (AP.1.4.2.1), Compiler (AP.1.4.2.3)
Quality measures category	Reliability (QM.2.1.2), Maintainability (QM.3.1), Interoperability (QM.4.1), Portability (QM.4.2), Scalability (QM.4.3), Reusability (QM.4.4)
Computing reviews category	Programming Languages (D.3)

References and Information Sources

[AdaLRM 95] *Ada95 Language Reference Manual*, International Standard ISO/IEC 8652: 1995(E), Version 6.0 [online]. Available WWW <URL: http://www.adahome.com/rm95/> (1995).

[AdaIC 95] AdaIC News Brief: November 3, 1995 [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/news/weekly/1995/95-11-03.html >(1995).

[AdaIC 96a] *AdaIC NEWS* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/news/> (1996).

[AdaIC 97a] *Validation and Evaluation Test Suites: The Ada compiler certification process* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/testing/> (1997).

[AdaIC 97b] Ada Compiler Validation Capability, Version 2.1 (ACVC 2.1) [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/compilers/acvc/95acvc/acvc2_1> (1997).

[AJPO 94] Ada Joint Program Office. *Ada 9X Transition Planning Guide: A Living Document and Working Guide for PEOs and PMs* Version 1.0. Falls Church, VA: Ada Joint Program Office, 1994.

[AJPO 95] Ada Joint Program Office. *Ada 95 Adoption Handbook: A Guide to Investigating Ada 95 Adoption* Version 1.1. Falls Church, VA: Ada Joint Program Office, 1995.

[Brosgol 93] Brosgol, Benjamin. "Object-Oriented Programming in Ada 9X." *Object Magazine 2*, 6 (March-April 1993): 64-65.

[Compilers 97] *Ada 95 Validated Compilers List* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/cgi-bin/vcl/report95.pl> (1997).

[HBAP 96] *Ada Home: The Home of the Brave Ada Programmers (HBAP)* [online]. Available WWW <URL: http://lglwww.epfl.ch:80/Ada/> (1996).

[Lawlis 96] Lawlis, Patricia K. *Guidelines for Choosing a Computer Language: Support for the Visionary Organization* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/> (1996).

[Mangold 96] Mangold, K. "Ada95-An Approach to Overcome the Software Crisis?" 4-10. *Proceedings of Ada in Europe 1995*. Frankfurt, Germany, October 2-6, 1995. Berlin, Germany: Springer-Verlag, 1996.

[Patton 95] Patton II, I. Lee. "Early Experiences Adopting Ada 95," 426-34. *Proceedings of TRI-Ada '95*. Anaheim, CA, November 5-10, 1995. New York, NY: Association for Computing Machinery, 1995.

[Taylor 95] Taylor, B. *Ada Compatibility Guide* Version 6.0 [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/docs/compat-guide/compat-guide6-0.txt> (1995).

[Tokar 96] Tokar, Joyce L. "Ada 95: The Language for the 90's and Beyond." *Object Magazine 6*, 4 (June 1996): 53-56.

[Wheeler 96] Wheeler, David A. *Java and Ada* [online]. Available WWW <URL: http://www.adahome.com/Tutorials/Lovelace/java.htm> (1996).

Current Author/Maintainer

Capt Gary Haines, AFMC SSSG
Cory Vondrak, TRW, Redondo Beach, CA

External Reviewers

Charles (Chuck) Engle (former AJPO director)

John Goodenough, SEI

Modifications

2 October 97: updated URL for [Compilers 97].
20 June 97: updated URLs for [AdaIC 96a] and [AdaLRM 95].
14 April 97: updated number of validated Ada compilers and validation suite information.
10 Jan 97 (original)

Pending

In March 1997, changes to Ada policy were directed by Mr. Emmett Page (ASD/C31). This technology does *not* reflect those changes.

A revised assessment of toolset maturity (see Maturity section) is also needed.

Footnotes

[1] From John Goodenough, SEI, in email to John Foreman, Re: Ada 95, August 16, 1996.

URL: http://www.sei.cmu.edu/str/descriptions/ada95_body.html
Last Modified: 24 July 2008

Related Topics

Programming Language (AP.1.4.2.1)

- Ada 83
- Ada 95
- Assembly
- Basic
- C
- C++
- COBOL
- Common LISP Object System (CLOS)
- Eiffel
- FORTRAN
- HTML
- Java
- LISP
- Motif User Interface Language (UIL)
- Object-Oriented Programming Languages
- Object Pascal
- Objective C
- Pascal
- PERL
- Simula
- Smalltalk
- TCL

Related Topics

Compiler (AP.1.4.2.3)

- Ada 83
- Ada 95
- Architecture Description Languages
- Assembly
- Basic
- C
- C++
- COBOL
- FORTRAN
- Java
- Module Interconnection Languages
- Object Pascal
- Objective C
- Pascal

Related Topics

Reliability (QM.2.1.2)

- Ada 83
- Ada 95
- Cleanroom Software Engineering
- Distributed/Collaborative Enterprise Architecture
- Personal Software Process for Module-Level Development
- Rate Monotonic Analysis
- Simplex Architecture
- Software Inspections
- Three Tier Software Architecture

Related Topics

Maintainability (QM.3.1)

- Ada 83
- Ada 95
- Application Programming Interface
- Argument-Based Design Rationale Capture Methods for Requirements Tracing
- Cleanroom Software Engineering
- Client/Server Software Architectures
- Common Management Information Protocol
- Common Object Request Broker Architecture
- Component-Based Software Development/ COTS Integration
- Component Object Model (COM), DCOM, and Related Capabilities
- COTS and Open Systems - An Overview
- Cyclomatic Complexity
- Distributed/Collaborative Enterprise Architectures
- Distributed Computing Environment
- Domain Engineering and Domain Analysis
- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Feature-Oriented Domain Analysis
- Graphic Tools for Legacy Database Migration
- Graphical User Interface Builders
- Halstead Complexity Measures
- Java
- Mainframe Server Software Architectures
- Maintainability Index Technique for Measuring Program Maintainability
- Maintenance of Operational Systems - An Overview
- Message-Oriented Middleware
- Network Management -- An Overview
- Object-Oriented Analysis
- Object-Oriented Database
- Object-Oriented Design
- Object-Oriented Programming Languages
- Object Request Broker
- Organization Domain Modeling
- Rate Monotonic Analysis
- Reference Models, Architectures, Implementations -- an Overview

- Remote Procedure Call
- Requirements Tracing
- Simple Network Management Protocol
- Simplex Architecture
- Software Inspections
- TAFIM Reference Model
- Three Tier Software Architectures
- Transaction Processing Monitor Technology
- Two Tier Software Architectures

Related Topics

Interoperability (QM.4.1)

- Ada 83
- Ada 95
- Application Programming Interface
- Client/Server Software Architectures
- Common Object Request Broker Architecture
- Component Object Model (COM), DCOM, and Related Capabilities
- COTS and Open Systems - An Overview
- Defense Information Infrastructure Common Operating Environment
- Distributed Computing Environment
- Java
- Message-Oriented Middleware
- Middleware
- Network Management -- An Overview
- Object Request Broker
- Reference Models, Architectures, Implementations -- An Overview
- Remote Procedure Call
- TAFIM Reference Model

Related Topics

Portability (QM.4.2)

- Ada 83
- Ada 95
- Common Object Request Broker Architecture
- Defense Information Infrastructure Common Operating Environment
- Distributed Computing Environment
- Java
- Message-Oriented Middleware
- Reference Models, Architectures, Implementations -- An Overview
- Remote Procedure Call

Related Topics

Scalability (QM.4.3)

- Ada 83
- Ada 95
- Client/Server Software Architectures
- Common Management Information Protocol
- Common Object Request Broker Architecture
- Distributed/Collaborative Enterprise Architectures
- Distributed Computing Environment
- Mainframe Server Software Architectures
- Network Management -- An Overview
- Simple Network Management Protocol
- Three Tier Software Architectures
- Two Tier Software Architectures

Related Topics

Reusability (QM.4.4)

- Ada 83
- Ada 95
- Architecture Description Languages
- Argument-Based Design Rationale Capture Methods for Requirements Tracing
- Common Object Request Broker Architecture
- Component Object Model (COM), DCOM, and Related Capabilities
- Defense Information Infrastructure Common Operating Environment
- Domain Engineering and Domain Analysis
- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Feature-Oriented Domain Analysis
- Mainframe Server Software Architectures
- Module Interconnection Languages
- Object-Oriented Analysis
- Object-Oriented Design
- Organization Domain Modeling
- Requirements Tracing
- Three Tier Software Architectures
- Transaction Processing Monitor Technology

Section Explanation

Footnotes. This section contains all of the footnotes used in this technology description. Individual footnotes can also be viewed in the lower frame by selecting a superscript number in the upper frame.

Glossary Term

Consistency

the degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component [IEEE 90].

References and Information Sources

[Smith 93b] Smith, Douglas R. "Derivation of Parallel Sorting Algorithms," 55-69. *Parallel Algorithm Derivation and Program Transformation*. New York, NY: Kluwer Academic Publishers, 1993.

Glossary Term

Efficiency

the degree to which a system or component performs its designated functions with minimum consumption of resources (CPU, Memory, I/O, Peripherals, Networks) [IEEE 90].

References and Information Sources

[Smith 90] Smith, Douglas R. & Lowry, Michael R. "Algorithm Theories and Design Tactics." *Science of Computer Programming 14*, 2-3 (1990): 305-321.

References and Information Sources

[Smith 86] Smith, Douglas R. "Top-Down Synthesis of Divide-and-Conquer Algorithms," 35-61. *Readings in Artificial Intelligence and Software Engineering*. Palo Alto, CA: Morgan Kaufmann, 1986.

References and Information Sources

[Smith 91] Smith, Douglas R. "KIDS-A Knowledge-Based Software Development System," 483-514. *Automating Software Design*. Menlo Park, CA: AAAI Press, 1991.

References and Information Sources

[Smith 93c] Smith, Douglas R. "Transformational Approach to Transportation Scheduling," 60-68. *Proceedings of the Eighth Knowledge-Based Software Engineering Conference*. Chicago, IL, September 20-23, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

Related Topics

Select or Develop Algorithms (AP.1.3.4)

- Algebraic Specification Techniques
- Algorithm Formalization
- Component-Based Software Development / COTS Integration
- Resolution-Based Theorem Proving
- Software Generation Systems

Related Topics

Consistency (QM.1.3.2)

- Algorithm Formalization
- Argument-Based Design Rationale Capture Methods for Requirements Tracing
- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Requirements Tracing

Related Topics

Provably Correct (QM.1.3.4)

- Algorithm Formalization

Related Topics

Throughput (QM.2.2.3)

- Algorithm Formalization
- Distributed Computing Environment
- Graphic Tools for Legacy Database Migration

Glossary Term

Modifiability

the degree to which a system or component facilitates the incorporation of changes, once the nature of the desired change has been determined [Boehm 78].

References and Information Sources

[Krechmer 92] Krechmer, K. "Interface APIs for Wide Area Networks." *Business Communications Review 22,* 11 (November 1992): 72-4.

References and Information Sources

[Hines 96] Hines, John R. "Software Engineering." *IEEE Spectrum* (January 1996): 60-64.

References and Information Sources

[King 95] King, Steven S. "Message Delivery APIs: The Message is the Medium." *Data Communications 21,* 6 (April 1995): 85-90.

Related Topics

Application Program Interfaces (APIs) (AP.2.7)

- Application Programming Interface
- Java

Notes

[1] While definitions of *architecture*, *component*, and *connector* vary among researchers, this definition of architecture serves as a baseline for this technology description. A generally accepted definition describing the difference between a "design" and an "architecture" is that while a design explicitly addresses functional requirements, an architecture explicitly addresses functional and non-functional requirements such as reusability, maintainability, portability, interoperability, testability, efficiency, and fault-tolerance [Paulisch 94].

Notes

[2] Source: Garlan, David, et al. "ACME: An Architecture Interchange Language." Submitted for publication.

References and Information Sources

[Garlan 93] Garlan, David & Shaw, Mary. "An Introduction to Software Architecture," 1-39. *Advances in Software Engineering and Knowledge Engineering* Volume 2. New York, NY: World Scientific Press, 1993.

Glossary Term

Understandability

the degree to which the purpose of the system or component is clear to the evaluator [Boehm 78].

References and Information Sources

[Garlan 94a] Garlan, D. & Allen, R. "Formalizing Architectural Connection," 71-80. *Proceedings of the 16th International Conference on Software Engineering*. Sorrento, Italy, May 16-21, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

References and Information Sources

[Luckham 95] Luckham, David C., et al. "Specification and Analysis of System Architecture Using Rapide." *IEEE Transactions on Software Engineering 21*, 6 (April 1995): 336-355.

References and Information Sources

[Shaw 95] Shaw, Mary, et al. "Abstractions for Software Architecture and Tools to Support Them." *IEEE Transactions on Software Engineering 21*, 6 (April 1995): 314-335.

References and Information Sources

[Hoare 85] Hoare, C.A.R. *Communicating Sequential Processes*. Englewood Cliffs, NJ: Prentice Hall International, 1985.

Architecture Description Languages

Software Technology Roadmap

Status

Complete

Purpose and Origin

When describing a computer software system, software engineers often talk about the *architecture* of the system, where an architecture is generally considered to consist of components and the connectors (interactions) between them.[1] Although architectural descriptions are playing an increasingly important role in the ability of software engineers to describe and understand software systems, these abstract descriptions are often informal and ad hoc.[2] As a result

- Architectural designs are often poorly understood and not amenable to formal analysis or simulation.
- Architectural design decisions are based more on default than on solid engineering principles.
- Architectural constraints assumed in the initial design are not enforced as the system evolves.
- There are few tools to help the architectural designers with their tasks [Garlan 93].

In an effort to address these problems, formal languages for representing and reasoning about software architecture have been developed. These languages, called architecture description languages (ADLs), seek to increase the *understandability* and *reusability* of architectural designs, and enable greater degrees of analysis.

Technical Detail

In contrast to Module Interconnection Languages (MILS), which only describe the structure of an implemented system, ADLs are used to define and model system architecture prior to system implementation. Further, ADLs typically address much more than system structure. In addition to identifying the components and connectors of a system, ADLs typically address:

- *Component behavioral specification.* Unlike MILs, ADLs are concerned with component functionality. ADLs typically provide support for

specifying both functional and non-functional characteristics of components. (Non-functional requirements include those associated with safety, security, reliability, and performance.) Depending on the ADL, timing constraints, properties of component inputs and outputs, and data accuracy may all be specified.

- *Component protocol specification.* Some ADLs, such as Wright [Garlan 94a] and Rapide [Luckham 95], support the specification of relatively complex component communication protocols. Other ADLs, such as UniCon [Shaw 95], allow the type of a component to be specified (e.g., filter, process, etc.) which in turn restricts the type of connector that can be used with it.
- *Connector specification.* ADLs contain structures for specifying properties of connectors, where connectors are used to define interactions between components. In Rapide, connector specifications take the form of partially-ordered event sequences, while in Wright, connector specifications are expressed using Hoare's Communicating Sequential Processes (CSP) language [Hoare 85].

As an example, consider the component shown in Figure 1. This component defines two data types, two operations (op), and an input and an output communication port. The component also includes specifications constraining the behavior of its two operations.

```
Component Simple is
      type in_type is ...
      type out_type is ...
      op f : in_type -> out_type
      op valid_input? : in_type -> Boolean
      port input_port : in_type
      port output_port : out_type
      axiom f-specification is
        (behavioral specification for the operation f)
      end axiom
      axiom valid_input?-specification is
        (behavioral specification for the operation
        valid_input?)
      end axiom
      interface is input_port!(x) ->
        ((output_port!f(x) -> Skip)
        < valid_input?(x) >
        (output_port!(Invalid_Data) -> Skip))
end Simple
```

Figure 1: Component

A protocol specification for this component, written in CSP, defines how it interacts with its environment. Specifically, component Simple will accept a data value x of type in_type on its input port, and, if the data value is valid, will output f (x) on its output port. If the data value is not valid, Simple will output an error message on its output port. Note that component Simple is a specification, not an implementation. Implementations of ADL components and connectors are expressed in traditional programming languages such as Ada (see Ada 83 and Ada 95) or C. Facilities for associating implementations with ADL entities vary between ADLs.

Usage Considerations

ADLs were developed to address a need that arose from programming in the large; they are well-suited for representing the architecture of a system or family of systems. Because of this emphasis, several changes to current system development practices may occur:

- *Training*. ADLs are formal, compilable languages that support one or more architectural styles; developers will need training to understand and use ADL technology and architectural concepts/styles effectively (e.g., the use of dataflow, layered, or blackboard architectural styles).
- *Change/emphasis in life-cycle phases*. The paradigm currently used for system development and maintenance may be affected. Specifically, architectural design and analysis will precede code development; results of analysis may be used to alter system architecture. As such, a growing role for ADLs is expected in evaluating competing proposed systems during acquisitions. An ADL specification should provide a good basis for programming activities [Shaw 95].
- *Documentation.* Because the structure of a software system can be explicitly represented in an ADL specification, separate documentation describing software structure is not necessary. This implies that if ADLs are used to define system structure, the architectural documentation of a given system will not become out of date.[3] Additionally, ADLs document system properties in a formal and rigorous way. These formal characterizations can be used to analyze system properties statically and dynamically. For example, dynamic simulation of Rapide [Luckham 95] specifications can be analyzed by automated tools to identify such things as communication bottlenecks and constraint violations. Further, these formal characterizations provide information that can be used to guide reuse.
- *Expanding scope of architecture*. ADLs are not limited to describing the software architecture; application to system architecture (to include hardware, software, and people) is also a significant opportunity.

Maturity

Several ADLs have been defined and implemented that support a variety of architectural styles, including

- Aesop, which supports the specification and analysis of architectural styles (formal characterizations of common architectures such as pipe and filters, and client-server) [Garlan 94b].
- Rapide, which uses event posets to specify component interfaces and component interaction [Luckham 95].
- Wright, which supports the specification and analysis of communication protocols [Garlan 94a].
- MetaH, which was developed for the real-time avionics domain [Vestal 96].
- LILEAnna, which is designed for use with Ada and generalizes Ada's

notion of generics [Tracz 93].
- UniCon, which addresses packaging and functional issues associated with components [Shaw 95].

Further information about these and other languages used to describe software architectures can be found in the *Software Architecture Technology Guide* and *Architectural Description Languages* [SATG 96, SEI 96].

Because ADLs are an emerging technology, there is little evidence in the published literature of successful commercial application. However, Rapide and UniCon have been used on various problems,[4] and MetaH appears to be in use in a commercial setting [Vestal 96]. ADLs often have graphical tools that are similar to CASE tools.

Costs and Limitations

The lack of a common semantic model coupled with differing design goals for various ADLs complicates the ability to share tool suites between them. Researchers are addressing this problem; an ADL called ACME is being developed with the goal that it will serve as an architecture interchange language.[5] Some ADLs, such as MetaH, are domain-specific.

In addition, support for asynchronous versus synchronous communication protocols varies between ADLs, as does the ability to express complex component interactions.

Dependencies

Simulation technology is required by those ADLs supporting event-based protocol specification.

Alternatives

The alternatives to ADLs include Module Interconnection Languages (which only represent the defacto structure of a system), object-oriented CASE tools, and various ad-hoc techniques for representing and reasoning about system architecture.

Another alternative is the use of VHSIC Hardware Description Language (VHDL) tools. While VHDL is often thought of exclusively as a hardware description language, its modularization and communication protocol modeling capabilities are very similar to the ones under development for use in ADLs.

Complementary Technologies

Behavioral specification technologies and their associated theorem proving environments are used by several ADLs to provide capabilities to define component behavior. In addition, formal logics and techniques for representing

relationships between them are being used to define mappings between architectures within an ADL and to define mappings between ADLs.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Architecture Description Languages
Application category	Architectural Design (AP.1.3.1), Compiler (AP.1.4.2.3), Plan and Perform Integration (AP.1.4.4)
Quality measures category	Correctness (QM.1.3), Structuredness (QM.3.2.3), Reusability (QM.4.4)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2), Organization and Design (D.4.7), Performance (D.4.8), Systems Programs and Utilities (D.4.9)

References and Information Sources

[Garlan 93] Garlan, David & Shaw, Mary. "An Introduction to Software Architecture," 1-39. *Advances in Software Engineering and Knowledge Engineering* Volume 2. New York, NY: World Scientific Press, 1993.

[Garlan 94a] Garlan, D. & Allen, R. "Formalizing Architectural Connection," 71-80. *Proceedings of the 16th International Conference on Software Engineering*. Sorrento, Italy, May 16-21, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[Garlan 94b] Garlan, D.; Allen, R.; & Ockerbloom, J. "Exploiting Style in Architectural Design Environments." *SIGSOFT Software Engineering Notes 19*, 5 (December 1994): 175-188.

[Luckham 95] Luckham, David C., et al. "Specification and Analysis of System Architecture Using Rapide." *IEEE Transactions on Software Engineering 21*, 6 (April 1995): 336-355.

[Hoare 85] Hoare, C.A.R. *Communicating Sequential Processes*. Englewood Cliffs, NJ: Prentice Hall International, 1985.

[Paulisch 94] Paulisch, Frances. "Software Architecture and Reuse- An Inherent Conflict?" 214. *Proceedings of the 3rd International Conference on Software Reuse*. Rio de Janeiro, Brazil, November 1-4, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[Perry 92] Perry, D.E. & Wolf, A.L. "Foundations for the Study of Software Architectures."*SIGSOFT Software Engineering Notes 17*,4 (October 1992): 40-52.

[SATG 96] *Software Architecture Technology Guide* [online]. Available WWW <URL: http://www-ast.tds-gn.lmco.com/arch/guide.html> (1996).

[SEI 96] *Architectural Description Languages* [online]. Available WWW <URL: http://www.sei.cmu.edu/architecture/adl.html> (1996).

[Shaw 95] Shaw, Mary, et al. "Abstractions for Software Architecture and Tools to Support Them." *IEEE Transactions on Software Engineering 21*, 6 (April 1995): 314-335.

[Shaw 96] Shaw, M. & Garlan, D. *Perspective on an Emerging Discipline: Software Architecture*. Englewood Cliffs, NJ: Prentice Hall, 1996.

[STARS 96] *Scenarios for Analyzing Architecture Description Languages* Version 2.0 [online]. Originally available WWW <URL: http://www.asset.com/WSRD/abstracts/ABSTRACT_1183.html> (1996).

[Tracz 93] Tracz, W. "LILEANNA: a Parameterized Programming Language," 66-78. *Proceedings of the Second International Workshop on Software Reuse*. Lucca, Italy, March 24-26, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

[Vestal 93] Vestal, Steve. *A Cursory Overview and Comparison of Four Architecture Description Languages* [online]. Originally available FTP <URL: ftp://ftp.htc.honeywell.com/pub/dssa/papers/four_adl.ps> (1996).

[Vestal 96] Vestal, Steve. *Languages and Tools for Embedded Software Architectures* [online]. Available WWW <URL: http://www.htc.honeywell.com/projects/dssa/dssa_tools.html> (1996).

Current Author/Maintainer

Mark Gerken, Air Force Rome Laboratory

External Reviewers

Paul Clements, SEI
Paul Kogut, Lockheed Martin, Paoli, PA
Will Tracz, Lockheed Martin Federal Systems, Owego, NY

Modifications

10 Jan 97 (original)

Footnotes

[1] While definitions of *architecture*, *component*, and *connector* vary among researchers, this definition of architecture serves as a baseline for this technology description. A generally accepted definition describing the difference between a "design" and an "architecture" is that while a design explicitly addresses functional requirements, an architecture explicitly addresses functional and non-functional requirements such as reusability, maintainability, portability, interoperability, testability, efficiency, and fault-tolerance [Paulisch 94].

[2] Source: Garlan, David, et al. "ACME: An Architecture Interchange Language." Submitted for publication.

[3] However, one can easily imagine a case where an ADL is used to document the architecture, but then the project moves to the implementation phase and the ADL is forgotten. The code or low-level design migrates, but the architecture is lost. This is often referred to as architectural drift [Perry 92].

[4] For example, Rapide has been used to specify/ analyze the architecture model of the Sparc Version 9 64-bit instruction set, a standard published by Sparc International. Models of the extensions for the Ultra Sparc have also been done; they are used extensively in benchmarking Rapide simulation algorithms. Further information is available via the World Wide Web at http://anna.stanford.edu/rapide/rapide.html.

[5] Source: Garlan, David, et al. "ACME: An Architecture Interchange Language." Submitted for publication.

URL: http://www.sei.cmu.edu/str/descriptions/adl_body.html
Last Modified: 24 July 2008

Notes

[3] However, one can easily imagine a case where an ADL is used to document the architecture, but then the project moves to the implementation phase and the ADL is forgotten. The code or low-level design migrates, but the architecture is lost. This is often referred to as architectural drift [Perry 92].

References and Information Sources

[Garlan 94b] Garlan, D.; Allen, R.; & Ockerbloom, J. "Exploiting Style in Architectural Design Environments." *SIGSOFT Software Engineering Notes 19*, 5 (December 1994): 175-188.

References and Information Sources

[Vestal 96] Vestal, Steve. *Languages and Tools for Embedded Software Architectures* [online]. Available WWW <URL: http://www.htc.honeywell.com/projects/dssa/dssa_tools.html> (1996).

References and Information Sources

[Tracz 93] Tracz, W. "LILEANNA: a Parameterized Programming Language," 66-78. *Proceedings of the Second International Workshop on Software Reuse*. Lucca, Italy, March 24-26, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

References and Information Sources

[SATG 96] *Software Architecture Technology Guide* [online]. Available WWW <URL: http://www-ast.tds-gn.lmco.com/arch/guide.html> (1996).

References and Information Sources

[SEI 96] *Architectural Description Languages* [online]. Available WWW <URL: http://www.sei.cmu.edu/architecture/adl.html> (1996).

Notes

[4] For example, Rapide has been used to specify/ analyze the architecture model of the Sparc Version 9 64-bit instruction set, a standard published by Sparc International. Models of the extensions for the Ultra Sparc have also been done; they are used extensively in benchmarking Rapide simulation algorithms. Further information is available via the World Wide Web at http://anna.stanford.edu/rapide/rapide.html.

Notes

[5] Source: Garlan, David, et al. "ACME: An Architecture Interchange Language." Submitted for publication.

Related Topics

Architectural Design (Hardware-Software Co-Design) (AP.1.3.1)

- Architecture Description Languages
- Module Interconnection Languages

Related Topics

Plan and Perform Integration (AP.1.4.4)

- Architecture Description Languages
- Component-Based Software Development/ COTS Integration
- Module Interconnection Languages

Related Topics

Correctness (QM.1.3)

- Architecture Description Languages
- Cleanroom Software Engineering
- Module Interconnection Languages
- Software Inspections

Related Topics

Structuredness (QM.3.2.3)

- Architecture Description Languages
- Cyclomatic Complexity
- Module Interconnection Languages

References and Information Sources

[Paulisch 94] Paulisch, Frances. "Software Architecture and Reuse- An Inherent Conflict?" 214. *Proceedings of the 3rd International Conference on Software Reuse*. Rio de Janeiro, Brazil, November 1-4, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

References and Information Sources

[Perry 92] Perry, D.E. & Wolf, A.L. "Foundations for the Study of Software Architectures."*SIGSOFT Software Engineering Notes 17*,4 (October 1992): 40-52.

Glossary Term

Evolvability

the ease with which a system or component can be modified to take advantage of new software or hardware technologies.

References and Information Sources

[Shum 94] Shum, Buckingham Simon & Hammond, Nick. "Argumentation-Based Design Rationale: What Use at What Cost?" *International Journal of Human-Computer Studies 40*, 4 (April 1994): 603-52.

References and Information Sources

[Ramesh 92] Ramesh, Balasubramaniam & Dhar, Vasant. "Supporting Systems Development by Capturing Deliberations During Requirements Engineering." *IEEE Transactions on Software Engineering 18,* 6 (June 1992): 498-510.

Related Topics

Requirements Tracing (AP.1.2.3)

- Argument-Based Design Rationale Capture Methods for Requirements Tracing
- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Maintenance of Operational Systems an Overview
- Representation and Maintenance of Process Knowledge Method
- Requirements Tracing

Related Topics

Completeness/Incompleteness (QM.1.3.1)

- Argument-Based Design Rationale Capture Methods for Requirements Tracing
- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Requirements Tracing

Related Topics

Traceability (QM.1.3.3)

- Argument-Based Design Rationale Capture Methods for Requirements Tracing
- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Requirements Tracing

Related Topics

Effectiveness (QM.1.1)

- Argument-Based Design Rationale Capture Methods for Requirements Tracing
- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Requirements Tracing

Related Topics

Understandability (QM.3.2)

- Argument-Based Design Rationale Capture Methods for Requirements Tracing
- Cleanroom Software Engineering
- Domain Engineering and Domain Analysis
- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Feature-Oriented Domain Analysis
- Graphic Tools for Legacy Database Migration
- Halstead Complexity Measures
- Maintainability Index Technique for Measuring Program Maintainability
- Organization Domain Modeling
- Requirements Tracing

References and Information Sources

[IEEE 83] *IEEE Standard Glossary of Software Engineering Terminology*. New York, NY: Institute of Electrical and Electronic Engineers, 1983.

References and Information Sources

[Coleman 94] Coleman, Don, et al. "Using Metrics to Evaluate Software System Maintainability." *Computer 27*, 8 (August 1994): 44-49.

References and Information Sources

[Coleman 95] Coleman, Don; Lowther, Bruce; & Oman, Paul. "The Application of Software Maintainability Models in Industrial Software Systems." *Journal of Systems Software 29*, 1 (April 1995): 3-16.

Maintenance of Operational Systems--An Overview

Software Technology Roadmap

Status

Complete

Note

This description provides background information for technologies for optimizing maintenance environments. We recommend Cyclomatic Complexity; Halstead Complexity Measures; Maintainability Index Technique for Measuring Program Maintainability; and Function Point Analysis as concurrent reading, as they contain information about specific technologies.

Purpose and Origin

Technologies specific to the maintenance of software evolved (and are still evolving) out of development-oriented technologies. As large systems have proliferated and aged, the special needs of the operational environment have begun to emerge. Maintenance is defined here as the modification of a software product after delivery to correct faults, improve performance or other attributes, or to adapt the product to a changed environment [IEEE 83]. Historically, the software lifecycle has usually focused on development. However, so much of a system's cost is incurred during its operational lifetime that maintenance issues have become more important and, arguably, this should be reflected in development practices. Systems are required to last longer than originally planned; inevitably, the percentage of costs going to maintenance has been steadily climbing. Hewlett-Packard estimates that 60% to 80% of its R&D personnel are involved in maintaining existing software, and that 40% to 60% of production costs were directly related to maintenance [Coleman 94]. There was a rule of thumb that eighty percent of a Department of Defense (DoD) system's cost is in maintenance; older Cheyenne Mountain Complex systems may have surpassed ninety percent. Yet software development practices still do not put much emphasis on making the product highly maintainable.

Cost and risk of maintenance of older systems are further exacerbated by a shortage of suitable maintenance skills; analysts and programmers are not trained to deal with these systems. Industry wide, it is claimed that 75%-80% of all operational software was written without the discipline of structured programming [Coleman 95]. Only a minuscule fraction of current operational systems were built using the object-oriented techniques taught today.

The purpose of this description is to provide a framework or a contextual reference for some of the maintenance and reengineering technologies described in this document.

Technical Detail

The operational system lifecycle. The operational environment has its own lifecycle that, while connected to the development lifecycle, has specific and unique characteristics and needs. As shown in Figure 15, a system's total lifecycle is defined as having four major phases:

- the development or pre-delivery phase
- the early operational phase
- the mature operational phase
- the evolution/replacement phase

Each of the phases has typical characteristics and problems. The *operational* phases are most of the lifecycle and cost. The narrative following describes each phase, and identifies specific technologies in (or planned for) this document that can be applied to correct or improve the situation. In almost every case, taking the proper action in a given phase can eliminate, or greatly reduce, problems in a later phase- at much less cost.

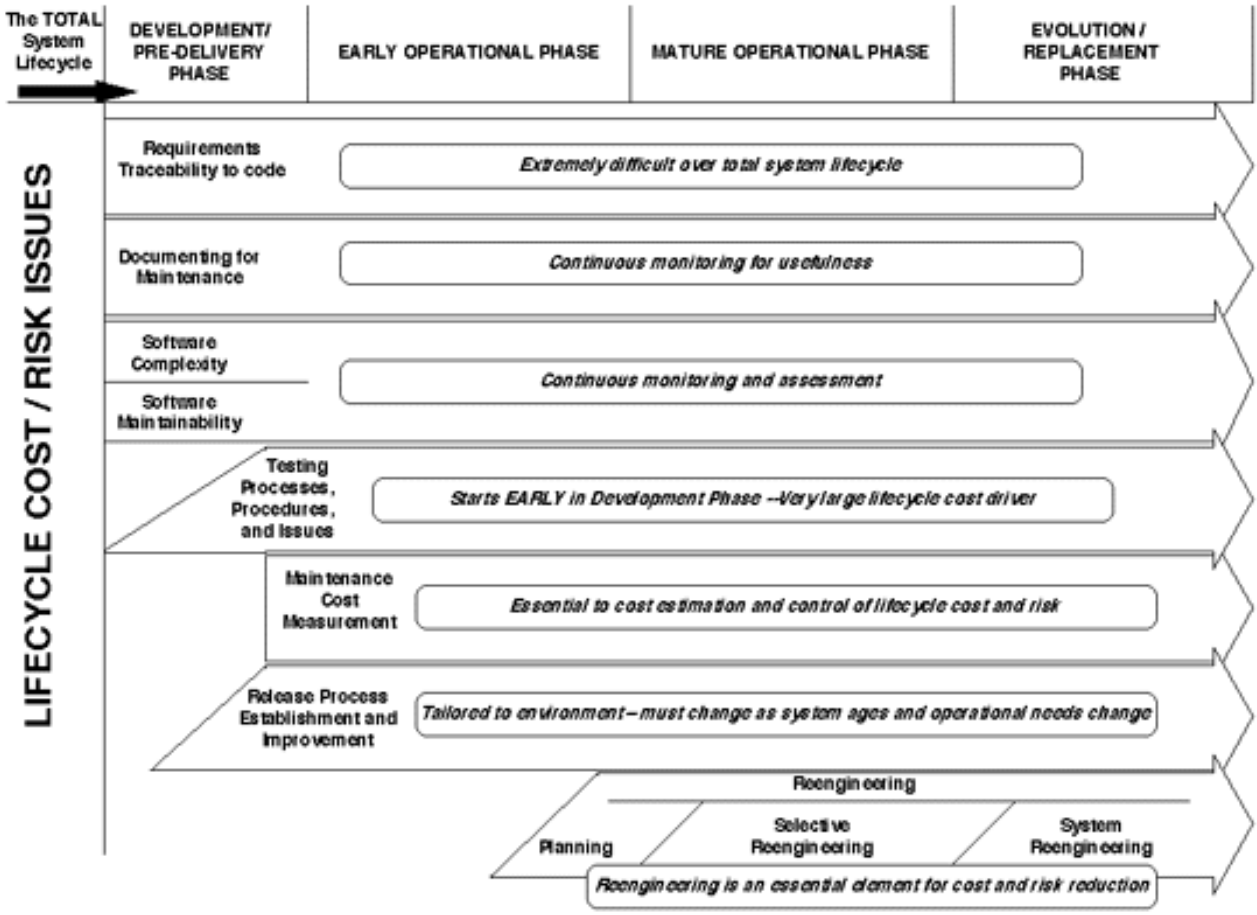


Figure 15: Total System Life Cycle

Terminology. To set a baseline for the descriptions of these phases, the following definitions are used:

Reengineering: rebuilding a piece of software to suit some new purpose (to work on another platform, to switch to another language, to make it more maintainable, etc.); often preceded by reverse engineering. Examination and alteration of a subject system to reconstitute it in a new

form. Any activity that improves one's understanding of software, or prepares or improves the software itself for increased maintainability, reusability, or evolvability.

Restructuring: transformation of a program from one representation to another at the same relative abstraction level, usually to simplify or clarify it in some way (e.g., remove GOTOs, increase modularity), while preserving external behavior.

Reverse engineering: the process of analyzing a system's code, documentation, and behavior to identify its current components and their dependencies to extract and create system abstractions and design information. The subject system is not altered; however, additional knowledge about the system is produced. Redocumenting and design recovery are techniques associated with reverse engineering.

Software complexity: some measure of the mental effort required to understand a piece of software.

Software maintainability: some measure of the ease and/or risk of making a change to a piece of software. The measured complexity of the software is often used in quantifying maintainability.

Translation: conversion of a program from one language to another, often as a companion action to restructuring the program.

Phase 1: The development or pre-delivery phase, when the system is not yet operational. Most of the effort in this phase goes into making Version One of the system function. But if total lifecycle costs are to be minimized, planning and preparation for maintenance during the development phase are essential. Most currently operational systems did not receive this attention during development. Several areas should be addressed:

- *Requirements traceability to code*. Requirements are the foundation of a system, and one of the most common faults of an operational system is that the relationship between its requirements and its code cannot be determined. Recovering this information for a system after it goes operational is a costly and time-consuming task. See Requirements Tracing, Feature-Based Design Rationale Capture Method for Requirements Tracing, and Argument-Based Design Rationale Capture Methods for Requirements Tracing for assistance in creating initial mappings from requirements to code.
- *Documentation and its usefulness in maintenance*. The ostensible purpose of documentation is to aid in understanding what the system does, and (for the maintenance programmer) how the system does it. There is at least anecdotal evidence that
    - Classical specification-type documentation is not a good primary source of information for the maintenance programmer looking for a problem's origin, especially since the documentation is frequently inconsistent with the code.
    - The most useful maintenance information is derived directly and automatically from the code; examples include structure charts, program flow diagrams, and cross-reference lists. This suggests that tools that create and maintain these documentation forms should be used during development of the code, and delivered with it.
- *The complexity of the software*. If the software is too complex to understand when it is first developed, it will only become more complex and brittle as it is changed. Measuring complexity during code development is useful for checking code condition, helps in quantifying testing costs, and aids in forecasting future maintenance costs (see

Cyclomatic Complexity, Halstead Complexity Measures, and Maintainability Index Technique for Measuring Program Maintainability).

- *The maintainability of the software*. This is perhaps the key issue for the maintainer. The ability to measure a system's maintainability directly affects the ability to predict future costs and risks. Maintainability Index Technique for Measuring Program Maintainability describes a practical approach to such a measurement, applicable throughout the lifecycle.

Phase 2: The early operational phase, when the delivered system is being maintained and changed to meet new needs and fix problems. Typically the tools and techniques used for maintenance are those that were used to develop the system. In this phase, the following issues are critical:

- *Complexity and maintainability* must be measured and controlled in this phase if the major problems of Phase 3 are to be avoided. Ideally, this a continuation of the same effort that began in Phase 1, and it depends on the same tools and techniques (see Cyclomatic Complexity, Halstead Complexity Measures, and Maintainability Index Technique for Measuring Program Maintainability). In a preventative maintenance regime, use of these types of measures will help establish guidelines about how much complexity and/or deterioration of maintainability is tolerable. If a critical module becomes too complex under the guidelines, it should be considered for rework before it becomes a problem. Early detection of problems, such as risk due to increasing complexity of a module, is far cheaper than waiting until a serious problem arises.
- *A formal release-based maintenance process* that suits the environment must be established. This process should always be subject to inspection, and should be revised when it does not meet the need.
- *The gathering of cost data* must be part of the maintenance process if lifecycle costs are to be understood and controlled. The cost of each change (e.g., person-hours, computer-hours) should be known down to a suitable granularity such as phase within the release (e.g., design, code and unit test, integration testing). Without this detailed cost information, it is very hard to estimate future workload or the cost of a proposed change.

Phase 3: Mature operational phase, in which the system still meets the users' primary needs but is showing signs of age. For example

- The incidence of bugs caused by changes or "day-one errors" (problems that existed at initial code delivery) is rising, and the documentation, especially higher-level specification material, is not trustworthy. Most analyses of changes to the software must be done by investigating the code itself.
- Code "entropy" and complexity are increasing and, even by subjective measures, its maintainability is decreasing.
- New requirements increasingly uncover limitations that were designed into the system.
- Because of employee turnover, the programming staff may no longer be intimately familiar with the code, which increases both the cost of a change and the code's entropy.
- A change may have a ripple effect: Because the true nature of the code is not well known, coupling across modules has increased and made it more likely that a change in one area will affect another area. It may be appropriate to restructure or reengineer selected parts of the system to lessen this problem.
- Testing has become more time-consuming and/or risky because as code complexity increases, test path coverage also increases. It may be appropriate to consider more sophisticated test approaches (see Preventive Maintenance).

- The platform is obsolete: The hardware is not supported by the manufacturer and parts are not readily available; the COTS software is not supported through new releases (or the new releases will not work with the application, and it is too risky to make the application changes needed to align with the COTS software).

At this point, the code has not been rewritten en masse or reverse engineered to recover design, but the risk and cost of evolution by modification of the system have increased significantly. The system has become brittle with age. It may be appropriate to assess the system's condition. Sittenauer describes a quick methodology for gauging the need for reengineering, and the entire approach for measuring maintainability (see Maintainability Index Technique for Measuring Program Maintainability) allows continuous or spot assessment of the system's maintainability [Sittenauer 92].

Phase 4: Evolution/Replacement Phase, in which the system is approaching or has reached insupportability. The software is no longer maintainable. It has become so "entropic" or brittle that the cost and/or risk of significant change is too high, and/or the host hardware/software environment is obsolete. Even if none of these is true, the cost of implementing a new requirement is not tolerable because it takes too long under the maintenance environment. It is time to consider reengineering (see Cleanroom Software Engineering and Graphical User Interface Builders).

Usage Considerations

Software maintainability factors. The characteristics influencing or determining a system's maintainability have been extensively studied, enumerated, and organized. One thorough study is described in Oman; such characteristics were analyzed and a simplified maintainability taxonomy was constructed [Oman 91]. Maintainability Index Technique for Measuring Program Maintainability describes an approach to measuring and controlling code maintainability that was founded on several years of work and analysis and includes analysis of commercial software maintenance. References to other maintainability research results also appear in that technology description.

Preventive maintenance approaches. The approaches listed below are a few of the ways current technology can help to enhance system maintainability.

- *Complexity analysis*. Before attempting to reach a destination, it is essential to know where you are. For a software system, a good first step is measuring the complexity of the component modules (see Cyclomatic Complexity and Halstead Complexity Measures). Maintainability Index Technique for Measuring Program Maintainability describes a method of assessing maintainability of code using those complexity measures. Test path coverage can also be determined from complexity measures, which can help in optimizing system testing (see Test generation and optimization).
- *Functionality analysis*. Function Point Analysis describes the uses and limitations of function point analysis (also known as functional size measurement) in measuring software. By measuring a program's functionality, one can arrive at some estimate of its value in a system, which is of use when making decisions about rewriting the program or reengineering the system. Measures of functionality can also guide decisions about where to put testing effort (see Test generation and optimization).
- *Reverse engineering / design recovery*. Over time, a system's code diverges from the documentation; this is a well-known tendency of operational systems. Another

phenomenon that is frequently underestimated or ignored is that (regardless of the divergence effect) the information required to make a given change is often found only in the code. Several approaches are possible here. Various tools offer the ability to construct program flow diagrams (PFDs) from code. More sophisticated techniques, often classified as program understanding, are emerging. These technologies are implemented as tools that act as agents for the human analyst to assist in gathering information about a program's function at higher levels of abstraction than a program flow diagram (e.g., retask a satellite).

- *Piecewise reengineering*. If the system's known lifetime is sufficiently short, and if the evolutionary changes needed are sufficiently bounded, the system may benefit from a piecewise reengineering approach:
    - Brittle, high-risk modules that are likely to need changes are identified and reengineered to make them more maintainable. Techniques such as wrappers, an emerging technology, are expected to aid here.
    - For the sake of prudence, other risky modules are "locked," so that a prospective change to them can be made only after thoroughly assessing the risks involved.
    - For database systems, it may be possible to retrofit a modern relational or object-oriented database to the system; Common Object Request Broker Architecture and Graphic Tools for Legacy Database Migration describe technologies of possible use here. Piecewise reengineering can generally be done at a lower cost than complete reengineering of the system. If it is the right choice, it delays the inevitable obsolescence. The downsides of piecewise reengineering include the following:
    - Platform obsolescence is not reversed. Risks arising from the platform's software are unchanged; if the original database or operating system has risks, the application using them will also.
    - Unforeseen requirements changes still carry high risk if they affect the old parts of the system.
    - Performance may suffer because of the interface structures added to splice reengineered functions to old ones.
- *Translation/restructuring/modularizing*. Translation and/or restructuring of code are often of interest when migrating software to a new platform. Frequently the new environment will not support the old language or dialect. Restructuring/modularizing, or rebuilding the code to reduce complexity, can be done simply to improve the code's maintainability, but code to be translated is often restructured first so that the result will be less complex and more easily understood. There are several commercial tools that do one or more of these operations, and energetic research to achieve more automated approaches is being done. Welker cites evidence that translation does little or nothing to enhance maintainability [Welker 95]. Most often, it simply continues the existing problem in a different syntactical form; the mechanical forms output by translators decrease understandability, which is a key component of maintainability. None of these technologies is a cure-all, and none of them should be applied without first assessing the quality of the output and the amount of programmer resources required.

Test generation and optimization. Mission criticality of many DoD systems drives the maintenance activity to test very thoroughly. Boehm reported integration testing activities consuming only 16-34% of project totals [Boehm 81], but other evidence is available to show that commercial systems testing activity can take half of a development effort's resources [Alberts 76, DeMillo 87, Myers 79]. Recent composite post-release reviews of operational Cheyenne Mountain Complex system releases show that testing consumed 60-70% of the total release effort.[1] Any technology that can improve testing efficiency will have high leverage on the system's life-cycle costs. Technologies that can possibly help include: automatic test case generation; generation of test and analysis tools; redundant test case elimination; test data

generation by chaining; techniques for software regression testing; and techniques for statistical test plan generation and coverage analysis.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Maintenance of Operational Systems--An Overview
Application category	Requirements Tracing (AP.1.2.3) Cost Estimation (AP.1.3.7) Test (AP.1.4.3) System Testing (AP.1.5.3.1) Regression Testing (AP.1.5.3.4) Reapply Software Lifecycle (AP.1.9.3) Reverse Engineering (AP.1.9.4) Reengineering (AP.1.9.5)
Quality measures category	Maintainability (QM.3.1)
Computing reviews category	Software Engineering Distribution and Maintenance (D.2.7) Software Engineering Metrics (D.2.8) Software Engineering Management (D.2.9)

References and Information Sources

[Alberts 76] Alberts, D. "The Economics of Software Quality Assurance." *National Computer Conference*. New York, NY, June 7-10, 1976. Montvale, NJ: American Federation of Information Processing Societies Press, 1976.

[Boehm 81] Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[Coleman 94] Coleman, Don, et al. "Using Metrics to Evaluate Software System Maintainability." *Computer 27,* 8 (August 1994): 44-49.

[Coleman 95] Coleman, Don; Lowther, Bruce; & Oman, Paul. "The Application of Software Maintainability Models in Industrial Software Systems." *Journal of Systems Software 29,* 1 (April 1995): 3-16.

[DeMillo 87] DeMillo, R., et al. *Software Testing and Evaluation*. Menlo Park, CA: Benjamin/ Cummings, 1987.

[IEEE 83] *IEEE Standard Glossary of Software Engineering Terminology*. New York, NY: Institute of Electrical and Electronic Engineers, 1983.

[Myers 79]	Myers, G. *The Art of Software Testing.* New York, NY: John Wiley and Sons, 1979.
[Oman 91]	Oman, P.; Hagermeister, J.; & Ash, D. *A Definition and Taxonomy for Software Maintainability* (91-08-TR). Moscow, ID: Software Engineering Test Laboratory, University of Idaho, 1991.
[Sittenauer 92]	Sittenauer, Chris & Olsem, Mike. "Time to Reengineer?" *Crosstalk, Journal of Defense Software Engineering* 32 (March 1992): 7-10.
[Welker 95]	Welker, Kurt D. & Oman, Paul W. "Software Maintainability Metrics Models in Practice." *Crosstalk, Journal of Defense Software Engineering 8,* 11 (November/ December 1995): 19-23.

Current Author/Maintainer

Edmond VanDoren, Kaman Sciences, Colorado Springs

External Reviewers

Brian Gallagher, SEI
Ed Morris, SEI
Dennis Smith, SEI

Modifications

10 Jan 97 (original)

Footnotes

[1] Source: Kaman Sciences Corp. *Minutes of the 96-1 Composite Post-Release Review (CPRR), Combined CSS/CSSR and ATAMS Post-Release Review and Software Engineering Post-Release Review* KSWENG Memo # 96-03, 26 July, 1996.

URL: http://www.sei.cmu.edu/str/descriptions/mos_body.html
Last Modified: 24 July 2008

References and Information Sources

[Sittenauer 92] Sittenauer, Chris & Olsem, Mike. "Time to Reengineer?" *Crosstalk, Journal of Defense Software Engineering* 32 (March 1992): 7-10.

References and Information Sources

[Oman 91] Oman, P.; Hagermeister, J.; & Ash, D. *A Definition and Taxonomy for Software Maintainability* (91-08-TR). Moscow, ID: Software Engineering Test Laboratory, University of Idaho, 1991.

References and Information Sources

[Welker 95] Welker, Kurt D. & Oman, Paul W. "Software Maintainability Metrics Models in Practice." *Crosstalk, Journal of Defense Software Engineering 8*, 11 (November/ December 1995): 19-23.

References and Information Sources

[Boehm 81] Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

References and Information Sources

[Alberts 76] Alberts, D. "The Economics of Software Quality Assurance." *National Computer Conference*. New York, NY, June 7-10, 1976. Montvale, NJ: American Federation of Information Processing Societies Press, 1976.

References and Information Sources

[DeMillo 87] DeMillo, R., et al. *Software Testing and Evaluation*. Menlo Park, CA: Benjamin/ Cummings, 1987.

References and Information Sources

[Myers 79] Myers, G. *The Art of Software Testing.* New York, NY: John Wiley and Sons, 1979.

Notes

[1] Source: Kaman Sciences Corp. *Minutes of the 96-1 Composite Post-Release Review (CPRR), Combined CSS/CSSR and ATAMS Post-Release Review and Software Engineering Post-Release Review* KSWENG Memo # 96-03, 26 July, 1996.

Related Topics

Cost Estimation (AP.1.3.7)

- COCOMO Method
- Function Point Analysis
- Maintenance of Operational Systems -- an Overview

Related Topics

Test (AP.1.4.3)

- Bowles Metrics
- Cyclomatic Complexity
- Halstead Complexity Measures
- Henry and Kafura Metrics
- Ligier Metrics
- Maintainability Index Technique for Measuring Program Maintainability
- Maintenance of Operational Systems -- an Overview
- Troy and Zweben Metric

Related Topics

System Testing (AP.1.5.3.1)

- Cleanroom Software Engineering
- Maintenance of Operational Systems -- an Overview

Related Topics

Regression Testing (AP.1.5.3.4)

- Maintenance of Operational Systems -- an Overview
- Regression Testing Techniques

Related Topics

Reapply Software Life Cycle (AP.1.9.3)

- Bowles Metrics
- Cyclomatic Complexity
- Data Complexity
- Design Complexity
- Essential Complexity
- Graphical User Interface Builders
- Halstead Complexity Measures
- Henry and Kafura Metrics
- Ligier Metrics
- Maintainability Index Technique for Measuring Program Maintainability
- Maintenance of Operational Systems -- an Overview
- Personal Software Process for Module-Level Development
- Rate Monotonic Analysis
- Simplex Architecture
- Troy and Zweben Metrics

Related Topics

Reverse Engineering (AP.1.9.4)

- Cyclomatic Complexity
- Data Mining
- Data Warehousing
- Maintenance of Operational Systems -- an Overview

Related Topics

Reengineering (AP.1.9.5)

- Bowles Metrics
- Cleanroom Software Engineering
- Component-Based Software Development/ COTS Integration
- Cyclomatic Complexity
- Data Complexity
- Design Complexity
- Essential Complexity
- Graphic Tools for Legacy Database Migration
- Graphical User Interface Builders
- Halstead Complexity Measures
- Henry and Kafura Metrics
- Ligier Metrics
- Maintainability Index Technique for Measuring Program Maintainability
- Maintenance of Operational Systems -- an Overview
- Object-Oriented Analysis
- Object-Oriented Design
- Personal Software Process for Module-Level Development
- Rate Monotonic Analysis
- Simplex Architecture
- Troy and Zweben Metrics

Glossary Term

Complexity

1. (Apparent) the degree to which a system or component has a design or implementation that is difficult to understand and verify [IEEE 90].
2. (Inherent) the degree of complication of a system or system component, determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, and the types of data structures [Evans 87].

References and Information Sources

[Rao 95] Rao, B.R. "Making the Most of Middleware." *Data Communications International 24*, 12 (September 1995): 89-96.

Message-Oriented Middleware

Software Technology Roadmap

Status

Advanced

Note

We recommend Middleware *as prerequisite reading for this technology description.*

Purpose and Origin

Message-oriented middleware (MOM) is a client/server infrastructure that increases the *interoperability*, *portability*, and *flexibility* of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the *complexity* of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces- Application Programming Interfaces (APIs) that extend across diverse platforms and networks are typically provided by the MOM [Rao 95].

Technical Detail

Message-oriented middleware, as shown in Figure 22 [Steinke 95], is software that resides in both portions of a client/server architecture and typically supports asynchronous calls between the client and server applications. Message queues provide temporary storage when the destination program is busy or not connected. MOM reduces the involvement of application developers with the *complexity* of the master-slave nature of the client/server mechanism.

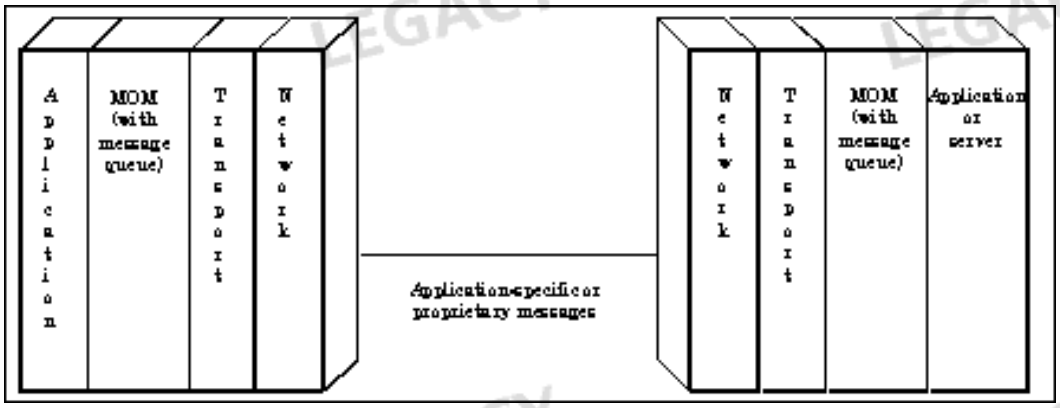


Figure 22: Message-Oriented Middleware

MOM increases the flexibility of an architecture by enabling applications to exchange messages with other programs without having to know what platform or processor the other application resides on within the network. The aforementioned messages can contain formatted data, requests for action, or both. Nominally, MOM systems provide a message queue between interoperating processes, so if the destination process is busy, the message is held in a temporary storage location until it can be processed. MOM is typically asynchronous and peer-to-peer, but most implementations support synchronous message passing as well.

Usage Considerations

MOM is most appropriate for event-driven applications. When an event occurs, the client application hands off to the messaging middleware application the responsibility of notifying a server that some action needs to be taken. MOM is also well-suited for object-oriented systems because it furnishes a conceptual mechanism for peer-to-peer communications between objects. MOM insulates developers from connectivity concerns- the application developers write to APIs that handle the complexity of the specific interfaces.

Asynchronous and synchronous mechanisms each have strengths and weaknesses that should be considered when designing any specific application. The asynchronous mechanism of MOM, unlike Remote Procedure Call (RPC) , which uses a a synchronous, blocking mechanism, does not guard against overloading a network. As such, a negative aspect of MOM is that a client process can continue to transfer data to a server that is not keeping pace. Message-oriented middleware's use of message queues, however, tends to be more flexible than RPC-based systems, because most implementations of MOM can default to synchronous and fall back to asynchronous communication if a server becomes unavailable [Steinke 95].

Maturity

Implementations of MOM first became available in the mid-to-late 1980s. Many MOM implementations currently exist that support a variety of protocols and operating systems. Many implementations support multiple protocols and operating systems simultaneously.

Some vendors provide tool sets to help extend existing interprocess communication across a heterogeneous network.

Costs and Limitations

MOM is typically implemented as a proprietary product, which means MOM implementations are nominally incompatible with other MOM implementations. Using a single implementation of a MOM in a system will most likely result in a dependence on the MOM vendor for maintenance support and future enhancements. This could have a highly negative impact on a system's flexibility, maintainability, portability, and interoperability.

The message-oriented middleware software (kernel) must run on every platform of a network. The impact of this varies and depends on the characteristics of the system in which the MOM will be used:

- Not all MOM implementations support all operating systems and protocols. The flexibility to choose a MOM implementation may be dependent on the chosen application platform or network protocols supported, or vice versa.
- Local resources and CPU cycles must be used to support the MOM kernels on each platform. The performance impact of the middleware implementation must be considered; this could possibly require the user to acquire greater local resources and processing power.
- The administrative and maintenance burden would increase significantly for a network manager with a large distributed system, especially in a mostly heterogeneous system.
- A MOM implementation may cost more if multiple kernels are required for a heterogeneous system, especially when a system is maintaining kernels for old platforms and new platforms simultaneously.

Alternatives

Other infrastructure technologies that allow the distribution of processing across multiple processors and platforms are

- Object Request Broker (ORB)
- Distributed Computing Environment (DCE)
- Remote Procedure Call (RPC)
- Transaction Processing Monitor Technology
- Three Tier Software Architectures

Complementary Technologies

MOM can be effectively combined with remote procedure call (RPC) technology-RPC can be used for synchronous support by a MOM.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Message-Oriented Middleware
Application category	Client/Server (AP.2.1.2.1) Client/Server Communication (AP.2.2.1)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1) Portability (QM.4.2)
Computing reviews category	Distributed Systems (C.2.4) Network Architecture and Design (C.2.1)

References and Information Sources

[Rao 95]	Rao, B.R. "Making the Most of Middleware." *Data Communications International 24*, 12 (September 1995): 89-96.
[Steinke 95]	Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

Current Author/Maintainer

Cory Vondrak, TRW, Redondo Beach, CA

External Reviewers

Ed Morris, SEI

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/momt_body.html
Last Modified: 24 July 2008

References and Information Sources

[Steinke 95] Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

Related Topics

Client/Server (AP.2.1.2.1)

- Common Object Request Broker Architecture
- Component Object Model (COM), DCOM, and Related Capabilities
- Database Two Phase Commit
- Distributed/Collaborative Enterprise Architectures
- Mainframe Server Software Architectures
- Message-Oriented Middleware
- Middleware
- Object Request Broker
- Remote Data Access (RDA)
- Remote Procedure Call
- Session-Based Technology
- Three Tier Software Architectures
- Transaction Processing Monitor Technology
- Two Tier Software Architectures

Related Topics

Client/Server Communication (AP.2.2.1)

- Common Object Request Broker Architecture
- Component Object Model (COM), DCOM, and Related Capabilities
- Message-Oriented Middleware
- Middleware
- Object Request Broker
- Remote Data Access
- Remote Procedure Call
- Session-Based Technology
- Transaction Processing Monitor Technology

References and Information Sources

[Eckerson 95] Eckerson, Wayne. "Searching for the Middle Ground." *Business Communications Review 25,* 9 (September 1995): 46-50.

Middleware

Software Technology Roadmap

Status

Advanced

Purpose and Origin

Middleware is connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. Middleware is essential to migrating mainframe applications to client/server applications and to providing for communication across heterogeneous platforms. This technology has evolved during the 1990s to provide for *interoperability* in support of the move to client/server architectures (see Client/Server Software Architectures). The most widely-publicized middleware initiatives are the Open Software Foundation's Distributed Computing Environment (DCE) , Object Management Group's Common Object Request Broker Architecture (CORBA), and Microsoft's COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities) [Eckerson 95].

Technical Detail

As outlined in Figure 17, middleware services are sets of distributed software that exist between the application and the operating system and network services on a system node in the network.

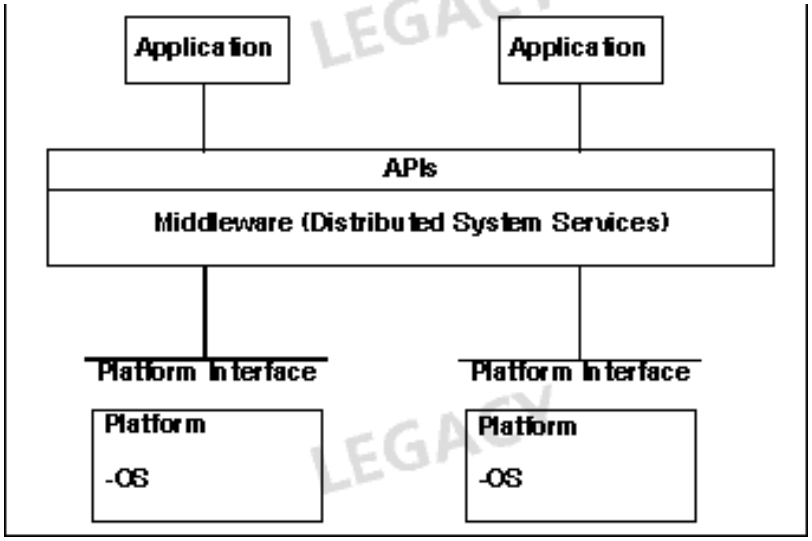


Figure 17: Use of Middleware [Bernstein 96]

Middleware services provide a more functional set of Application Programming Interfaces (API) than the operating system and network services to allow an application to

- locate transparently across the network, providing interaction with another application or service
- be independent from network services
- be reliable and available
- scale up in capacity without losing function [Schreiber 95]

Middleware can take on the following different forms:

- Transaction processing (TP) monitors (see Transaction Processing Monitor Technology), which provide tools and an environment for developing and deploying distributed applications.
- Remote Procedure Call (RPCs), which enable the logic of an application to be distributed across the network. Program logic on remote systems can be executed as simply as calling a local routine.
- Message-Oriented Middleware (MOM), which provides program-to-program data exchange, enabling the creation of distributed applications. MOM is analogous to email in the sense it is asynchronous and requires the recipients of messages to interpret their meaning and to take appropriate action.
- Object Request Brokers (ORBs) , which enable the objects that comprise an application to be distributed and shared across heterogeneous networks.

Usage Considerations

The main purpose of middleware services is to help solve many application connectivity and interoperability problems. However, middleware services are not a panacea:

- There is a gap between principles and practice. Many popular middleware services use proprietary implementations (making applications dependent on a single vendor's product).
- The sheer number of middleware services is a barrier to using them. To keep their computing environment manageably simple, developers have to select a small number of services that meet their needs for functionality and platform coverage.
- While middleware services raise the level of abstraction of programming distributed applications, they still leave the application developer with hard design choices. For example, the developer must still decide what functionality to put on the client and server sides of a distributed application [Bernstein 96].

The key to overcoming these three problems is to fully understand both the application problem and the value of middleware services that can enable the distributed application. To determine the types of middleware services required, the developer must identify the functions required, which fall into one of three classes:

1. Distributed system services, which include critical communications, program-to-program, and data management services. This type of service includes RPCs, MOMs and ORBs.
2. Application enabling services, which give applications access to distributed services and the underlying network. This type of services includes transaction monitors (see Transaction Processing Monitor Technology) and database services such as Structured Query Language (SQL).
3. Middleware management services, which enable applications and system functions to be continuously monitored to ensure optimum performance of the distributed environment [Schreiber 95].

Maturity

A significant number of middleware services and vendors exist. Middleware applications will continue to grow with the installation of more heterogeneous networks. An example of middleware in use is the Delta Airlines Cargo Handling System, which uses middleware technology to link over 40,000 terminals in 32 countries with UNIX services and IBM mainframes. By 1999, middleware sales are expected to exceed $6 billion [Client 95].

Costs and Limitations

The costs of using middleware technology (i.e., license fees) in system development are entirely dependent on the required operating systems and the types of platforms. Middleware product implementations are unique to the vendor. This results in a dependence on the vendor for maintenance support and future enhancements. This reliance could have a negative effect on a system's flexibility and maintainability. However, when evaluated against the cost of developing a unique middleware solution, the system developer and maintainer may view the potential negative effect as acceptable.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Middleware
Application category	Client/Server (AP.2.1.2.1) Client/Server Communication (AP.2.2.1)
Quality measures category	Interoperability (QM.4.1)
Computing reviews category	Distributed Systems (C.2.4) Network Architecture and Design (C.2.1) Database Management Languages (D.3.2)

References and Information Sources

[Bernstein 96]	Bernstein, Philip A. "Middleware: A Model for Distributed Services." *Communications of the ACM 39*, 2 (February 1996): 86-97.
[Client 95]	"Middleware Can Mask the Complexity of your Distributed Environment." *Client/Server Economics Letter 2*, 6 (June 1995): 1-5.
[Eckerson 95]	Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." *Open Information Systems 10*, 1 (January 1995): 3(20).
[Schreiber 95]	Schreiber, Richard. "Middleware Demystified." *Datamation 41*, 6 (April 1, 1995): 41-45.

Current Author/Maintainer

Mike Bray, Lockheed-Martin Ground Systems

Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/middleware_body.html
Last Modified: 24 July 2008

References and Information Sources

[Bernstein 96] Bernstein, Philip A. "Middleware: A Model for Distributed Services." *Communications of the ACM 39*, 2 (February 1996): 86-97.

References and Information Sources

[Schreiber 95] Schreiber, Richard. "Middleware Demystified." *Datamation 41*, 6 (April 1, 1995): 41-45.

References and Information Sources

[Client 95] "Middleware Can Mask the Complexity of your Distributed Environment." *Client/Server Economics Letter 2*, 6 (June 1995): 1-5.

References and Information Sources

[Gluch 98] Gluch, D. & Weinstock, C. *Model-Based Verification: A Technology for Dependable System Upgrade* (CMU/SEI-98-TR-009, ADA 354756). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1998. Available WWW: <http://www.sei.cmu.edu/publications/documents/98.reports/98tr009/98tr009abstract.html>

References and Information Sources

[Clark 95] Clarke, Edmund M., et al. "Verification of the Futurebus+ Cache Coherence Protocol." *Formal Methods in System Design 6,* 2 (March 1995): 217-232.

References and Information Sources

[Fujita 96] Fujita, M. "Debugging a Communications Chip." *IEEE Spectrum 33*, 6 (June 1996): 64.

References and Information Sources

[Raimi 97] Raimi, R. & Lear, J. "Analyzing a PowerPCTM 620 Microprocessor Silicon Failure Using Model Checking," 964-973. *Proceedings of the International Test Conference 1997*, Washington, D.C., November 1-6, 1997.

References and Information Sources

[Gluch 99] Gluch, D. & Brockway, J. *An Introduction to Software Engineering Practices Using Model-Based Verification* (CMU/SEI-99-TR-005, ESC-TR-99-005). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1999. Available WWW: <http://www.sei.cmu.edu/publications/documents/99.reports/99tr005/99tr005abstract.html>

Related Topics

Models (AP.2.1.1)

- Client/Server Software Architectures
- Component Object Model (COM), DCOM, and Related Capabilities
- Defense Information Infrastructure Common Operating Environment
- European Computing Manufacturer's Association Reference Model (ECMA)
- Joint Technical Architecture
- Project Support Environment Reference Model (PSERM)
- Reference Models, Architectures, Implementations -- An Overview
- TAFIM Reference Model

Related Topics

Real-time Responsiveness/Latency (QM.2.2.2)

- Rate Monotonic Analysis
- Simplex Architecture

References and Information Sources

[DeRemer 76] DeRemer, F. & Kron, H. "Programming-in-the-Large Versus Programming-in-the-Small." *IEEE Transactions on Software Engineering SE-2*, 2 (June 1976): 321-327.

References and Information Sources

[Prieto-Diaz 86] Prieto-Diaz, Ruben & Neighbors, James. "Module Interconnection Languages." *Journal of Systems and Software 6*, 4 (1986): 307-334.

References and Information Sources

[Tichy 79] Tichy, W. F. "Software Development Control Based on Module Interconnection," 29-41. *Proceedings of the 4th International Conference on Software Engineering*. Munich, Germany, September 17-19, 1979. New York, NY: IEEE Computer Society Press, 1979.

References and Information Sources

[Cooprider 79] Cooprider, Lee W. *The Representation of Families of Software Systems* (CMU-CS-79-116). Pittsburgh, PA: Computer Science Department, Carnegie Mellon University, 1979.

Module Interconnection Languages

Software Technology Roadmap

Status

Complete

Purpose and Origin

As software system size and complexity increase, the task of integrating independently-developed subsystems becomes increasingly difficult. In the 1970s, manual integration was augmented with various levels of automated support, including support from module interconnection languages (MILs). The first MIL, MIL75, was described by DeRemer and Kron [DeRemer 76], who argued with integrators and developers about the differences between programming in the small, for which typical languages are suitable, and programming in the large, for which a MIL is required for knitting modules together [Prieto-Diaz 86]. MILs provide formal grammar constructs for identifying software system modules and for defining the interconnection specifications required to assemble a complete program [Prieto-Diaz 86]. MILs increase the *understandability* of large systems in that they formally describe the structure of a software system; they consolidate design and module assembly in a single language. MILs can also improve the *maintainability* of a large system in that they can be used to prohibit maintainers from accidentally changing the architectural design of a system, and they can be integrated into a larger development environment in which changes in the MIL specification of a system are automatically reflected at the code level and vice versa.

Technical Detail

A MIL identifies the system modules and states how they fit together to implement the system's function; MILs are *not* concerned with what the system does, how the major parts of the system are embedded in the organization, or how the individual modules implement their functions [Prieto-Diaz 86]. A MIL specification of a system constitutes a written description of the system design. A MIL specification can be used to

- Enforce system integrity and inter-modular compatibility.
- Support incremental modification. Modules can be independently compiled and linked; full recompilation of a modified system is not needed.
- Enforce version control. Different versions (implementations) of a module

can be identified and used in the construction of a software system. This idea has been generalized to allow different versions of subsystems to be defined in terms of different versions of modules. Thus MILs can be used to describe families of modules and systems [Tichy 79, Cooprider 79].

For example, consider the simplified MIL specification shown in Figure 24 and its associated graphical representation shown in Figure 25. The hypothetical MIL used in Figure 24 contains structures for identifying the modules of interest (in this case the modules are ABC, Z, and YBC); structures for identifying required and provided data; provided functions; and structures for identifying module and function versions. The module ABC defined in the figure consists of two parts, a function XA and a module YBC; the structure of each of these entities is also defined. Note that function XA has three versions, a Pascal, an Ada, and a FORTRAN version. These three versions would be written and compiled using their respective language development environments. A compilation system for this hypothetical MIL would process the specification given in Figure 24 to check that all required resources (such as x and z) are provided, and to check data type compatibility between required and provided resources. Provided these checks passed, the MIL compilation system, in conjunction with outside (user or environmental) inputs such as version availability and language choices, would select, compile (if necessary), and link the system. Incremental compilation is supported; for example, if the implementations for function XA change, the MIL compilation system will analyze the system structure and recompile and relink only those portions of the overall system affected by that change.

```
Module ABC
   provides a,b,c
   requires x,y
   consists-of function XA, module YBC

   function XA
      must-provide a
      requires x
      has-access-to Module Z
      real x, integer a
      realization
         version Pascal resources file
         (<Pascal_XA>) end Pascal
         version Ada resources file
         (<Ada_XA>) end Ada
         version FORTRAN resources file
         (<FORTRAN_XA>) end FORTRAN
   end XA

   Module YBC
      must-provide b, c
      requires a, y
      real y, integer a, b, c
   end YBC
end ABC
```

Figure 24: MIL Specification of a Simple Module

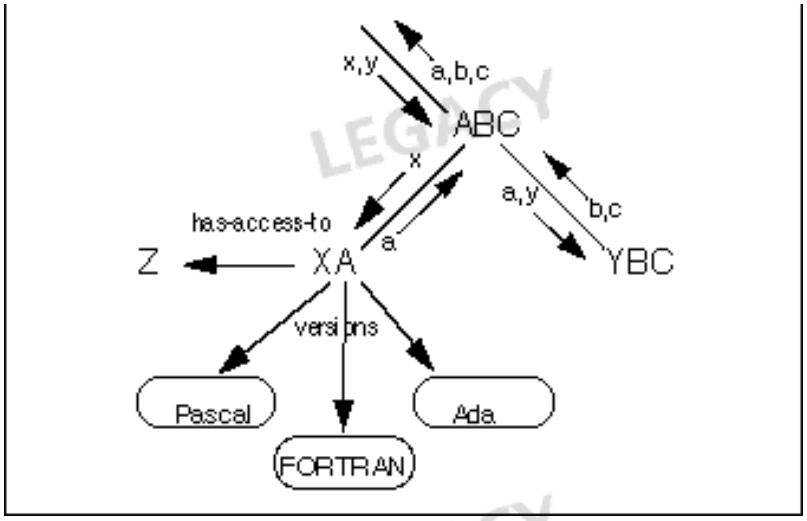


Figure 25: Graphical Representation

MILs do *not* attempt to do the following [Prieto-Diaz 86]:

- *Load compiled images.* This function is left to a separate facility within the development environment.
- *Define system function.* A MIL defines only the structure, not the function, of a system.
- *Provide type specifications.* A MIL is concerned with showing or identifying the separate paths of communication between modules. Syntactic checks along these communications paths may be performed by a MIL, but because MILs are independent of the language chosen to implement the modules they reference, such type checking will be limited to simple syntactic- not semantic- compatibility.
- Define embedded link-edit instructions.

Recently, MILs have been extended with notions of communication protocols [Garlan 94] and with constructs for defining semantic properties of system function. These extended MILs are referred to as Architecture Description Languages (ADLs).

Usage Considerations

MILs were developed to address the need for automated integration support when programming in the large; they are well-suited for representing the structure of a system or family of systems, and are typically used for project management and support. When adopting the use of MILs, an organization will need to consider the effect on its current system development and maintenance philosophy.

Because the structure of a software system can be explicitly represented in a MIL specification, separate documentation describing software structure may be unnecessary. This implies that if MILs are used to define the structure, then the architectural documentation of a given system will not become outdated.

Although some support is provided for ensuring data type compatibility, MILs typically lack the structures required to define or enforce protocol compatibility between modules, and the structures necessary to enforce semantic compatibility.

Maturity

The MESA system at Xerox PARC was developed during 1975 and has been used extensively within Xerox [Geschke 77, Mitchell 79, Prieto-Diaz 86]. Other MILs have been proposed, defined, and implemented, but most of these appear to have been within a research context. For example, MIL concepts have been used to help design and build software reuse systems such as Goguen's library interconnection language (LIL) that was extended by Tracz for use with parameterized Ada components [Tracz 93]. Zand, et al., describe a system called ROPCO that can be used to "facilitate the selection and integration of reusable modules" [Zand 93].

At the time of publication, however, there are no tools supporting MILs and little research in this area.[1] Recent MIL-based research has shifted focus and now centers around the themes of software reuse and architecture description languages (ADLs). Architecture Description Languages can be viewed as extended MILs in that ADLs augment the structural information of a MIL with information about communication protocols [Garlan 94] and system behavior.

Costs and Limitations

MILs are formal compilable languages. Developers will need training to understand and use a MIL effectively. Training in architectural concepts may also be required.

The lack of a formal semantic for defining module function has at least the following implications:

- *Limited inter-module consistency checking.* MIL-based consistency checking is limited to simple type checking and- if supported- simple protocol checking.
- *Limited consistency checking among module versions.* MILs lack the facilities to ensure that different versions of a module satisfy a common specification, and may potentially lead to inconsistent versions within a family.
- *Limited type checking.* If mixing languages with a system, a developer may need to augment standard MIL tools with more sophisticated type checking utilities. For example, data types may be represented differently in C than in Ada, but the simple type checking found in a typical MIL will not flag unconverted value passing between languages.

Dependencies

Incremental compilers and linkers are required by most MILs.

Alternatives

Alternatives to MILs include documenting the structure of a system externally, such as in an interface control document or a structure chart. Architecture Description Languages (ADLs) can also be used to define the structure of a system, and are believed to be the current direction for this technology area.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Module Interconnection Languages
Application category	Architectural Design (AP.1.3.1) Compiler (AP.1.4.2.3) Plan and Perform Integration (AP.1.4.4)
Quality measures category	Correctness (QM.1.3) Structuredness (QM.3.2.3) Reusability (QM.4.4)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2) Organization and Design (D.4.7) Performance (D.4.8) Systems Programs and Utilities (D.4.9)

References and Information Sources

[Cooprider 79]	Cooprider, Lee W. *The Representation of Families of Software Systems* (CMU-CS-79-116). Pittsburgh, PA: Computer Science Department, Carnegie Mellon University, 1979.
[DeRemer 76]	DeRemer, F. & Kron, H. "Programming-in-the-Large Versus Programming-in-the-Small." *IEEE Transactions on Software Engineering SE-2*, 2 (June 1976): 321-327.
[Garlan 94]	Garlan, David & Allen, Robert. "Formalizing Architectural Connection," 71-80. *Proceedings of the 16th International Conference on Software Engineering*. Sorrento, Italy, May 16-21, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[Geschke 77]	Geschke, C.; Morris, J.; & Satterthwaite, E. "Early Experience with MESA." *Communications of the ACM 20*, 8 (August 1977): 540-553.
[Mitchell 79]	Mitchell, J.; Maybury, W.; & Sweet, R. *MESA Language Manual* (CSL-79-3). Palo Alto, CA: Xerox Palo Alto Research Center, April 1979.
[Prieto-Diaz 86]	Prieto-Diaz, Ruben & Neighbors, James. "Module Interconnection Languages." *Journal of Systems and Software 6*, 4 (1986): 307-334.
[Tichy 79]	Tichy, W. F. "Software Development Control Based on Module Interconnection," 29-41. *Proceedings of the 4th International Conference on Software Engineering*. Munich, Germany, September 17-19, 1979. New York, NY: IEEE Computer Society Press, 1979.
[Tracz 93]	Tracz, W. "LILEANNA: a Parameterized Programming Language," 66-78. *Proceedings of the Second International Workshop on Software Reuse*. Lucca, Italy, March 24-26, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.
[Zand 93]	Zand, M., et al. "An Interconnection Language for Reuse at the Template/Module Level." *Journal of Systems and Software 23*, 1 (October 1993): 9-26.

Current Author/Maintainer

Mark Gerken, Air Force Rome Laboratory

External Reviewers

Will Tracz, Lockheed Martin Federal Systems, Owego, NY

Modifications

10 Jan 97 (original)

Footnotes

[1] Source: Will Tracz in *Re: External Review - MILS*, email to Bob Rosenstein (1996).

URL: http://www.sei.cmu.edu/str/descriptions/mil_body.html
Last Modified: 24 July 2008

References and Information Sources

[Garlan 94] Garlan, David & Allen, Robert. "Formalizing Architectural Connection," 71-80. *Proceedings of the 16th International Conference on Software Engineering*. Sorrento, Italy, May 16-21, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

References and Information Sources

[Geschke 77] Geschke, C.; Morris, J.; & Satterthwaite, E. "Early Experience with MESA." *Communications of the ACM 20*, 8 (August 1977): 540-553.

References and Information Sources

[Mitchell 79] Mitchell, J.; Maybury, W.; & Sweet, R. *MESA Language Manual* (CSL-79-3). Palo Alto, CA: Xerox Palo Alto Research Center, April 1979.

References and Information Sources

[Zand 93] Zand, M., et al. "An Interconnection Language for Reuse at the Template/Module Level." *Journal of Systems and Software 23*, 1 (October 1993): 9-26.

Notes

[1] Source: Will Tracz in *Re: External Review - MILS*, email to Bob Rosenstein (1996).

Multi-Level Secure Database Management Schemes

Software Technology Roadmap

Status

Advanced

Note

We recommend Computer System Security--An Overview *as prerequisite reading for this technology description.*

Purpose and Origin

Conventional database management systems (DBMS) do not recognize different security levels of the data they store and retrieve. They treat all data at the same security level. Multi-level secure (MLS) DBMS schemes provide a means of maintaining a collection of data with mixed security levels. The access mechanisms allow users or programs with different levels of security clearance to store and obtain only the data appropriate to their level.

Technical Detail

As shown in Figure 20, multi-level secure DBMS architecture schemes are categorized into two general types:

- the Trusted Subject architecture
- the Woods Hole architectures

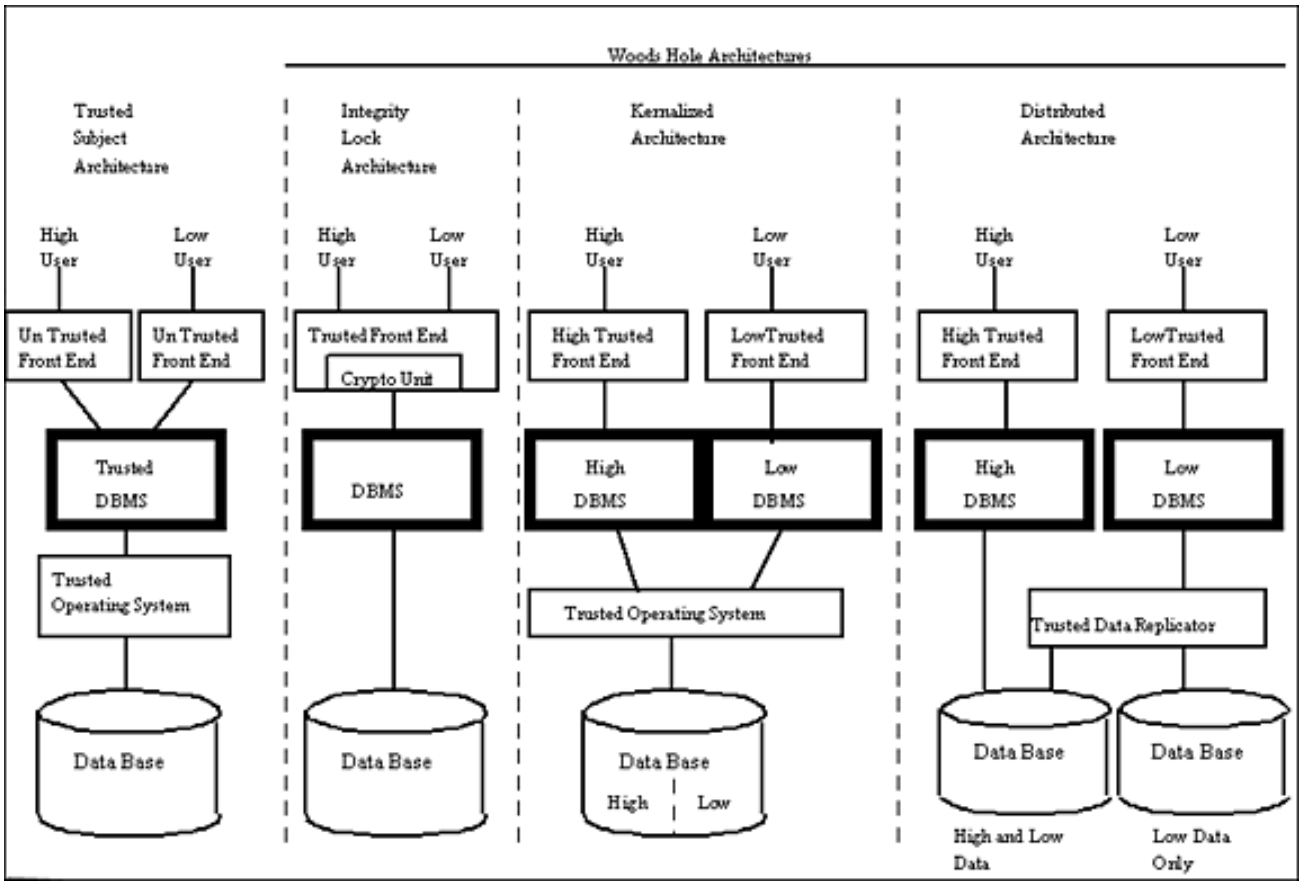


Figure 20: MLS DBMS Schemes

The Woods Hole architectures are named after an Air Force-sponsored study on multi-level data management security that was conducted at Woods Hole, Massachusetts.

The Trusted Subject architecture is a scheme that contains a trusted DBMS and operating system (see Trusted Operating Systems). The DBMS is custom-developed with all the required security policy (the security rules that must be enforced) developed in the DBMS itself. The DBMS uses the associated trusted operating system to make actual disk data accesses. This is the traditional way of developing MLS DBMS capabilities and can achieve high mandatory assurance for a particular security policy at the sacrifice of some DBMS functionality [Abrams 95]. This scheme results in a special purpose DBMS and operating system that requires a large amount of trusted code to be developed and verified along with the normal DBMS features.Trusted code provides security functionality and has been designed and developed using a rigorous process, tested, and protected from tampering in a manner that ensures the Designated Approving Authority (DAA) that it performs the security functions correctly. The DAA is the security official with the authority to say a system is secure and is permitted to be used. A benefit of the trusted subject architecture is that the DBMS has access to all levels of data at the same time, which minimizes retrieval and update processing. This scheme also can handle a wide range of sensitivity labels and supports complex access control. A sensitivity label identifies the classification level (e.g., confidential, secret) and a set of categories or compartments that apply to the data associated with the label.

The Woods Hole architectures assume that an untrusted (usually commercial-off-the-shelf (COTS)) DBMS is used to access data and that trusted code is developed around that DBMS to provide an overall secure DBMS system. The three different Woods Hole architectures address three different ways to wrap code around the untrusted DBMS.

The Integrity Lock architecture scheme places a trusted front end filter between the users and the

DBMS. The filter provides security for the MLS. When data is added to the database, the trusted front end filter adds an encrypted integrity lock to each unit of data added to the database. The lock is viewed by the DBMS as just another element in the unit stored by the DBMS. The encrypted lock is used to assure that the retrieved data has not been tampered with and contains the security label of the data. When data is retrieved, the filter decrypts the lock to determine if the data can be returned to the requester. The filter is designed and trusted to keep users separate and to store and provide data appropriate to the user. A benefit of this scheme is that an untrusted COTS DBMS can perform most indexed data storage and retrieval.

The Kernalized architecture scheme uses a trusted operating system and multiple copies of the DBMS; each is associated with a trusted front end. The trusted front end-DBMS pair is associated with a particular security level. Between the DBMS and the database, a portion of the trusted operating system keeps the data separated by security level. Each trusted front end is trusted to supply requests to the proper DBMS. The database is separated by security level. The trusted operating system separates the data when it is added to the database by a DBMS and combines the data when it is retrieved (if allowed by the security rules it enforces for the requesting DBMS). The high DBMS gets data combined from the high and low segments of the database. The low DBMS can only get data from the low segment of the database. A benefit of this scheme is that access control and separation of data at different classification levels is performed by a trusted operating system rather than the DBMS. Data at different security levels is isolated in the database, which allows for higher level assurance. Users interact with a DBMS at the user's single-session level.

The Distributed architecture scheme uses multiple copies of the trusted front end and DBMS, each associated with its own database storage. In this architecture scheme, low data is replicated in the high database. When data is retrieved, the DBMS retrieves it only from its own database. A benefit of this architecture is that data is physically separated into separate hardware databases. Since separate replicated databases are used for each security level, the front end does not need to decompose user query data to different DBMSs.

Castano and Abrams provide thorough discussions of these alternative architecture schemes and their merits [Castano 95, Abrams 95].

Usage Considerations

This technology is most likely to be used when relational databases must be accessed by users with different security clearances. This is typical of Command and Control systems. The different architectures suit different needs. The Trusted Subject architecture is best for applications where the trusted operating system and the hardware used in the architecture already provide an assured, trusted path between applications and the DBMS [Castano 95]. The Integrity Lock architecture provides the ability to label data down to the row (or record) level, the ability to implement a wide range of categories, and is easiest to validate [Castano 95]. The Kernalized architecture scheme is suited to MLS DBMS systems with more simple table structures because it is economical and easier to implement for simple structures [Castano 95]. The Distributed architecture is best suited for DBMSs where physical separation of data by security level is required [Abrams 95].

Maturity

The four different architectures have different maturity characteristics. As of August 1996, an R&D A1[1] system and six commercial[2] DBMSs have been implemented using the Trusted Subject architecture scheme for different assurance levels and security policies. One R&D system and one commercial DBMS have been implemented using the Integrity Lock architecture scheme. One R&D system and one commercial DBMS have been implemented using the Kernalized architecture

scheme [Castano 95]. The Distributed architecture scheme has only been used in prototype systems because of the high performance cost of the replicater, although one commercial DBMS claims to have this feature [Abrams 95]. This DBMS however, has not been evaluated by the National Computer Security Center (NCSC) [TPEP 96].

Costs and Limitations

Each of the different MLS architecture schemes has different costs and limitations. The Trusted Subject architecture scheme has a closely linked DBMS and Operating System that must be proven trusted together. This makes it hardest to validate and gives it the highest accreditation cost compared to the other schemes. The Integrity Lock architecture scheme requires that a Crypto Key management system is implemented and supported in operation. The Kernalized architecture requires a DBMS for each security level, which makes it expensive as more than two or three levels are considered. The Distributed architecture requires a different hardware platform for each security level and the data replicater provides a heavy processor and I/O load for high access data.

Dependencies

The MLS architecture schemes have individual dependencies. The Trusted Subject scheme is dependent on trusted schemes for a related DBMS and operating system. The Integrity Lock scheme is dependent on cryptographic technologies to provide the integrity lock. The Kernalized architecture scheme depends on Trusted Operating Systems technologies. The Distributed architecture scheme is dependent on efficient automatic data replication techniques.

Alternatives

The alternative to these technologies is to use a single-level DBMS and use manual review of retrieved data or have every user cleared for the data in the database. That may not be feasible in a Command and Control system.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Multi-Level Secure Database Management Schemes
Application category	Data Management Security (AP.2.4.2)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating Systems Security & Protection (D.4.6) Security & Protection (K.6.5) Computer-Communications Network Security and Protection (C.2.0)

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Castano 95] Castano, Silvana, et al. *Database Security*. New York, NY: ACM Press, 1995.

[DoD 85] *Department of Defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC)* (DoD 5200.28-STD 1985). Fort Meade, MD: Department of Defense, 1985. Also available WWW <URL: http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html> (1985).

[TPEP 96] *Trusted Product Evaluation Program Evaluated Product List* [online]. Available WWW <URL: http://www.radium.ncsc.mil/tpep/index.html> (1996).

Current Author/Maintainer

Tom Mills, Lockheed Martin

Modifications

10 Jan 97 (original)

Footnotes

[1] An A1 system is one that meets the highest (most stringent) set of requirements in the Department of Defense Trusted Computer Systems Evaluation Criteria (the Orange Book) [DoD 85]. See Trusted Operating Systems for a further description of the classes of trusted operating systems.

[2] A commercial DBMS does not imply a general-purpose DBMS. It means that it can be packaged and sold to other people. If a MLS DBMS has been developed to provide specific security functions that customers need, and the customer is willing to be restricted to that set of functions and use the same hardware and support software, then it can be sold as a product. It is then a commercial DBMS. The six commercial DBMSs that have been implemented with the Trusted Subject architecture are all different from each other, as they have been developed with different security policies for different hardware and software environments.

URL: http://www.sei.cmu.edu/str/descriptions/mlsdms_body.html
Last Modified: 24 July 2008

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays*. Los Alamitos, CA: IEEE Computer Society Press, 1995.

References and Information Sources

[Castano 95] Castano, Silvana, et al. *Database Security*. New York, NY: ACM Press, 1995.

Notes

[1] An A1 system is one that meets the highest (most stringent) set of requirements in the Department of Defense Trusted Computer Systems Evaluation Criteria (the Orange Book) [DoD 85]. See Trusted Operating Systems for a further description of the classes of trusted operating systems.

Notes

[2] A commercial DBMS does not imply a general-purpose DBMS. It means that it can be packaged and sold to other people. If a MLS DBMS has been developed to provide specific security functions that customers need, and the customer is willing to be restricted to that set of functions and use the same hardware and support software, then it can be sold as a product. It is then a commercial DBMS. The six commercial DBMSs that have been implemented with the Trusted Subject architecture are all different from each other, as they have been developed with different security policies for different hardware and software environments.

References and Information Sources

[TPEP 96] *Trusted Product Evaluation Program Evaluated Product List* [online]. Available WWW <URL: http://www.radium.ncsc.mil/tpep/index.html> (1996).

Related Topics

Data Management Security (AP.2.4.2)

- Multi-Level Secure Database Management Schemes
- Trusted DBMS

Related Topics

Security (see also QM.2.1.4) (QM.2.1.5)

- Common Management Information Protocol
- Computer System Security -- an Overview
- Distributed Computing Environment
- Firewalls and Proxies
- Intrusion Detection
- Multi-Level Secure One Way Guard with Random Acknowledgment
- Multi-Level Secure Database Management Schemes
- Network Management -- An Overview
- Rule-Based Intrusion Detection
- Simple Network Management Protocol
- Statistical-Based Intrusion Detection
- Trusted Operating Systems
- Virus Detection

References and Information Sources

[DoD 85] *Department of Defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC)* (DoD 5200.28-STD 1985). Fort Meade, MD: Department of Defense, 1985. Also available WWW <URL: http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html> (1985).

References and Information Sources

[Weiderman 97] Weiderman, Nelson; Northrop, Linda; Smith, Dennis; Tilley, Scott; & Wallnau, Kurt; *Implications of Distributed Object Technology for Reengineering (CMU/SEI-97-TR-005 ADA326945)*. Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, [online]. Available WWW URL: http://www.sei.cmu.edu/publications/documents/97.reports/97tr005/97tr005abstract.html (1997).

References and Information Sources

[Carr 98] Carr, David F. *Web-Enabling Legacy Data When Resources Are Tight.* Internet World. August 10 1998.

References and Information Sources

[Karpinski 98] Karpinski, Richard. *Databases, Tools Push XML Into Enterprise.* Internet Week Online. Available WWW URL: http://www.internetwk.com/news1198/news111698-3.htm (November 1998).

References and Information Sources

[Shklar] Shklar, Leon. *Web Access to Legacy Data* [online]. Available WWW: URL: http://athos.rutgers.edu/~shklar/web-legacy/summary.html.

References and Information Sources

[Eichman 95] David Eichmann. *Application Architectures for Web-Based Data Access* [online]. Available WWW: URL: http://www.cs.rutgers.edu/~shklar/www4/eichmann.html

References and Information Sources

[Phoenix Group 97]	Phoenix Group. *Legacy Systems Wrapping with Objects* [online]. Available WWW URL: http://www.phxgrp.com/jodewp.htm.

References and Information Sources

[De Lucia 97] De Lucia, A.; Di Lucca, G.A.; Fasolino, A.R.; Guerra, P.; & Petruzzelli, S. *Migrating Legacy Systems towards Object-Oriented Platforms*, International Conference of Software Maintenance (ICSM97), 1997.

References and Information Sources

[Bisdal 97] Bisdal, Jesus; Lawless, Deirdre; Wu, Bing; Grimson, Jane; Wade, Vincent; Richardson, Ray; & O'Sullivan, D. *An Overview of Legacy Information System Migration*, Proceedings of the 4th Asian-Pacific Software Engineering and International Computer Science Conference (APSEC 97, ICSC 97), 1997.

Empty Taxonomy Category

Currently, no technology descriptions are classified into the taxonomy category you have selected.

References and Information Sources

[Linger 94] Linger, R.C. "Cleanroom Process Model." *IEEE Software 11*, 2 (March 1994): 50-58.

References and Information Sources

[Mills 87] Mills, H.; Dyer, M.; & Linger, R. "Cleanroom Software Engineering." *IEEE Software 4*, 5 (September 1987): 19-25.

Glossary Term

Correctness

the degree to which a system or component is free from faults in its specification, design, and implementation [IEEE 90].

References and Information Sources

[Hausler 94] Hausler, P. A.; Linger, R. C.; & Trammel, C. J. "Adopting Cleanroom Software Engineering with a Phased Approach." *IBM Systems Journal 33*, 1 (1994): 89-109.

Notes

[1] STARS: Software Technology for Adaptable Reliable Systems

References and Information Sources

[STARSSCAI 95] Air Force/STARS Demonstration Project Home Page [online]. Available WWW <URL: http://www.asset.com/stars/afdemo/home.html> (1995).

References and Information Sources

[Sherer 96a] Sherer, S. W. *Cleanroom Software Engineering- the Picatinny Experience* [online]. Available WWW <URL: http://software.pica.army.mil/cleanroom/cseweb.html> (1996).

References and Information Sources

[Sherer 96b] Sherer, S.W.; Kouchakdjian, A.; & Arnold, P.G. "Experience Using Cleanroom Software Engineering." *IEEE Software 13*, 3 (May 1996): 69-76.

References and Information Sources

[Ett 96] Ett, William & Trammell, Carmen. *A Guide to Integration of Object-Oriented Methods and Cleanroom Software Engineering* [online]. Available WWW <URL: http://www.asset.com/stars/loral/cleanroom/oo/guidhome.htm> (1996).

References and Information Sources

[Linger 96b] Linger, R.C. & Trammel, C.J. *Cleanroom Software Engineering Reference Model* (CMU/SEI-96-TR-022). Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute, 1996.

References and Information Sources

[Linger 96a] Linger, R.C.; Paulk, M.C.; & Trammel, C.J. *Cleanroom Software Engineering Implementation of the CMM for Software* (CMU/SEI-96-TR-023). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

References and Information Sources

[Paulk 93] Paulk, M.; Curtis B.; Chrissis, M.; & Weber, C. *Capability Maturity Model for Software* Version 1.1 (CMU/SEI-96-TR-24, ADA263403). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

Related Topics

Detailed Design (Design Notations, Design Techniques) (AP.1.3.5)

- Cleanroom Software Engineering
- Dynamic Simulation
- Finite State Automata
- Object-Oriented Design
- Peer Reviews
- Personal Software Process for Module-Level Development
- Probabilistic Automata
- Rate Monotonic Analysis
- Software Inspections
- Software Walkthroughs
- Stochastic Methods
- Structured Analysis and Design

Related Topics

Component Testing (AP.1.4.3.5)

- Cleanroom Software Engineering
- Halstead Complexity Measures
- Maintainability Index Technique for Measuring Program Maintainability
- Personal Software Process for Module-Level Development
- Simplex Architecture

Related Topics

Performance Testing (Statistical Testing) (AP.1.5.3.5)

- Cleanroom Software Engineering
- Rate Monotonic Analysis

Related Topics

Availability/Robustness (Error Tolerance, Fault Tolerance, Fail Safe, Fail Soft) (QM.2.1.1)

- Cleanroom Software Engineering
- Personal Software Process for Module-Level Development
- Simplex Architecture
- Software Inspections

Glossary Term

Usability

the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component [IEEE 90].

Glossary Term

Scalability

the ease with which a system or component can be modified to fit the problem area.

References and Information Sources

[Schussel 96] Schussel, George. *Client/Server Past, Present, and Future* [online]. Available WWW <URL: http://news.dci.com/geos/dbsejava.htm> (1995).

References and Information Sources

[Edelstein 94] Edelstein, Herb. "Unraveling Client/Server Architecture." *DBMS 7*, 5 (May 1994): 34 (7).

References and Information Sources

[OMG 96] Object Management Group home page [online]. Available WWW <URL: http://www.omg.org> (1996).

References and Information Sources

[Shelton 93] Shelton, Robert E. "The Distributed Enterprise (Shared, Reusable Business Models the Next Step in Distributed Object Computing)." *Distributed Computing Monitor 8*, 10 (October 1993): 1.

References and Information Sources

[Adler 95] Adler, R. M. "Distributed Coordination Models for Client/Sever Computing." *Computer 28*, 4 (April 1995): 14-22.

Related Topics

Usability (QM.2.3)

- Client/Server Software Architectures
- Graphical User Interface Builders
- Two Tier Software Architectures

Notes

[1] The OSI model is a framework for defining communications protocols. It consists of seven layers of protocols that range from low level methods for dealing with a physical communications medium, to high level methods for dealing with the communications needs of user applications. Developed by the International Standards Organization (ISO), specific protocols have been designed to implement the functionality specified by the OSI model.

Notes

[2] International Telegraph and Telephone Consultative Committee: This organization is part of the United National International Telecommunications Union (ITU) and is responsible for making technical recommendations about telephone and data communications systems.

Notes

[3] **International Organization for Standardization (ISO).** A voluntary, non-treaty organization founded in 1946 which is responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of the 89 member countries, including ANSI for the U.S.

International Electrotechnical Commission (NEC). The international standards and conformity assessment body for all fields of electrotechnology. IEC and ISO technical committees collaborate in fields of mutual interest.

References and Information Sources

[Vallillee 96] Vallillee, Tyler. *SNMP & CMIP: An Introduction To Network Management* [online]. Available WWW <URL: http://www.inforamp.net/~kjvallil/t/snmp.html> (1996).

References and Information Sources

[Stallings 93] Stallings, William. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards.* Reading, MA: Addison-Wesley, 1993.

Notes

[4] The ITU is an international organization within which governments and the private sector coordinate global telecom networks and services. It also develops standards to facilitate the interconnection of telecommunication systems on a worldwide scale regardless of the type of technology used.

Notes

[5] A management architecture framework developed by the International Telecommunication Union (ITU), which provides an environment for interfacing a telecommunication network with computer systems to provide different management functions at several different levels. The framework allows the management of business information between different components (operations systems, communication equipment, network and computer systems) and provides control of service operations and information flow.

Common Management Information Protocol

Software Technology Roadmap

Status

Advanced

Note

We recommend Network Management--An Overview *as prerequisite reading for this technology description.*

Purpose and Origin

Common Management Information Protocol (CMIP) is an Open Systems Interconnection (OSI)[1] -based network management protocol that supports information exchange between network management applications and management agents. CMIP is part of the X.700 (CCITT[2] number for the OSI Management Framework, also designated as ISO/IEC 7498-4[3]) OSI series of management standards. Its design is similar to the Simple Network Management Protocol (SNMP). CMIP was developed and funded by government and corporations to replace and makeup for the deficiencies in SNMP, thus improving the capabilities of network management systems.

Technical Detail

CMIP is a well designed protocol that defines how network management information is exchanged between network management applications and management agents. It uses an ISO reliable connection-oriented transport mechanism and has built in security that supports access control, authorization and security logs. The management information is exchanged between the network management application and management agents thru managed objects. Managed objects are a characteristic of a managed device that can be monitored, modified or controlled and can be used to perform tasks. The network management application can initiate transactions with management agents using the following operations:

- ACTION - Request an action to occur as defined by the managed object.
- CANCEL_GET - Cancel an outstanding GET request.
- CREATE - Create an instance of a managed object.
- DELETE - Delete an instance of a managed object.

- GET - Request the value of a managed object instance.
- SET - Set the value of a managed object instance.

A management agent can initiate a transaction with the network management application using the EVENT_REPORT operation. This operation can be used to send notifications or alarms to the network management application based upon predetermined conditions set by the network management application using the ACTION operation.

CMIP does not specify the functionality of the network management application, it only defines the information exchange mechanism of the managed objects and not how the information is to be used or interpreted.

The major advantages of CMIP over SNMP are [Vallillee 96]:

- CMIP variables not only relay information, but also can be used to perform tasks. This is impossible under SNMP.
- CMIP is a safer system as it has built in security that supports authorization, access control, and security logs.
- CMIP provides powerful capabilities that allow management applications to accomplish more with a single request.
- CMIP provides better reporting of unusual network conditions

The CMIP specification for TCP/IP networks is called CMOT (CMIP Over TCP) and the version for IEEE 802 LAN's is called CMOL (CMIP Over LLC) [Stallings 93].

Usage Considerations

CMIP is widely used in the telecommunication domain and telecommunication devices typically support CMIP. The International Telecommunication Union (ITU)[4] endorses CMIP as the protocol for the management of devices in the Telecommunication Management Network (TMN)[5] standard.

The CMIP protocol is designed to run on the ISO protocol stack [Stallings 93]. However, the technology standard used today in most LAN environments is TCP/IP and most LAN devices only support SNMP. Implementations of CMOT are extremely scarce.

CMIP requires a large amount of system resources, this has resulted in very few implementations. Additionally, CMIP is very complex thus making it difficult to program; therefore skilled personnel with specialized training may be required to deploy, maintain and operate a CMIP based network management system.

Maturity

CMIP was developed over a decade ago; however few implementations exist because of the problems described above in Usage Considerations.

Costs and Limitations

Systems may not be capable of supporting the resource requirements of CMIP and difficulties may exist in the procurement of CMIP software because of limited availability.

Alternatives

SNMP is widely available and is the de facto standard network management protocol; however, it does not provide all of the functionality of CMIP. SNMP deficiencies are discussed in Usage Considerations for SNMP.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Common Management Information Protocol (CMIP)
Application category	Protocols (AP.2.2.3) Network Management (AP.2.2.2)
Quality measures category	Maintainability (QM.3.1) Simplicity (QM.3.2.2) Complexity (QM.3.2.1) Efficiency/Resource Utilization (QM.2.2) Scalability (QM.4.3) Security (QM.2.1.5)
Computing reviews category	Network Operations (C.2.3) Distributed Systems (C.2.4)

References and Information Sources

[Korinko 96]	Korinko, Joe. *CMIP-Common Management Information Protocol* [online]. Available WWW <URL: http://www.rit.edu/~jek0539/icsa750/exam2/ex2pg1.htm> (1996).
[Stallings 93]	Stallings, William. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards.* Reading, MA: Addison-Wesley, 1993.

[Vallillee 96] Vallillee, Tyler. *SNMP & CMIP: An Introduction To Network Management* [online]. Available WWW <URL: http://www.inforamp.net/~kjvallil/t/snmp.html> (1996).

[X.700 96] *X.700 and Other Network Management Services* [online]. Available WWW <URL: http://ganges.cs.tcd.ie/4ba2/x700/index.html> (1996).

Current Author/Maintainer

Dan Plakosh, SEI

Modifications

9 February 98: minor modifications

13 May 97 (Original)

Footnotes

[1] The OSI model is a framework for defining communications protocols. It consists of seven layers of protocols that range from low level methods for dealing with a physical communications medium, to high level methods for dealing with the communications needs of user applications. Developed by the International Standards Organization (ISO), specific protocols have been designed to implement the functionality specified by the OSI model.

[2] International Telegraph and Telephone Consultative Committee: This organization is part of the United National International Telecommunications Union (ITU) and is responsible for making technical recommendations about telephone and data communications systems.

[3] **International Organization for Standardization (ISO).** A voluntary, non-treaty organization founded in 1946 which is responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of the 89 member countries, including ANSI for the U.S.
International Electrotechnical Commission (NEC). The international standards and conformity assessment body for all fields of electrotechnology. IEC and ISO technical committees collaborate in fields of mutual interest.

[4] The ITU is an international organization within which governments and the private sector coordinate global telecom networks and services. It also develops standards to facilitate the interconnection of telecommunication systems on a worldwide scale regardless of the type of technology used.

[5] A management architecture framework developed by the International Telecommunication Union (ITU), which provides an environment for interfacing a telecommunication network with computer systems to provide different

management functions at several different levels. The framework allows the management of business information between different components (operations systems, communication equipment, network and computer systems) and provides control of service operations and information flow.

URL: http://www.sei.cmu.edu/str/descriptions/cmip_body.html
Last Modified: 24 July 2008

Related Topics

Protocols (AP.2.2.3)

- ATM
- Common Management Information Protocol
- Network Management -- An Overview
- OSI
- Simple Network Management Protocol
- TCP/IP
- X.25

Related Topics

Network Management (AP.2.2.2)

- Common Management Information Protocol
- Network Management -- An Overview
- Simple Network Management Protocol

Related Topics

Simplicity (QM.3.2.2)

- Common Management Information Protocol
- Simple Network Management Protocol

Related Topics

Complexity (Apparent, Inherent) (QM.3.2.1)

- Common Management Information Protocol
- Cyclomatic Complexity
- Distributed Computing Environment
- Halstead Complexity Measures
- Java
- Remote Procedure Call
- Simple Network Management Protocol

Related Topics

Efficiency/Resource Utilization (Speed, Compactness) (QM.2.2)

- Common Management Information Protocol
- Simple Network Management Protocol
- Transaction Processing Monitor Technology

References and Information Sources

[OMG 96] Object Management Group home page [online]. Available WWW <URL: http://www.omg.org> (1996).

Common Object Request Broker Architecture

Software Technology Roadmap

Status

ADVANCED

Note

We recommend Object Request Broker, as prerequisite reading for this technology description.

Purpose and Origin

The Common Object Request Broker Architecture (CORBA) is a *specification* of a standard architecture for object request brokers (ORBs) (see Object Request Broker). A standard architecture allows vendors to develop ORB products that support application *portability* and *interoperability* across different programming languages, hardware platforms, operating systems, and ORB implementations:

"Using a CORBA-compliant ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call, and is responsible for finding an object that can implement the request, passing it the parameters, invoking its method, and returning the results of the invocation. The client does not have to be aware of where the object is located, its programming language, its operating system or any other aspects that are not part of an object's interface" [OMG 96]. The "vision" behind CORBA is that distributed systems are conceived and implemented as distributed objects. The interfaces to these objects are described in a high-level, architecture-neutral specification language that also supports object-oriented design abstraction. When combined with the Object Management Architecture (see Technical Detail), CORBA can result in distributed systems that can be rapidly developed, and can reap the benefits that result from using high-level building blocks provided by CORBA, such as *maintainability* and *adaptability*.

The CORBA specification was developed by the Object Management Group (OMG), an industry group with over six hundred member companies representing computer manufacturers, independent software vendors, and a variety of government and academic organizations [OMG 96]. Thus, CORBA specifies an industry/consortium standard, not a "formal" standard in the IEEE/

ANSI/ISO sense of the term. The OMG was established in 1988, and the initial CORBA specification emerged in 1992. Since then, the CORBA specification has undergone significant revision, with the latest major revision (CORBA v2.0) released in July 1996.

Technical Detail

CORBA ORBs are middleware mechanisms (see Middleware), as are all ORBs. CORBA can be thought of as a generalization of remote procedure call (RPC) that includes a number of refinements of RPC, including:

- a more abstract and powerful interface definition language
- direct support for a variety of object-oriented concepts
- a variety of other improvements and generalizations of the more primitive RPC

CORBA and the Object Management Architecture. It is impossible to understand CORBA without appreciating its role in the Object Management Architecture (OMA), shown in Figure 2. The OMA is itself a specification (actually, a collection of related specifications) that defines a broad range of services for building distributed applications. The OMA goes far beyond RPC in scope and complexity. The distinction between CORBA and the OMA is an important one because many services one might expect to find in a middleware product such as CORBA (e.g., naming, transaction, and asynchronous event management services) are actually specified as services in the OMA. For reference, the OMA reference architecture encompasses both the ORB and remote service/object depicted in Figure 21, Middleware.

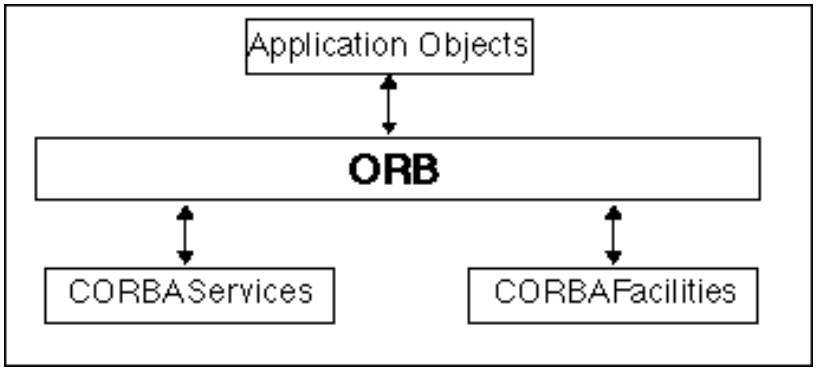


Figure 2: Object Management Architecture

OMA services are partitioned into three categories: CORBAServices, CORBAFacilities, and ApplicationObjects. The ORB (whose details are specified by CORBA) is a communication infrastructure through which applications access these services, and through which objects interact with each other. CORBAServices, CORBAFacilities, and ApplicationObjects define different categories of objects in the OMA; these objects (more accurately object *types*) define a range of functionality needed to support the development of distributed software systems.

- CORBAServices are considered fundamental to building non-trivial

distributed applications. These services currently include asynchronous event management, transactions, persistence, externalization, concurrency, naming, relationships, and lifecycle. Table 1 summarizes the purpose of each of these services.

- CORBAFacilities may be useful for distributed applications in some settings, but are not considered as universally applicable as CORBAServices. These "facilities" include: user interface, information management, system management, task management, and a variety of "vertical market" facilities in domains such as manufacturing, distributed simulation, and accounting.
- Application Objects provide services that are particular to an application or class of applications. These are not (currently) a topic for standardization within the OMA, but are usually included in the OMA reference model for completeness, i.e., objects are either application-specific, support common facilities, or are basic services.

Table 1: Overview of CORBA Services

Naming Service	Provides the ability to bind a name to an object. Similar to other forms of directory service.
Event Service	Supports asynchronous message-based communication among objects. Supports chaining of event channels, and a variety of producer/consumer roles.
Lifecycle Service	Defines conventions for creating, deleting, copying and moving objects.
Persistence Service	Provides a means for retaining and managing the persistent state of objects.
Transaction Service	Supports multiple transaction models, including mandatory "flat" and optional "nested" transactions.
Concurrency Service	Supports concurrent, coordinated access to objects from multiple clients.
Relationship Service	Supports the specification, creation and maintenance of relationships among objects.
Externalization Service	Defines protocols and conventions for externalizing and internalizing objects across processes and across ORBs.

CORBA in detail. Figure 3 depicts most of the basic components and interfaces

defined by CORBA. This figure is an expansion of the ORB component of the OMA depicted in Figure 2.

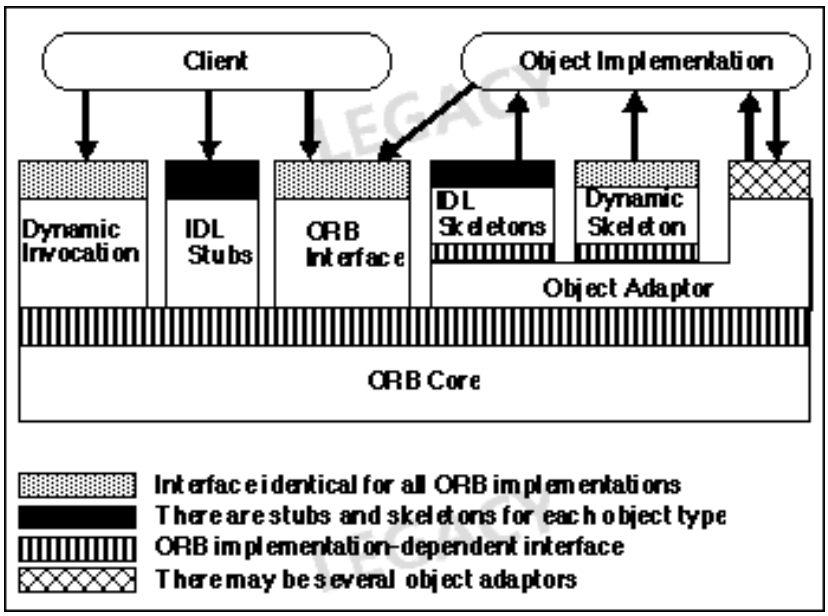


Figure 3: Structure of CORBA Interfaces

One element (not depicted in Figure 2) that is crucial to the understanding of CORBA is the interface definition language (IDL) processor. All objects are defined in CORBA (actually, in the OMA) using IDL. IDL is an object-oriented interface definition formalism that has some syntactic similarities with C++. Unlike C++, IDL can only define interfaces; it is not possible to specify behavior in IDL. Language mappings are defined from IDL to C, C++, Ada95, and Smalltalk80.

An important point to note is that CORBA specifies that clients and object implementations can be written in different programming languages and execute on different computer hardware architectures and different operating systems, and that clients and object implementations can not detect any of these details about each other. Put another way, the IDL interface completely defines the interface between clients and objects; all other details about objects (such as their implementation language and location) can be made "transparent."

Table 2 summarizes the components of CORBA and their functional role.

Table 2: Components of the CORBA Specification

ORB Core	The CORBA runtime infrastructure. The interface to the ORB Core is not defined by CORBA, and will be vendor proprietary.

ORB Interface	A standard interface (defined in IDL) to functions provided by all CORBA- compliant ORBs.
IDL Stubs	Generated by the IDL processor for each interface defined in IDL. Stubs hide the low-level networking details of object communication from the client, while presenting a high-level, object type-specific application programming interface (API).
Dynamic Invocation Interface (DII)	An alternative to stubs for clients to access objects. While stubs provide an object type-specific API, DII provides a generic mechanism for constructing requests at run time (hence "dynamic invocation"). An interface repository (another CORBA component not illustrated in Figure 2) allows some measure of type checking to ensure that a target object can support the request made by the client.
Object Adaptor	Provides extensibility of CORBA- compliant ORBs to integrate alternative object technologies into the OMA. For example, adaptors may be developed to allow remote access to objects that are stored in an object-oriented database. Each CORBA-compliant ORB must support a specific object adaptor called the Basic Object Adaptor (BOA) (not illustrated in Figure 2). The BOA defines a standard API implemented by all ORBs.
IDL Skeletons	The server-side (or object implementation-side) analogue of IDL stubs. IDL skeletons receive requests for services from the object adaptor, and call the appropriate operations in the object implementation.

Dynamic Skeleton Interface (DSI)	The server-side (or object implementation-side) analogue of the DII. While IDL skeletons invoke specific operations in the object implementation, DSI defers this processing to the object implementation. This is useful for developing bridges and other mechanisms to support inter-ORB interoperation.

Usage Considerations

Compliance. As noted, CORBA is a specification, not an implementation. Therefore, the question of compliance is important: How does a consumer know if a product is CORBA-compliant, and, if so, what does that mean? CORBA compliance is defined by the OMG:

"The minimum required for a CORBA-compliant system is adherence to the specifications in CORBA Core and one mapping" [CORBA 96] where "mapping" refers to a mapping from IDL to a programming language (C, C++ or Smalltalk80; Ada95 is specified but has not been formally adopted by the OMG at the time of this writing). The CORBA Core (*not* the same as the ORB Core denoted in Figure 3 and Table 2) is defined for compliance as including the following:

- the interfaces to all of the elements depicted in Figure 3
- interfaces to the interface repository (not shown in Figure 3)
- a definition of IDL syntax and semantics
- the definition of the object model that underlies CORBA (e.g., what is an object, how is it defined, where do they come from)

Significantly, the CORBA Core does *not* include CORBA interoperability, nor does it include interworking, the term used to describe how CORBA is intended to work with Microsoft's COM (see Component Object Model (COM), DCOM, and Related Capabilities). A separate but related point is that CORBA ORBs need not provide implementations of any OMA services.

There are as yet no defined test suites for assessing CORBA compliance. Users must evaluate vendor claims on face value, and assess the likelihood of vendor compliance based upon a variety of imponderables, such as the role played by the vendor in the OMG; vendor market share; and press releases and testimonials. Hands-on evaluation of ORB products is an absolute necessity. However, given the lack of a predefined compliance test suite, the complexity of the CORBA specification (see next topic), and the variability of vendor implementation choices, even this will be inadequate to fully assess "compliance."

Although not concerned with compliance testing in a formal sense, one organization has developed an operational testbed for demonstrating ORB

interoperability [CORBANet 96]. It is conceivable that other similar centers may be developed that address different aspects of CORBA (e.g., real time, security), or that do formal compliance testing. However, no such centers exist at the time of this writing.

Complexity. CORBA is a complex specification, and considerable effort may be required to develop expertise in its use. A number of factors compound the inherent complexity of the CORBA specification.

- While CORBA defines a standard, there is great latitude in many of the implementation details- ORBs developed by different vendors may have significantly different features and capabilities. Thus, users must learn a specification, the way vendors implement the specification, and their value-added features (which are often necessary to make a CORBA product usable).
- While CORBA makes the development of distributed applications easier than with previous technologies, this ease of use may be deceptive: The difficult issues involved in designing robust distributed systems still remain (e.g., performance prediction and analysis, failure mode analysis, consistency and caching, and security).
- Facility with CORBA may require deep expertise in related technologies, such as distributed systems design, distributed and multi-threaded programming and debugging; inter-networking; object-oriented design, analysis, and programming. In particular, expertise in object-oriented technology may require a substantial change in engineering practice, with all the technology transition issues that implies (see The Technology Adoption Challenge).

Stability. CORBA (and the OMA) represent a classical model of distributed computing, despite the addition of object-oriented abstraction. Recent advances in distributed computing have altered the landscape CORBA occupies. Specifically, the recent emergence of mobile objects via Java (see Java), and the connection of Java with "web browser" technologies has muddied the waters concerning the role of CORBA in future distributed systems. CORBA vendors are responding by supporting the development of "ORBlets", i.e., Java applets that invoke the services of remote CORBA objects. However, recent additions to Java support remote object invocation directly in a native Java form. The upshot is that, at the time of this writing, there is great instability in the distributed object technology marketplace.

Industry standards such as CORBA have the advantage of flexibility in response to changes in market conditions and technology advances (in comparison, formal standards bodies move much more slowly). On the other hand, changes to the CORBA specifications- while technically justified- have resulted in unstable ORB implementations. For example, CORBA v2.0, released in July 1995 with revisions in July 1996, introduced features to support interoperation among different vendor ORBs. These features are not yet universally available in all CORBA ORBs, and those ORBs that implement these features do so in uneven ways. Although the situation regarding interoperation among CORBA ORBs is improving, instability of implementations is the price paid for flexibility and evolvability of specification.

The OMA is also evolving, and different aspects are at different maturity levels. For instance, CORBAFacilities defines more of a framework for desired services than a specification suitable for implementation. The more fundamental CORBAServices, while better defined, are not rigorously defined; a potential consequence is that different vendor implementations of these services may differ widely both in performance and in semantics. The consequence is particularly troubling in light of the new interoperability features; prior to inter-ORB interoperability the lack of uniformity among CORBAServices implementations would not have been an issue.

Maturity

A large and growing number of implementations of CORBA are available in the marketplace, including implementations from most major computer manufacturers and independent software vendors. See Object Request Broker for a listing of available CORBA-compliant ORBs. CORBA ORBs are also being developed by university research and development projects, for example Stanford's Fresco, XeroxPARC's ILU, Cornell's Electra, and others.

At the same time, it must be noted that not all CORBA ORBs are equally mature, nor has the OMA sufficiently matured to support the vision that lies behind CORBA (see Purpose and Origin). While CORBA and OMA products are maturing and are being used in increasingly complex and demanding situations, the specifications and product implementations are not entirely stable. This is in no small way a result of the dynamism of distributed object technology and middleware in general and is no particular fault of the OMG. Fortunately techniques exist for evaluating technology in the face of such dynamism [Wallace 96, Brown 96].

Costs and Limitations

Costs and limitations include the following:

- *Real time.* CORBA v2.0 does not address real-time issues.
- *Programming language support.* IDL is a "least-common denominator" language. It does not fully exploit the capabilities of programming languages to which it is mapped, especially where the definition of abstract types is concerned.
- *Pricing and licensing*. The price of ORBs varies greatly, from a few hundred to several thousand dollars. Licensing schemes also vary.
- *Training.* Training is essential for the already experienced programmer: five days of hands-on training for CORBA programming fundamentals is suggested [Mowbray 93].
- *Security.* CORBA specifies only a minimal range of security mechanisms; more ambitious and comprehensive mechanisms have not yet been adopted by the OMG. Deng discusses the potential integration of security into CORBA-based systems [Deng 95].

Dependencies

Dependencies include the following:

- TCP/IP is needed to support the CORBA-defined inter-ORB interoperability protocol (IIOP).
- Most commercial CORBA ORBs rely on C++ as the principal client and server programming environment. Java-specific ORBs are also emerging.

Alternatives

Alternatives include the following:

- The Open Group's Distributed Computing Environment (DCE) is sometimes cited as an alternative "open" specification for distributed computing (see Distributed Computing Environment).
- Where openness is not a concern and PC platforms are dominant, Microsoft's COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities) may be suitable alternatives.
- Other middleware technologies may be appropriate in different settings (e. g., message-oriented middleware (see Message-Oriented Middleware)).

Complementary Technologies

Complementary technologies include the following:

- Java and/or web browsers can be used in conjunction with CORBA, although precise usage patterns have not yet emerged and are still highly volatile.
- Object-oriented database management systems (OODBMS) vendors are developing object adaptors to support more robust three-tier architecture (see Three Tier Software Architectures) development using CORBA.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Common Object Request Broker Architecture
Application category	Client/Server (AP.2.1.2.1), Client/Server Communication (AP.2.2.1)
Quality measures category	Maintainability (QM.3.1), Interoperability (QM.4.1), Portability (QM.4.2), Scalability (QM.4.3), Reusability (QM.4.4)

Computing reviews category	Distributed Systems (C.2.4), Object-Oriented Programming (D.1.5)

References and Information Sources

[Baker 94] Baker, S. "CORBA Implementation Issues." *IEEE Colloquium on Distributed Object Management Digest 1994 7* (January 1994): 24-25.

[Brando 96] Brando, T. "Comparing CORBA & DCE." *Object Magazine 6*, 1 (March 1996): 52-7.

[Brown 96] Brown, A. & Wallnau, K. "A Framework for Evaluating Software Technology." *IEEE Software 13*, 5 (September 1996): 39-49.

[CORBA 96] *The Common Object Request Broker: Architecture and Specification*, Version 2.0. Framingham, MA: Object Management Group, 1996. Also available [online] WWW <URL: http://www.omg.org> (1996).

[CORBANet 96] Distributed Software Technology Center Home Page [online]. Available WWW <URL: http://corbanet.dstc.edu.au> (1996).

[Deng 95] Deng, R.H., et al. "Integrating Security in CORBA-Based Object Architectures," 50-61. *Proceedings of the 1995 IEEE Symposium on Security and Privacy*. Oakland, CA, May 8-10, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Foody 96] Foody, M.A. "OLE and COM vs. CORBA." *UNIX Review 14*, 4. (April 1996): 43-45.

[Jell 95] Jell, T. & Stal, M. "Comparing, Contrasting, and Interweaving CORBA and OLE," 140-144. *Object Expo Europe 1995*. London, UK, September 25-29, 1995. Newdigate, UK: SIGS Conferences, 1995.

[Kain 94] Kain, J.B. "An Overview of OMG's CORBA," 131-134. *Proceedings of OBJECT EXPO `94*. New York, NY, June 6-10, 1994. New York, NY: SIGS Publications, 1994.

[Mowbray 93] Mowbray, T.J. & Brando, T. "Interoperability and CORBA-Based Open Systems." *Object Magazine 3*, 3 (September/ October 1993): 50-4.

[OMG 96] Object Management Group home page [online]. Available WWW <URL: http://www.omg.org> (1996).

[Roy 95] Roy, Mark & Ewald, Alan. "Distributed Object Interoperability." *Object Magazine 5*, 1 (March/April 1995): 18.

[Steinke 95] Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

[Tibbets 95] Tibbets, Fred. "CORBA: A Common Touch for Distributed Applications." *Data Comm Magazine 24*, 7 (May 1995): 71-75.

[Wallace 96] Wallnau, Kurt & Wallace, Evan. "A Situated Evaluation of the Object Management Group's (OMG) Object Management Architecture (OMA)," 168-178. *Proceedings of the OOPSLA'96*. San Jose, CA, October 6-10, 1996. New York, NY: ACM, 1996. Presentation available [online] FTP. <URL: ftp://ftp.sei.cmu.edu/pub/corba/OOPSLA/present> (1996).

[Watson 96] Watson, A. "The OMG After CORBA 2." *Object Magazine 6*, 1 (March 1996): 58-60.

Current Author/Maintainer

Kurt Wallnau, SEI

External Reviewers

Dave Carney, SEI
Ed Morris, SEI

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/corba_body.html
Last Modified: 24 July 2008

Glossary Term

Adaptability

the ease with which software satisfies differing system constraints and user needs [Evans 87].

References and Information Sources

[CORBA 96] *The Common Object Request Broker: Architecture and Specification*, Version 2.0. Framingham, MA: Object Management Group, 1996. Also available [online] WWW <URL: http://www.omg.org> (1996).

References and Information Sources

[CORBANet 96] Distributed Software Technology Center Home Page [online]. Available WWW
<URL: http://corbanet.dstc.edu.au> (1996).

References and Information Sources

[Wallace 96] Wallnau, Kurt & Wallace, Evan. "A Situated Evaluation of the Object Management Group's (OMG) Object Management Architecture (OMA)," 168-178. *Proceedings of the OOPSLA'96*. San Jose, CA, October 6-10, 1996. New York, NY: ACM, 1996. Presentation available [online] FTP. <URL: ftp://ftp.sei.cmu.edu/pub/corba/OOPSLA/present> (1996).

References and Information Sources

[Brown 96] Brown, A. & Wallnau, K. "A Framework for Evaluating Software Technology." *IEEE Software 13*, 5 (September 1996): 39-49.

References and Information Sources

[Mowbray 93] Mowbray, T.J. & Brando, T. "Interoperability and CORBA-Based Open Systems." *Object Magazine 3*, 3 (September/October 1993): 50-4.

References and Information Sources

[Deng 95] Deng, R.H., et al. "Integrating Security in CORBA-Based Object Architectures," 50-61. *Proceedings of the 1995 IEEE Symposium on Security and Privacy*. Oakland, CA, May 8-10, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

References and Information Sources

[Brooks 87] Brooks, F. P. Jr. "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer 20,* 4 (April 1987): 10-9.

References and Information Sources

[Brown 96a] Brown, Alan W. "Preface: Foundations for Component-Based Software Engineering," vii-x. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute*. Los Alamitos, CA: IEEE Computer Society Press, 1996.

References and Information Sources

[Brown 96b] Brown, Alan W. & Wallnau, Kurt C. "Engineering of Component-Based Systems," 7-15. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute*. Los Alamitos, CA: IEEE Computer Society Press, 1996.

Notes

[1] See the definition of NDI in COTS and Open Systems - An Overview.

References and Information Sources

[Clements 95] Clements, Paul C. "From Subroutines to Subsystems: Component-Based Software Development," 3-6. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute*. Los Alamitos, CA: IEEE Computer Society Press, 1996.

Component-Based Software Development / COTS Integration

Software Technology Roadmap

Status

Advanced

Note

We recommend COTS and Open Systems--An Overview as prerequisite reading for this technology description.

Purpose and Origin

Component-based software development (CBSD) focuses on building large software systems by integrating previously-existing software components. By enhancing the *flexibility* and *maintainability* of systems, this approach can potentially be used to reduce software development costs, assemble systems rapidly, and reduce the spiraling maintenance burden associated with the support and upgrade of large systems. At the foundation of this approach is the assumption that certain parts of large software systems reappear with sufficient regularity that common parts should be written once, rather than many times, and that common systems should be assembled through reuse rather than rewritten over and over. CBSD embodies the "buy, don't build" philosophy espoused by Fred Brooks [Brooks 87]. CBSD is also referred to as component-based software engineering (CBSE) [Brown 96a, Brown 96b].

Component-based systems encompass both commercial-off-the-shelf (COTS) products and components acquired through other means, such as nondevelopmental items (NDIs).[1] Developing component-based systems is becoming feasible due to the following:

- the increase in the quality and variety of COTS products
- economic pressures to reduce system development and maintenance costs
- the emergence of component integration technology (see Object Request Broker)
- the increasing amount of existing software in organizations that can be reused in new systems

CBSD shifts the development emphasis from programming software to composing software systems [Clements 95].

Technical Detail

In CBSD, the notion of building a system by writing code has been replaced with building a system by assembling and integrating existing software components. In contrast to traditional development, where system integration is often the tail end of an implementation effort, component integration is the centerpiece of the approach; thus, implementation has given way to integration as the focus of system construction. Because of this, integrability is a key consideration in the decision whether to acquire, reuse, or build the components.

As shown in Figure 4, four major activities characterize the component-based development approach; these have been adapted from Brown [Brown 96b]:

- component qualification (sometimes referred to as suitability testing)
- component adaptation
- assembling components into systems
- system evolution

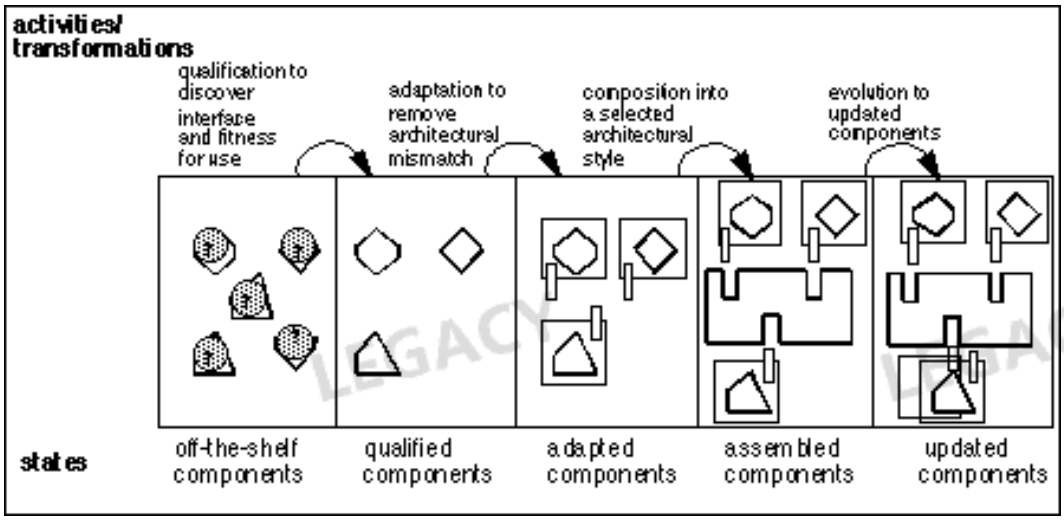


Figure 4: Activities of the Component-Based Development Approach

Each activity is discussed in more detail in the following paragraphs.

Component qualification. Component qualification is a process of determining "fitness for use" of previously-developed components that are being applied in a new system context. Component qualification is also a process for selecting components when a marketplace of competing products exists. Qualification of a component can also extend to include qualification of the development process used to create and maintain it (for example, ensuring algorithms have been validated, and that rigorous code inspections have taken place). This is most obvious in safety-critical applications, but can also reduce some of the attraction

of using preexisting components.

There are two phases of component qualification: *discovery* and *evaluation*. In the discovery phase, the properties of a component are identified. Such priorities include component functionality (what services are provided) and other aspects of a component's interface (such as the use of standards). These properties also include quality aspects that are more difficult to isolate, such as component reliability, predictability, and usability. In some circumstances, it is also reasonable to discover "non-technical" component properties, such as the vendor's market share, past business performance, and process maturity of the component developer's organization. Discovery is a difficult and ill-defined process, with much of the needed information being difficult to quantify and, in some cases, difficult to obtain.

There are some relatively mature evaluation techniques for selecting from among a group of peer products. For example, the International Standards Organization (ISO) describes general criteria for product evaluation [ISO 91] while others describe techniques that take into account the needs of particular application domains [IEEE 93, Poston 92]. These evaluation approaches typically involve a combination of paper-based studies of the components, discussion with other users of those components, and hands-on benchmarking and prototyping.

One recent trend is toward a "product line" approach that is based on a reusable set of components that appear in a range of software products. This approach assumes that similar systems (e.g., most radar systems) have a similar software architecture and that a majority of the required functionality is the same from one product to the next. (See Domain Engineering and Domain Analysis for further details on techniques to help determine similarity). The common functionality can therefore be provided by the same set of components, thus simplifying the development and maintenance life cycle. Results of implementing this approach can be seen in two different efforts [Lettes 96, STARSSCAI 95].

Component adaptation. Because individual components are written to meet different requirements, and are based on differing assumptions about their context, components often must be adapted when used in a new system. Components must be adapted based on rules that ensure conflicts among components are minimized. The degree to which a component's internal structure is accessible suggests different approaches to adaptation [Valetto 95]:

- *white box,* where access to source code allows a component to be significantly rewritten to operate with other components
- *grey box,* where source code of a component is not modified but the component provides its own extension language or application programming interface (API) (see Application Programming Interface)
- *black box,* where only a binary executable form of the component is available and there is no extension language or API

Each of these adaptation approaches has its own positives and negatives;

however, white box approaches, because they modify source code, can result in serious maintenance and evolution concerns in the long term. Wrapping, bridging, and mediating are specific programming techniques used to adapt grey- and black-box components.

Assembling components into systems. Components must be integrated through some well-defined infrastructure. This infrastructure provides the binding that forms a system from the disparate components. For example, in developing systems from COTS components, several architectural styles are possible:

- *database*, in which centralized control of all operational data is the key to all information sharing among components in the system
- *blackboard*, in which data sharing among components is opportunistic, involving reduced levels of system overhead
- *message bus*, in which components have separate data stores coordinated through messages announcing changes among components
- *object request broker (ORB) mediated*, in which the ORB technology (see Object Request Broker) provides mechanisms for language-independent interface definition and object location and activation

Each style has its own particular strengths and weaknesses. Currently, most active research and product development is taking place in object request brokers (ORBs) conforming to the Common Object Request Broker Architecture (CORBA).[2]

System evolution. At first glance, component-based systems may seem relatively easy to evolve and upgrade since components are the unit of change. To repair an error, an updated component is swapped for its defective equivalent, treating components as plug-replaceable units. Similarly, when additional functionality is required, it is embodied in a new component that is added to the system.

However, this is a highly simplistic (and optimistic) view of system evolution. Replacement of one component with another is often a time-consuming and arduous task since the new component will never be identical to its predecessor and must be thoroughly tested, both in isolation and in combination with the rest of the system. Wrappers must typically be rewritten, and side-effects from changes must be found and assessed. One possible approach to remedying this problem is Simplex (see Simplex Architecture).

Usage Considerations

Several items need to be considered when implementing component-based systems:

Short-term considerations

- *Development process*. An organization's software development process and philosophy may need to change. System integration can no longer be at the end of the implementation phase, but must be planned early and be continually managed throughout the development process. It is also recommended that as tradeoffs are being made among components during the development process, the rationale used in making the tradeoff decisions should be recorded and then evaluated in the final product [Brown 96b].
- *Planning*. Many of the problems encountered when integrating COTS components cannot be determined before integration begins. Thus, estimating development schedules and resource requirements is extremely difficult [Vigder 96].
- *Requirements*. When using a preexisting component, the component has been written to a preexisting, and possibly unknown, set of requirements. In the best case, these requirements will be very general, and the system to be built will have requirements that either conform or can be made to conform to the preexisting general requirements. In the worst case, the component will have been written to requirements that conflict in some critical manner with those of the new system, and the system designer must choose whether using the existing component is viable at all.
- *Architecture*. The selection of standards and components needs to have a sound architectural foundation, as this becomes the foundation for system evolution. This is especially important when migrating from a legacy system to a component-based system.
- *Standards*. If an organization chooses to use the component-based system development approach and it also has the goal of making a system open, then interface standards need to come into play as criteria for component qualification. The degree to which a software component meets certain standards can greatly influence the interoperability and portability of a system. Reference the COTS and Open Systems--An Overview description for further discussion.
- *Reuse of existing components*. Component-based system development spotlights reusable components. However, even though organizations have increasing amounts of existing software that can be reused, most often some amount of reengineering must be accomplished on those components before they can be adapted to new systems.
- *Component qualification.* While there are several efforts focusing on component qualification, there is little agreement on which quality attributes or measures of a component are critical to its use in a component-based system. A useful work that begins to address this issue is "SAAM: A Method for Analyzing the Properties of Software Architecture" [Abowd 94]. Another technique addresses the complexity of component selection and provides a decision framework that supports multi-variable component selection analysis [Kontio 96]. Other approaches, such as the qualification process defined by the US Air Force PRISM program, emphasize "fitness for use" within specific application domains, as well as the primacy of integrability of components [PRISM 96]. Another effort is Product Line Asset Support [PLAS 96].

Long-term considerations

- *External dependencies/vendor-driven upgrade problem.* An organization loses a certain amount of autonomy and acquires additional dependencies when integrating COTS components. COTS component producers frequently upgrade their components based on error reports, perceived market needs and competition, and product aesthetics. DoD systems typically change at a much slower rate and have very long lifetimes. An organization must juggle its new functionality requirements to accommodate the direction in which a COTS product may be going. New component releases require a decision from the component-based system developer/integrator on whether to include the new component in the system. To answer "yes" implies facing an undetermined amount of rewriting of wrapper code and system testing. To answer "no" implies relying on older versions of components that may be behind the current state-of-the-art and may not be adequately supported by the COTS supplier. This is why the component-based system approach is sometimes considered a risk transfer and not a risk reduction approach.
- *System evolution/technology insertion.* System evolution is not a simple plug-and-play approach. Replacing one component often has rippling affects throughout the system, especially when many of the components in the system are black box components; the system's integrator does not know the details of how a component is built or will react in an interdependent environment. Further complicating the situation is that new versions of a component often require enhanced versions of other components, or in some cases may be incompatible with existing components.

  Over the long-term life of a system, additional challenges arise, including inserting COTS components that correspond to new functionality (for example, changing to a completely new communications approach) and "consolidation engineering" wherein several components may be replaced by one "integrated" component. In such situations, maintaining external interface compatibility is very important, but internal data flows that previously existed must also be analyzed to determine if they are still needed.

Maturity

To date, the commercial components available and reliable enough for operational systems, and whose interfaces are well-enough understood, have primarily been operating systems, databases, email and messaging systems, office automation software (e.g., calendars, word processors, spreadsheets), and Graphical User Interface Builders. The number of available components continues to grow and quality and applicability continue to improve. As such, most successful applications have been in the AIS/MIS and C3I areas, with rather limited success in applications having real-time performance, safety, and security requirements. Indeed, in spite of the possible savings, using COTS components to build safety-critical systems where reliability, availability, predictability, and security are essential is frequently too risky [Brown 96b]. An organization will typically not have complete understanding or control of the

COTS components and their development.

Examples of apparently successful integration of COTS into operational systems include the following

- Deep Space Network Program at the NASA Jet Propulsion Laboratory [NASA 96a]
- Lewis Mission at NASA's Goddard Space Center [NASA 96b]
- Boeing's new 777 aircraft with 4 million lines of COTS software [Vidger 96]
- Air Force Space and Missile System Center's telemetry, tracking, and control (TT&C) system called the Center for Research Support (CERES) [Monfort 96]

In addition to the increasing availability of components applicable to certain domains, understanding of the issues and technologies required to expand CBSD practice is also growing, although significant work remains. Various new technical developments and products, including Common Object Request Broker Architecture and Component Object Model (COM), DCOM, and Related Capabilities [Vidger 96] and changes in acquisition and business practices should further stimulate the move to CBSD.

Costs and Limitations

It is widely assumed that the component-based software development approach, particularly in the sense of using COTS components, will be significantly less costly (i.e., shorter development cycles and lower development costs) than the traditional method of building systems "from scratch." In the case of using such components as databases and operating systems, this is almost certainly true. However, there is little data available concerning the relative costs of using the component-based approach and, as indicated in Usage Considerations, there are a number of new issues that must be considered.

In addition, if integrating COTS components, an additional system development and maintenance cost will be to negotiate, manage, and track licenses to ensure uninterrupted operation of the system. For example, a license expiring in the middle of a mission might have disastrous consequences.

Dependencies

Adapting preexisting components to a system requires techniques such as Application Programming Interface, wrapping, bridging, or mediating, as well as an increased understanding of architectural interactions and components' properties.

Alternatives

The alternatives include using preexisting components or creating the entire system as a new item.

Complementary Technologies

The advantages of using the CBSD/COTS integration approach can be greatly enhanced by coupling the approach with open systems (see COTS and Open Systems--An Overview).

Domain Engineering and Domain Analysis aid in identifying common functions and data among a domain of systems which in turn identifies possible reusable components.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Component-Based Software Development/ COTS Integration
Application category	System Allocation (AP.1.2.1), Select or Develop Algorithms (AP.1.3.4), Plan and Perform Integration (AP.1.4.4), Reengineering (AP.1.9.5)
Quality measures category	Maintainability (QM.3.1)
Computing reviews category	Software Engineering Design (D.2.10), Software Engineering Miscellaneous (D.2.m)

References and Information Sources

[Abowd 94] Abowd, G., et al. "SAAM: A Method for Analyzing the Properties of Software Architecture," 81-90. *Proceedings of the 16th International Conference on Software Engineering.* Sorrento, Italy, May 16-21, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[Brooks 87] Brooks, F. P. Jr. "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer 20*, 4 (April 1987): 10-9.

[Brown 96a] Brown, Alan W. "Preface: Foundations for Component-Based Software Engineering," vii-x. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute.* Los Alamitos, CA: IEEE Computer Society Press, 1996.

[Brown 96b] Brown, Alan W. & Wallnau, Kurt C. "Engineering of Component-Based Systems," 7-15. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute.* Los Alamitos, CA: IEEE Computer Society Press, 1996.

[Clements 95] Clements, Paul C. "From Subroutines to Subsystems: Component-Based Software Development," 3-6. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute.* Los Alamitos, CA: IEEE Computer Society Press, 1996.

[IEEE 93] *IEEE Recommended Practice on the Selection and Evaluation of CASE Tools* (IEEE Std. 1209-1992). New York, NY: Institute of Electrical and Electronics Engineers, 1993.

[ISO 91] *Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use*. Geneve, Switzerland: International Standards Organization/International Electrochemical Commission, 1991.

[Kontio 96] Kontio, J. "A Case Study in Applying a Systematic Method for COTS Selection," 201-209. *Proceedings of the 18th International Conference on Software Engineering.* Berlin, Germany, March 25-30, 1996. Los Alamitos, CA: IEEE Computer Society Press, 1996.

[Lettes 96] Lettes, Judith A. & Wilson, John. *Army STARS Demonstration Project Experience Report* (STARS-VC-A011/003/02). Manassas, VA: Loral Defense Systems-East, 1996.

[Monfort 96] Monfort, Lt. Col. Ralph D. "Lessons Learned in the Development and Integration of a COTS-Based Satellite TT&C System." *33rd Space Congress*. Cocoa Beach, FL, April 23-26, 1996.

[NASA 96a] *COTS Based Development* [online]. Available WWW <URL: http://www-isds.jpl.nasa.gov/cwo/cwo_23/handbook/Dsnswdhb.htm> (1996).

[NASA 96b] *Create Mechanisms/Incentives for Reuse and COTS Use* [online]. Available WWW <URL: http://bolero.gsfc.nasa.gov/c600/workshops/sswssp4b.htm> (1996).

[PLAS 96] PLAS [online]. Available WWW <URL: http://www.cards.com/plas> (1996).

[Poston 92] Poston R.M. & Sexton M.P. "Evaluating and Selecting Testing Tools." *IEEE Software 9,* 3 (May 1992): 33-42.

[PRISM 96] Portable, Reusable, Integrated Software Modules (PRISM) Program [online]. Available WWW <URL: http://www.cards.com/PRISM/prism_ov.html>(1996).

[STARSSCAI 95] Air Force/STARS Demonstration Project Home Page [online]. Available WWW <URL: http://www.asset.com/stars/afdemo/home.html> (1995).

[Thomas 92] Thomas, I. & Nejmeh. B. "Definitions of Tool Integration for Environments." *IEEE Software 9,* 3 (March 1992): 29-35.

[Valetto 95] Valetto, G. & Kaiser, G.E. "Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments," 40-48. *Proceedings of 7th IEEE International Workshop on CASE.* Toronto, Ontario, Canada, July 10-14, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Vidger 96] Vidger, M.R.; Gentleman, W.M.; & Dean, J. *COTS Software Integration: State-of-the-Art* [online]. Available WWW <URL: http://wwwsel.iit.nrc.ca/abstracts/NRC39198.abs> (1996).

Current Author/Maintainer

Capt Gary Haines, AFMC SSSG
David Carney, SEI
John Foreman, SEI

External Reviewers

Paul Kogut, Lockheed Martin, Paoli, PA
Ed Morris, SEI
Tricia Oberndorf, SEI
Kurt Wallnau, SEI

Modifications

7 Oct 97: minor edits
20 Jun 97: updated URL for [NASA 96a]
10 Jan 97 (original)

Footnotes

[1] See the definition of NDI in COTS and Open Systems - An Overview.

[2] From Wallnau, K. & Wallace, E. *A Robust Evaluation of the Object Management Architecture: A Focused Case Study in Legacy Systems Migration.* Submitted for publication to OOPLSA'96.

URL: http://www.sei.cmu.edu/str/descriptions/cbsd_body.html
Last Modified: 24 July 2008

References and Information Sources

[ISO 91] *Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use*. Geneve, Switzerland: International Standards Organization/ International Electrochemical Commission, 1991.

References and Information Sources

[IEEE 93] *IEEE Recommended Practice on the Selection and Evaluation of CASE Tools* (IEEE Std. 1209-1992). New York, NY: Institute of Electrical and Electronics Engineers, 1993.

References and Information Sources

[Poston 92] Poston R.M. & Sexton M.P. "Evaluating and Selecting Testing Tools." *IEEE Software 9*, 3 (May 1992): 33-42.

References and Information Sources

[Lettes 96] Lettes, Judith A. & Wilson, John. *Army STARS Demonstration Project Experience Report* (STARS-VC-A011/003/02). Manassas, VA: Loral Defense Systems-East, 1996.

References and Information Sources

[STARSSCAI 95] Air Force/STARS Demonstration Project Home Page [online]. Available WWW <URL: http://www.asset.com/stars/afdemo/home.html> (1995).

References and Information Sources

[Valetto 95] Valetto, G. & Kaiser, G.E. "Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments," 40-48. *Proceedings of 7th IEEE International Workshop on CASE*. Toronto, Ontario, Canada, July 10-14, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

Notes

[2] From Wallnau, K. & Wallace, E. *A Robust Evaluation of the Object Management Architecture: A Focused Case Study in Legacy Systems Migration.* Submitted for publication to OOPLSA'96.

References and Information Sources

[Vidger 96] Vidger, M.R.; Gentleman, W.M.; & Dean, J. *COTS Software Integration: State-of-the-Art* [online]. Available WWW <URL: http://wwwsel.iit.nrc.ca/abstracts/NRC39198.abs> (1996).

References and Information Sources

[Abowd 94] Abowd, G., et al. "SAAM: A Method for Analyzing the Properties of Software Architecture," 81-90. *Proceedings of the 16th International Conference on Software Engineering.* Sorrento, Italy, May 16-21, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

References and Information Sources

[Kontio 96] Kontio, J. "A Case Study in Applying a Systematic Method for COTS Selection," 201-209. *Proceedings of the 18th International Conference on Software Engineering.* Berlin, Germany, March 25-30, 1996. Los Alamitos, CA: IEEE Computer Society Press, 1996.

References and Information Sources

[PRISM 96] Portable, Reusable, Integrated Software Modules (PRISM) Program [online]. Available WWW <URL: http://www.cards.com/PRISM/prism_ov.html>(1996).

References and Information Sources

[PLAS 96] PLAS [online]. Available WWW <URL: http://www.cards.com/plas> (1996).

References and Information Sources

[NASA 96a] *COTS Based Development* [online]. Available WWW <URL: http://www-isds.jpl.nasa.gov/isds/Cwo's%20bak/CWO_23/PBD.HTM> (1996).

References and Information Sources

[NASA 96b] *Create Mechanisms/Incentives for Reuse and COTS Use* [online]. Available WWW <URL: http://bolero.gsfc.nasa.gov/c600/workshops/sswssp4b.htm> (1996).

References and Information Sources

[Monfort 96] Monfort, Lt. Col. Ralph D. "Lessons Learned in the Development and Integration of a COTS-Based Satellite TT&C System." *33rd Space Congress*. Cocoa Beach, FL, April 23-26, 1996.

Related Topics

System Allocation (AP.1.2.1)

- Component-Based Software Development/ COTS Integration

References and Information Sources

[COM 95] Microsoft Corporation. *The Component Object Model Specification*, Version 0.9, October 24, 1995 [online]. Available WWW <URL: http://www.microsoft.com/Com/resources/comdocs.asp> (1995).

References and Information Sources

[DCOM 97] Microsoft Corporation. *Distributed Component Object Model Protocol*-DCOM/1.0, draft, November 1996 [online]. Available WWW <URL: http://www.microsoft.com/Com/resources/comdocs.asp> (1996).

References and Information Sources

[Brockschmidt 95]	Brockschmidt, Kraig. Inside OLE, 2nd edition, Microsoft Press, 1995

References and Information Sources

[Active 97] Active Group home page [online]. Available WWW <URL: http://www.activex.org/> (1997).

References and Information Sources

[Harmon 99] Harmon, Paul. *Microsoft transaction Server.* Component development Strategies Vol IX No 3. Available WWW <URL: http://www.cutter.com/cds/1999toc.htm#mar > 1999

Component Object Model (COM), DCOM, and Related Capabilities

Software Technology Roadmap

Status

Advanced

Note

We recommend Object Request Broker, Remote Procedure Call, *and* Component-Based Software Development/COTS Integration, *as prerequisite readings for this technology description.*

Purpose and Origin

COM [COM 95] refers to both a specification and implementation developed by Microsoft Corporation which provides a framework for integrating components. This framework supports *interoperability* and *reusability* of distributed objects by allowing developers to build systems by assembling reusable components from different vendors which communicate via COM. By applying COM to build systems of preexisting components, developers hope to reap benefits of *maintainability* and *adaptability*.

COM defines an application programming interface (API) to allow for the creation of components for use in integrating custom applications or to allow diverse components to interact. However, in order to interact, components must adhere to a binary structure specified by Microsoft. As long as components adhere to this binary structure, components written in different languages can interoperate.

Distributed COM [DCOM 97] is an extension to COM that allows network-based component interaction. While COM processes can run on the same machine but in different address spaces, the DCOM extension allows processes to be spread across a network. With DCOM, components operating on a variety of platforms can interact, as long as DCOM is available within the environment.

It is best to consider COM and DCOM as a single technology that provides a range of services for component interaction, from services promoting component integration on a single platform, to component interaction across heterogeneous networks. In fact, COM and its DCOM extensions are merged into a single runtime. This single runtime provides both local and remote access.

While COM and DCOM represent "low-level" technology that allows components to interact, OLE [Brockschmidt 95], ActiveX [Active 97] and MTS [Harmon 99] represent higher-level application services that are built on top of COM and DCOM. OLE builds on COM to provide services such as object "linking" and "embedding" that are used in the creation of compound documents (documents generated from multiple tool sources). ActiveX extends the basic capabilities to allow components to be embedded in Web sites. MTS expands COM capabilities with enterprise services such as transaction and security to allow Enterprise Information Systems (EIS) to be built using COM components. COM+ is the evolution of COM.

COM+ integrates MTS services and message queuing into COM, and makes COM programming easier through a closer integration with Microsoft languages as Visual Basic, Visual C++, and J++. COM+ will not only add MTS-like quality of service into every COM+ object, but it will hide some of the complexities in COM coding.

The distinctions among various Microsoft technologies and products are sometimes blurred. Thus, one might read about "OLE technologies" which encompass COM, or "Active Platform" as a full web solution. In this technology description, we focus on the underlying technology represented by COM, DCOM, and COM+.

Technical Detail

COM is a binary compatibility specification and associated implementation that allows clients to invoke services provided by COM-compliant components (COM objects). As shown in Figure 5, services implemented by COM objects are exposed through a set of interfaces that represent the only point of contact between clients and the object.

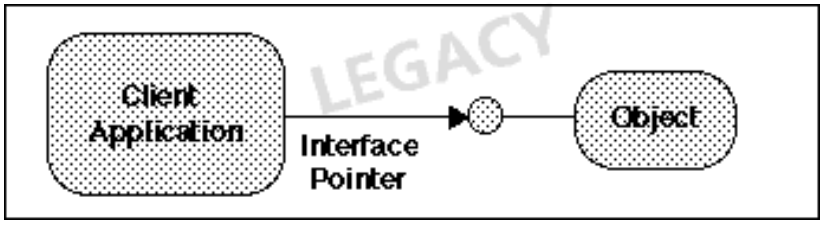


Figure 5: Client Using COM Object Through an Interface Pointer [COM 95]

COM defines a binary structure for the interface between the client and the object. This binary structure provides the basis for interoperability between software components written in arbitrary languages. As long as a compiler can reduce language structures down to this binary representation, the implementation language for clients and COM objects does not matter - the point of contact is the run-time binary representation. Thus, COM objects and clients can be coded in any language that supports Microsoft's COM binary structure.

A COM object can support any number of interfaces. An interface provides a grouped collection of related methods. For example, Figure 6 depicts a COM

object that emulates a clock. IClock, IAlarm and ITimer are the interfaces of the clock object. The IClock interface can provide the appropriate methods (not shown) to allow setting and reading the current time. The IAlarm and ITimer interfaces can supply alarm and stopwatch methods.

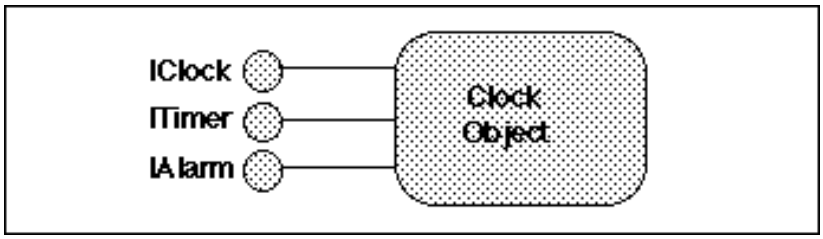


Figure 6: Clock COM object

COM objects and interfaces are specified using Microsoft Interface Definition Language (IDL), an extension of the DCE Interface Definition Language standard (see Distributed Computing Environment). To avoid name collisions, each object and interface must have a unique identifier.

Interfaces are considered logically immutable. Once an interface is defined, it should not be changed-new methods should not be added and existing methods should not be modified. This restriction on the interfaces is not enforced, but it is a rule that component developers should follow. Adhering to this restriction removes the potential for version incompatibility-if an interface never changes, then clients depending on the interface can rely on a consistent set of services. If new functionality has to be added to a component, it can be exposed through a different interface. For our clock example, we can design an enhanced clock COM object supporting the IClock2 interface that inherits from IClock. IClock2 may expose new functionality.

Every COM object runs inside of a server. A single server can support multiple COM objects. As shown in Figure 7, there are three ways in which a client can access COM objects provided by a server:

1. In-process server: The client can link directly to a library containing the server. The client and server execute in the same process. Communication is accomplished through function calls.
2. Local Object Proxy: The client can access a server running in a different process but on the same machine through an inter-process communication mechanism. This mechanism is actually a lightweight Remote Procedure Call (RPC).
3. Remote Object Proxy: The client can access a remote server running on another machine. The network communication between client and server is accomplished through DCE RPC. The mechanism supporting access to remote servers is called DCOM.

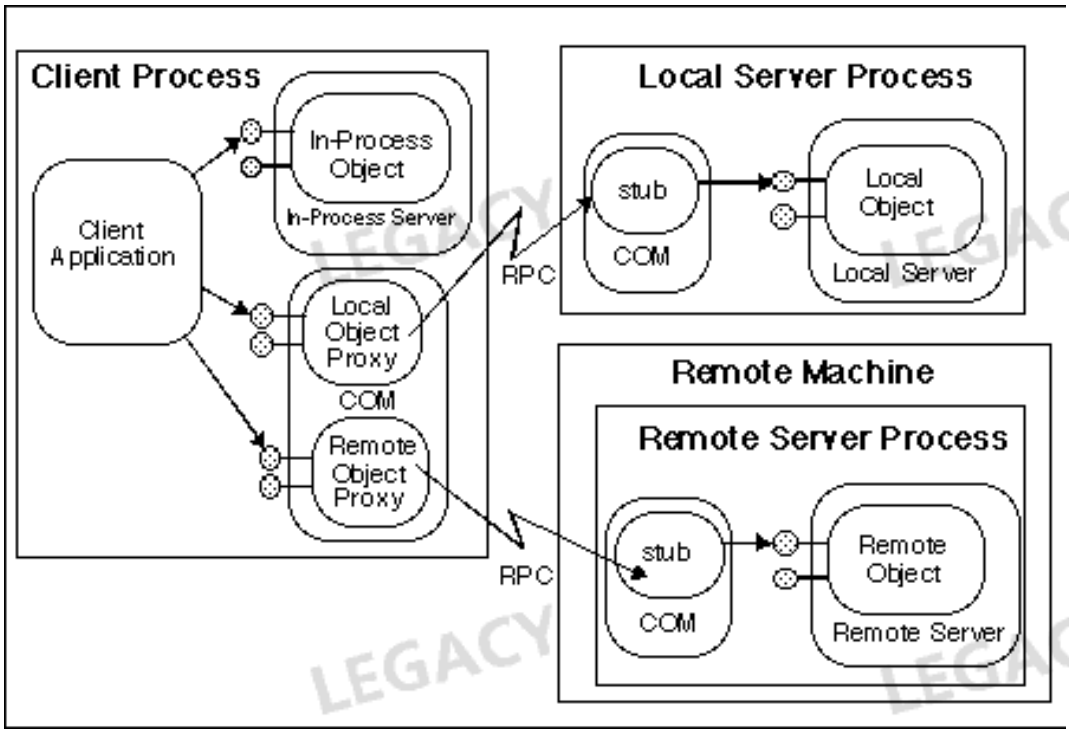


Figure 7: Three Methods for Accessing COM Objects [COM 95]

If the client and server are in the same process, the sharing of data between the two is simple. However, when the server process is separate from the client process, as in a local server or remote server, COM must format and bundle the data in order to share it. This process of preparing the data is called marshalling. Marshalling is accomplished through a "proxy" object and a "stub" object that handle the cross-process communication details for any particular interface (depicted in Figure 8). COM creates the "stub" in the object's server process and has the stub manage the real interface pointer. COM then creates the "proxy" in the client's process, and connects it to the stub. The proxy then supplies the interface pointer to the client.

The client calls the interfaces of the server through the proxy, which marshals the parameters and passes them to the server stub. The stub unmarshals the parameters and makes the actual call inside the server object. When the call completes, the stub marshals return values and passes them to the proxy, which in turn returns them to the client. The same proxy/stub mechanism is used when the client and server are on different machines. However, the internal implementation of marshalling and unmarshalling differs depending on whether the client and server operate on the same machine (COM) or on different machines (DCOM). Given an IDL file, the Microsoft IDL compiler can create default proxy and stub code that performs all necessary marshalling and unmarshalling.

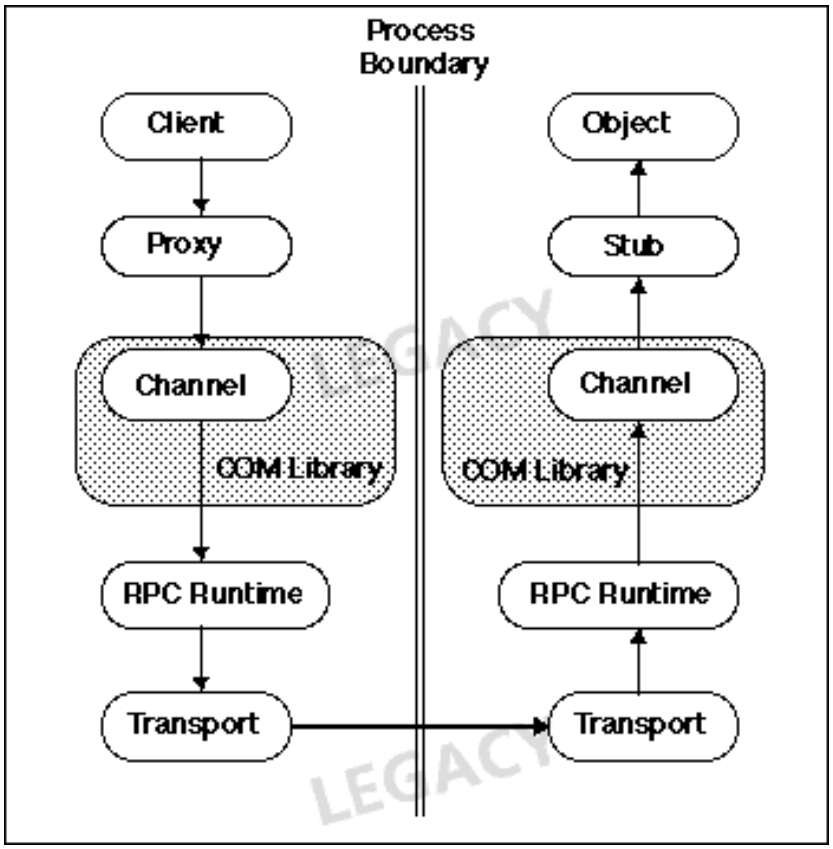


Figure 8: Cross-process communication in COM [COM 95]

All COM objects are registered with a component database. As shown in Figure 9, when a client wishes to create and use a COM object:

1. It invokes the COM API to instantiate a new COM object.
2. COM locates the object implementation and initiates a server process for the object.
3. The server process creates the object, and returns an interface pointer at the object.
4. The client can then interact with the newly instantiated COM object through the interface pointer.

An important aspect in COM is that objects have no identity, i.e. a client can ask for a COM object of some type, but not for a particular object. Every time that COM is asked for a COM object, a new instance is returned. The main advantage of this policy is that COM implementations can pool COM objects and return these pooled objects to requesting clients. Whenever a client has finished using an object the instance is returned to the pool. However, there are mechanisms to simulate identity in COM such as monikers (reviewed later).

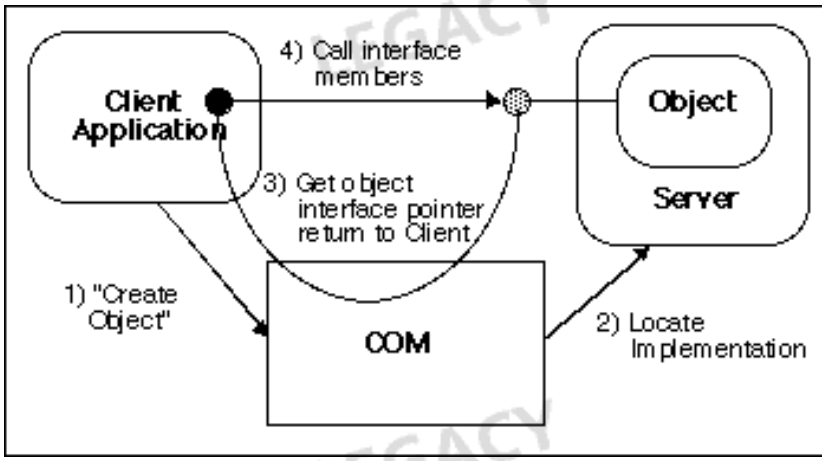


Figure 9: Creating a COM object pointer [COM 95]

COM includes interfaces and API functions that expose operating system services, as well as other mechanisms necessary for a distributed environment (naming, events, etc.). These are sometimes referred to as COM technologies (or services), and are shown in Table 3.

Table 3: COM Technologies

Service	Explanation
Type Information	Some clients need runtime access to type information about COM objects. This type information is generated by the Microsoft IDL compiler and is stored in a type library. COM provides interfaces to navigate the type library.
Structured Storage and Persistence	COM objects need a way to store their data when they are not running. The process of saving data for an object is called making an object persistent. COM supports object persistence through "Structured Storage", which creates an analog of a file system within a file. Individual COM objects can store data within the file, thus providing persistence.
Monikers	Clients often require a way to allow them to connect to the exact same object instance with the exact same state at a later point in time. This support is provided via "monikers". A moniker is a COM object that knows how to create and initialize the content of a single COM object instance. A moniker can be asked to bind to the COM object it represents, such as a COM object residing on specific machine on the network, or a group of cells inside a spreadsheet.

Uniform Data Transfer	COM objects often need to pass data amongst themselves. Uniform Data Transfer provides for data transfers and notifications of data changes between a source called the data object, and something that uses the data, called the consumer object.
Connectable Objects	Some objects require a way to notify clients that an event that has occurred. COM allows such objects to define outgoing interfaces to clients as well as incoming interfaces. The object defines an interface it would like to use (e.g., a notification interface) and the client implements the interface. This enables two-way communication between the client and the component.

COM has enjoyed great industrial support with thousands of ISVs developing COM components and applications. However, COM suffers from some weaknesses that have been recognized by Microsoft and addressed in Component Object Model+, which is the ongoing upgrade of COM.

1. COM is hard to use. Reference counting, Microsoft IDL, Global Unique Identifiers (GUID), etc. require deep knowledge of COM specification from developers.
2. COM is not robust enough for enterprise deployments. Services such as security, transactions, reliable communications, and load balancing are not integrated in COM.

Both issues were partially mitigated by add-ons of COM, complexity by integrated development environments and robustness by MTS. However, to further address those problems, the company is working to turn COM+ and the MTS (Microsoft Transaction Server) into one programming model that will simplifying the lives of developers building distributed, enterprise-wide COM applications. COM+ integrates seamlessly with all COM-aware languages (basically Microsoft languages). Users write components in their favorite language. The tool chosen and the COM+ runtime take care of turning these classes into COM components [Kirtland 97].

Usage Considerations

A number of issues must be evaluated when considering COM, DCOM, and COM+. They include

- *Platform support.* COM and DCOM are best supported on Windows 95 and NT platforms. However, Microsoft has released a version of COM/DCOM for MacOS that supports OLE-style compound documents and the creation of ActiveX controls. Software AG, a Microsoft partner, has

released DCOM for some UNIX operating systems, concretely OS/390, HP-UX 11.0, SUN Solaris, AIX 4.2, 4.3, Tru64 Unix 4.0 and Linux. However, DCOM over non-Windows platforms has few supporters. Until DCOM for alternate platforms has solidified, the technology is best applied in environments that are primarily Windows-based.

- *Platform specificity of COM/DCOM components.* Because COM and DCOM are based on a native binary format, components written to these specifications are not platform independent. Thus, either they must be recompiled for a specific platform, or an interpreter for the binary format must become available. Depending on your perspective, the use of a binary format may be either an advantage (faster execution, better use of native platform capabilities) or a disadvantage (ActiveX controls, unlike Java applets, are NOT machine independent). See Java for more information.
- *Security.* Because COM/DCOM components have access to a version of the Microsoft Windows API, "bad actors" can potentially damage the user's computing environment. In order to address this problem, Microsoft employs "Authenticode" [Microsoft 96] which uses public key encryption to digitally sign components. Independent certification authorities such as VeriSign issue digital certificates to verify the identity of the source of the component [VeriSign 97]. However, even certified code can contain instructions that accidentally, or even maliciously, compromise the user's environment.
- *Support for distributed objects.* COM/DCOM provides basic support for distributed objects. There is currently no support for situations requiring real time processing, high reliability, or other such specialized component interaction.
- *Stability of APIs.* In October of 1996 Microsoft turned over COM/DCOM, parts of OLE, and ActiveX to the Open Group (a merger of Open Software Foundation and X/Open). The Open Group has formed the Active Group to oversee the transformation of the technology into an open standard. The aim of the Active Group is to promote the technology's compatibility across systems (Windows, UNIX, and MacOS) and to oversee future extension by creating working groups dedicated to specific functions. However, it is unclear how much control Microsoft will relinquish over the direction of the technology. Certainly, as the inventor and primary advocate of COM and DCOM, Microsoft is expected to have strong influence on the overall direction of the technology and underlying APIs.
- *Long-term system maintainability.* Microsoft is actively supporting COM and DCOM technology and pushing it in distributed and Web-based directions. Microsoft is also trying to preserve existing investments in COM technology while introducing incremental changes. Microsoft, for example, has ensured backward compatibility of COM+. Although this affirmation is in general true, COM objects that access local information in the registry or in system folders may require modification. In general, the PC community has not been faced with the concern of very long-lived systems, and vendors often provide support only for recent releases.

Maturity

COM has its roots in OLE version 1, which was created in 1991 and was a

proprietary document integration and management framework for the Microsoft Office suite. Microsoft later realized that document integration is just a special case of component integration. OLE version 2, released in 1995 was a major enhancement over its predecessor. The foundation of OLE version 2, now called COM, provided a general-purpose mechanism for component integration on Windows platforms [Brockschmidt 95]. While this early version of COM included some notions of distributed components, more complete support for distribution became available with the DCOM specifications and implementations for Windows95 and Windows NT released in 1996. Beta versions of DCOM for Mac, Solaris and other operating systems followed shortly after.

There are many PC-based applications that take advantage of COM and DCOM technology. The basic approach has proven sound, and as previously mentioned, a large component industry has sprung up to take advantage of opportunities created by the Microsoft technology. On the other hand, DCOM has just arrived on non-Windows platforms, and there is little experience with it. DCOM for non-Windows platforms is mainly used to communicate COM based programs with legacy applications in Mainframes and Unix workstations.

COM+ is much younger than COM, it was announced in Sept. 23, 1997 and shipped with windows 2000 (a.k.a. Windows NT 5.0). COM+ can be considered the next release of COM. We are unaware of any large-scale distributed applications relying on COM+ support.

The computing paradigm for distributed applications is in flux, due to the relative immaturity of the technology and recent advances in web-based computing. The Web-centered computing industry has begun to align itself into two technology camps-with one camp centered around Microsoft's COM/DCOM/COM+, Internet Explorer, and ActiveX capabilities, and the other camp championing Netscape, CORBA, and Java/J2EE solutions. Both sides argue vociferously about the relative merits of their approach, but at this time there is no clear technology winner. Fortunately, both camps are working on mechanisms to support interplay between the technology bases. Thus, a COM/DCOM to CORBA mapping is supported by CORBA vendors [Foody 96], and Microsoft has incorporated Java into an Internet strategy. However, work on interconnection between the competing approaches is not complete, and each camp would shed few tears if the other side folded.

Costs and Limitations

Low cost development tools from Microsoft (such as Visual C++ or Visual Basic), as well as tools from other vendors provide the ability to build and access COM components for Windows platforms. Construction of clients and servers is straightforward on these platforms. In addition, the initial purchase price for COM and DCOM is low on Windows platforms. For other platforms the prices are considerably more expensive. DCOM for mainframes, for example, costs around two hundred thousand dollars by December 1999.

Beyond basic costs to procure the technology, any serious software development using COM/DCOM/COM+ requires substantial programmer expertise-the complexities of building distributed applications are not eliminated.

It would be a serious mistake to assume that the advent of distributed object technologies like COM/DCOM/COM+ reduces the need for expertise in areas like distributed systems design, multi-threaded applications, and networking.

However, Microsoft has a strong support organization to assist individuals developing COM/DCOM clients and objects: many sample components, books and guides on the subject of COM/DCOM development are available. Unfortunately, information on COM+ is limited at this time.

Dependencies

Dependencies include Remote Procedure Call and Distributed Computing Environment.

Alternatives

COM/DCOM/COM+ represents one of a number of alternate technologies that support distributed computing. Some technologies, such as remote procedure call, offer "low level" distribution support. Other technologies, such as message oriented middleware and transaction processing monitors, offer distribution support paradigms outside the realm of objects. The Common Object Request Broker Architecture (CORBA) and Java 2 Enterprise Edition (J2EE) can be considered direct competitors to COM/DCOM. Information about technologies supporting distributed computing is available in the following places:

- Distributed Computing Environment
- Remote Procedure Call
- Message-Oriented Middleware
- Transaction Processing Monitor Technology
- Common Object Request Broker Architecture
- Two Tier Software Architectures
- Java

Complementary Technologies

One commonly hears of COM and DCOM in conjunction with OLE, ActiveX, MTS and COM+. Indeed, these and other technologies constitute Microsoft's distributed and web-oriented strategy. This strategy is globally referred as Distributed interNet Architecture(tm) (DNA) and it comprises a full set of products and specifications to implement net-centric applications.

Technologies championed by other vendors can also be used in conjunction with COM. For example, COM objects can be created and manipulated from Java code. Tools are provided to create Java classes from COM type library information-these classes can be included in Java code. Using Internet Explorer, Java programs can also expose functionality as COM services. In general, Microsoft's approach for Java support involves tying it very closely to its existing Internet strategy (Internet Explorer, COM/DCOM, ActiveX); i.e., to provide a mechanism for interfacing to the wide range of components that already adhere

to Microsoft's strategy and specifications.

COM+ is a good candidate to implement the middle layer of multitier architectures. The distribution support and quality of service provided by COM+ can help to overcome some of the complexities involved in these architectures.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Component Object Model (COM), DCOM, and Related Capabilities
Application category	Software Architecture Models (AP.2.1.1) Client/Server (AP.2.1.2.1) Client/Server Communications (AP.2.2.1)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1) Reusability (QM.4.4)
Computing reviews category	Distributed Systems (C.2.4) Object-Oriented Programming (D.1.5)

References and Information Sources

[Active 97] Active Group home page [online]. Available WWW <URL: http://www.activex.org/> (1997).

[Brockschmidt 95] Brockschmidt, Kraig. Inside OLE, 2nd edition, Microsoft Press, 1995

[Chappell 96] Chappell, David. *DCE and Objects* [online]. Available WWW <URL: http://www.opengroup.org/dce/info/dce_objects.htm> (1996).

[COM 95] Microsoft Corporation. *The Component Object Model Specification*, Version 0.9, October 24, 1995 [online]. Available WWW <URL: http://www.microsoft.com/com/resources/comdocs.asp>(1995).

[DCOM 97] Microsoft Corporation. *Distributed Component Object Model Protocol*-DCOM/1.0, draft, November 1996 [online]. Available WWW <URL: http://www.globecom.net/ietf/draft/draft-brown-dcom-v1-spec-03.html> (1996).

[Foody 96] Foody, M.A. "OLE and COM vs. CORBA." *UNIX Review 14*, 4. (April 1996): 43-45.

[Harmon 99] Harmon, Paul. *Microsoft transaction Server*. Component development Strategies Vol IX No 3. Available WWW <URL: http://www.cutter.com/cds/1999toc.htm#mar > 1999

[Kirtland 97] Kirtland, Mary. "The COM+ Programming Model Makes it Easy to Write Components in Any Language". Microsoft System Journal. Dec, 1997.

[Microsoft 96] Microsoft Corporation. *Microsoft Authenticode Technology* [online]. Available WWW <URL: http://www.microsoft.com/security/tech/misf8.htm> (1996).

[MSCOM 97] Microsoft home page [online]. The site provides information about COM, DCOM and OLE. Available WWW <URL: http://www.microsoft.com/> (1997).

[OMG 97] Object Management Group home page [online]. The site provides information comparing DCOM (ActiveX) to CORBA. Available WWW <URL: http://www.omg.org/> (1997).

[VeriSign 97] Verisign home page [online]. Available WWW <URL: http://www.verisign.com> (1997).

Current Author/Maintainer

Santiago Comella-Dorda, SEI

External Reviewers

Modifications

13 Mar 2001: Update with new developments of COM
23 June 1997: Total replacement text
10 Jan 1997: Original

URL: http://www.sei.cmu.edu/str/descriptions/com_body.html
Last Modified: 24 July 2008

References and Information Sources

[Kirtland 97] Kirtland, Mary. "The COM+ Programming Model Makes it Easy to Write Components in Any Language". Microsoft System Journal. Dec, 1997.

References and Information Sources

[Microsoft 96] Microsoft Corporation. *Microsoft Authenticode Technology* [online]. Available WWW <URL: http://www.microsoft.com/security/tech/misf8.htm> (1996).

References and Information Sources

[VeriSign 97] Verisign home page [online]. Available WWW <URL: http://www.verisign.com> (1997).

References and Information Sources

[Foody 96] Foody, M.A. "OLE and COM vs. CORBA." *UNIX Review 14*, 4. (April 1996): 43-45.

Computer System Security--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

C4I systems include networks of computers that provide real-time situation data for military decision makers and a means of directing response to a situation. These networks collect data from sensors and subordinate commands. That data is fused with the existing situation status data and presented by the C4I system to decision makers through display devices. C4I networks today may incorporate two general types of networks: networks of Multi-level Secure (MLS) Systems, and Intranets of single level systems. Figure 5 shows the relevant major security components of a C4I computer system network.

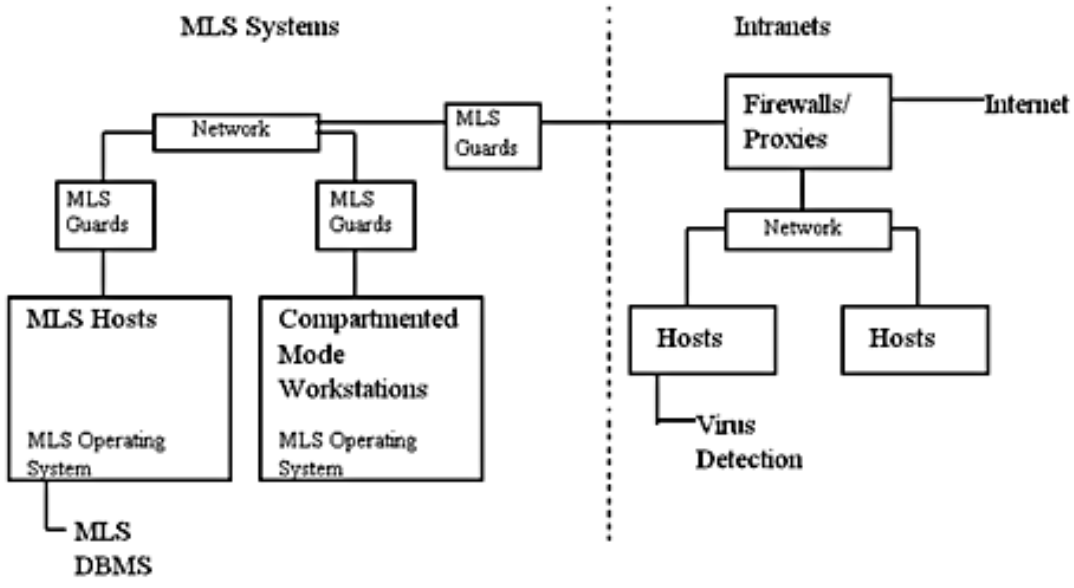


Figure 5: Computer System Security in C4I Systems

This technology description is tutorial in nature. It provides a general overview of key concepts and introduces key technologies. Detailed discussions of the

individual technologies can be found in the referenced technology descriptions.

Technical Detail

Some computers in the network are hosts that collect and process data. A host can be a mainframe, a server, a workstation, or a PC. It may perform the function of an application processor, a communication processor, a database processor, a display processor, or a combination. The security mode for the host may be single-level or multi-level. A single-level host processes all data as though it was one security level. A multi-level host can process data at different security levels, identify and isolate data in the appropriate levels or categories, and distribute data only to the appropriately cleared users.

C4I systems benefit from multi-level security implementations because C4I systems fuse data from sources with a wide range of security levels and provide status, warning data, or direction to war fighting systems that may be at lesser security levels. An MLS operating system (see Multi-Level Secure One Way Guard with Random Acknowledgment) provides the software that makes a host MLS. A particular kind of MLS host is the Compartmented Mode Workstation (CMW). A CMW is a MLS host that has been evaluated to satisfy the Defense Intelligence Agency CMW requirements [Woodward 87] in addition to the Trusted Computer System Evaluation Criteria [DoD 85]. A MLS host may use a MLS DBMS (see Multi-Level Secure Database Management Schemes) to store and retrieve data at multiple security levels. A MLS guard provides a secure interface across a security boundary between systems operating at different security levels or modes.

MLS guards may allow data across the interface automatically or may require manual review of data and approval of transfer on an attached terminal. They also may control data transfer across the interface in both directions or be limited to allowing data to be transferred one way, usually from the low security level side of a security boundary to the high security level side. One-way guards are usually the easiest to implement and accredit for use. Data integrity is an issue with one-way guards because an acknowledgment message can not be used. Recent research in one-way guards has addressed allowing an acknowledgment message (see Multi-Level Secure One Way Guard with Random Acknowledgment).

Intranets use the same kind of networking software (e.g., TCP/IP, Telnet, Netnews, DNS, browsers, home pages) that is used on the Internet, but Intranets use them on a private dedicated network. They are in essence a private Internet. They are used in a growing number of ways in many military and corporate networks including mission performance, off-line processing of raw data, administrative support, and mail networks. They may be incorporated into C4I systems using firewalls or proxies (see Firewalls and Proxies) and MLS guards. Firewalls or proxies may be used to provide a security interface to the Internet. If the Intranets are to be connected to MLS systems, they must be connected through MLS guards. In an environment with Intranet hosts, a major concern is Virus Detection and Intrusion Detection. PCs on a network are particularly susceptible to virus attacks from other hosts on the network or the

Internet. PCs are also vulnerable to viruses carried on floppy disks. Since PCs are now in most homes, transfer of files from home to work via floppy disk provides the risk of introducing a virus into the Intranet. PCs are more vulnerable to viruses than UNIX-based workstations or mainframes because the PC has no memory protection hardware and the operating system (DOS and Windows) allows a program to access any part of memory or disk.

Security across the networks in a C4I system is crucial. Traditionally this security is provided by physically protecting the equipment and cables in the network for localized networks. When that is not possible, the network connections are encrypted using encryption hardware in the communications paths. End-to-end encryption is an alternative that encrypts the data using software before it is put on the network and decrypts it after it has been taken off of the network. Then non-encrypted circuits can be used for communications.

Any encryption system involves the distribution of keys used by the encryption algorithm for the encryption/decryption of messages and data. Encryption keys must be replaced periodically to enhance security or when the key has been compromised or lost. Traditionally these keys have been distributed through couriers or encrypted circuits. Public key cryptography provides a means of electronic encryption key distribution that can lower the security risk and administrative workload associated with encryption.

Data integrity is another issue associated with the networks used in C4I systems. Public Key Digital Signatures and providing for Nonrepudiation in Network Communications are two means to enhance data integrity. Public key digital signatures, which make use of public key encryption and message authentication codes, are a means to authenticate that data came from the person identified as the sender and that the data has not been modified. The nonrepudiation process uses a digital signature and a trusted arbitrator process to assure that a particular message has been sent and received and to establish the time when this occurred.

Usage Considerations

MLS systems require specialized knowledge to build, accredit, and maintain. The cost of MLS systems can be high. The system development overhead and operational performance overhead associated with MLS systems are substantial. They are difficult to implement in an "open" configuration because open requirements sometimes conflict with MLS requirements. On the other hand, using MLS techniques may be the only allowable way to construct some C4I systems. Operational security vulnerabilities may be unacceptable without MLS implementations. Procedural security approaches may be too slow for an operational C4I system as a non-MLS approach. A single-level system approach may be too restrictive. For example, a secret single-level system that contains unclassified, confidential, and secret data will not release confidential data to a user who is cleared for confidential and needs the data. That is because the system cannot determine what data is confidential rather than secret. Further usage discussions are addressed in individual technology descriptions.

The National Security Agency (NSA) Multilevel Information Systems Security

Initiative (MISSI) is an evolutionary effort intended to provide better MLS capability in a cost-effective manner [MISSI 96]. This effort was initiated after the Gulf War when it was recognized that war fighting commanders needed MLS systems in order to incorporate intelligence and other highly classified data into their planning and operations in a timely manner. The MISSI effort is developing a set of building block products that can be obtained commercially to construct an MLS system. The initial products include the FORTEZZA crypto cards and associated FORTEZZA ready workstation applications to control access to and protect data on a workstation in a network environment. Other products include high-assurance guards and firewalls to provide access control and encryption services between the local security boundary and external networks. MISSI will also include secure computing products that provide high-trust operating systems and application programs for MLS hosts, and network encryption and security management products. These products can be incorporated into developing MLS systems as the products become available.

Maturity

See individual technologies.

Costs and Limitations

See individual technologies.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Computer System Security - an Overview
Application category	Information Security (AP.2.4)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating Systems Security & Protection (D.4.6), Security & Protection (K.6.5), Computer-Communications Networks Security and Protection (C.2.0)

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Woodward 87] Woodward, John. *Security Requirements for High and Compartmented Mode Workstations* (MTR 9992, DDS 2600-5502-87). Washington, DC: Defense Intelligence Agency, 1987.

[DoD 85] *Department of Defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC)* (DoD 5200.28-STD 1985). Fort Meade, MD: Department of Defense, 1985. Also available WWW <URL: http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html> (1985).

[MISSI 96] MISSI Web site [online]. Available WWW <URL: http://beta.missilab.com> (1996).

[Russel 91] Russel, Deborah & Gangemi, G.T. Sr. *Computer Security Basics.* Sebastopol, CA: O'Reilly & Associates, Inc., 1991.

[White 96] White, Gregory B.; Fisch, Eric A.; & Pooch, Udo W. *Computer System and Network Security.* Boca Raton, FL: CRC Press, 1996.

Current Author/Maintainer

Tom Mills, Lockheed Martin

External Reviewers

Brian Gallagher, SEI

Modifications

8 July 97: added reference to MLS One-Way Guard with Random Ack.
20 June 97: updated URL for [MISSI 96]
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/security_body.html
Last Modified: 24 July 2008

References and Information Sources

[Woodward 87] Woodward, John. *Security Requirements for High and Compartmented Mode Workstations* (MTR 9992, DDS 2600-5502-87). Washington, DC: Defense Intelligence Agency, 1987.

References and Information Sources

[MISSI 96] MISSI Web site [online]. Available WWW <URL: http://beta.missilab.com> (1996).

Related Topics

Information Security (AP.2.4)

- Computer System Security -- an Overview
- Electronic Encryption Key Distribution
- End-to-End Encryption
- Trusted Computing Base
- Virus Detection

References and Information Sources

[FAR 96] *Federal Acquisition Regulations.* Washington, DC: General Services Administration, 1996.

Notes

[1] "Property" in this definition explicitly excludes real property.

References and Information Sources

[Meyers 97] Meyers, Craig & Oberndorf, Tricia. *Open Systems: The Promises and the Pitfalls*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997.

Notes

[2] It should be noted that interface specifications are in general not sufficient to ensure full "plug-and-play" operation. In practice, the real interface between two components of a system consists of all the assumptions that each makes about the other. APIs, data formats, and protocols address a large number of these assumptions, but by no means all of them. It remains for further investigations to determine the full set of interface knowledge that must be standardized to ever get really close to an ideal "plug-and-play" system creation process.

Notes

[3] In June 1994 Secretary of Defense William Perry directed that DoD acquisitions should make maximum use of performance specifications and commercial standards. In November 1994 Undersecretary of Defense (Acquisition and Technology) Paul Kaminski directed "that `open systems' specifications and standards be used for acquisition of weapon systems electronics to the greatest extent practical."

References and Information Sources

[Carney 97a] Carney, D, & Oberndorf, P. "The Commandments of COTS: Still Searching for the Promised Land." *Crosstalk 10,* 5 (May 1997): 25-30. Also available online at <URL: http://www.sei.cmu.edu/cbs/SEI_refs.html> (Postscript) and <URL: http://www.stsc.hill.af.mil/CrossTalk/1997/may/commandments.html>.

References and Information Sources

[Carney 97b] Carney, D. *Assembling Large Systems from COTS Components: Opportunities, Cautions, and Complexities* [online]. Available WWW <URL: http://www.sei.cmu.edu/cbs/papers/paper13a.html>.

References and Information Sources

[IEWCS 96] *Open Systems Joint Task Force Case Study of U.S. Army Intelligence and Electronic Warfare Common Sensor* (IEWCS) [online]. Available WWW <URL: http://www.acq.osd.mil/osjtf/caserpt.htm> (1996).

References and Information Sources

[OSJTF 96] *Open Systems Joint Task Force Baseline Study* [online]. Available WWW <URL: http://www.acq.osd.mil/osjtf/baseline.doc> (1996).

Related Topics

Interfaces Design (AP.1.3.3)

- COTS and Open Systems - An Overview
- Graphical User Interface Builders
- Interface Definition Language

Related Topics

Software Architecture (AP.2.1)

- COTS and Open Systems - An Overview
- Fault Tolerant Computing
- File Server Software Architecture
- Real-Time Computing
- Reference Models, Architectures, Implementations -- An Overview
- Simplex Architecture
- Trusted Computing Base

Related Topics

Openness (Commonality) (QM.4.1.2)

- COTS and Open Systems - An Overview
- Network Management -- An Overview

References and Information Sources

[McCabe 94] McCabe, Thomas J. & Watson, Arthur H. "Software Complexity." *Crosstalk, Journal of Defense Software Engineering 7*, 12 (December 1994): 5-9.

References and Information Sources

[McCabe 89] McCabe, Thomas J. & Butler, Charles W. "Design Complexity Measurement and Testing." *Communications of the ACM 32,* 12 (December 1989): 1415-1425.

Cyclomatic Complexity

Software Technology Roadmap

Status

Advanced

Note

We recommend reading Maintenance of Operational Systems--An Overview before reading this description; it offers a view of the life cycle of software from development through reengineering. We also recommend concurrent reading of Maintainability Index Technique for Measuring Program Maintainability, which illustrates a specific application of cyclomatic complexity to quantify the maintainability of software. These descriptions provide a framework for assessing the applicability of cyclomatic complexity and other technologies to a specific environment.

Purpose and Origin

Cyclomatic *complexity* is the most widely used member of a class of static software metrics. Cyclomatic complexity may be considered a broad measure of *soundness* and *confidence* for a program. Introduced by Thomas McCabe in 1976, it measures the number of linearly-independent paths through a program module. This measure provides a single ordinal number that can be compared to the complexity of other programs. Cyclomatic complexity is often referred to simply as program complexity, or as McCabe's complexity. It is often used in concert with other software metrics. As one of the more widely-accepted software metrics, it is intended to be independent of language and language format [McCabe 94].

Cyclomatic complexity has also been extended to encompass the design and structural complexity of a system [McCabe 89].

Technical Detail

The cyclomatic complexity of a software module is calculated from a connected graph of the module (that shows the topology of control flow within the program):

Cyclomatic complexity (CC) = E - N + p

where E = the number of edges of the graph

N = the number of nodes of the graph

p = the number of connected components

To actually count these elements requires establishing a counting convention (tools to count cyclomatic complexity contain these conventions). The complexity number is generally considered to provide a stronger measure of a program's structural complexity than is provided by counting lines of code. Figure 6 is a connected graph of a simple program with a cyclomatic complexity of seven. Nodes are the numbered locations, which correspond to logic branch points; edges are the lines between the nodes.

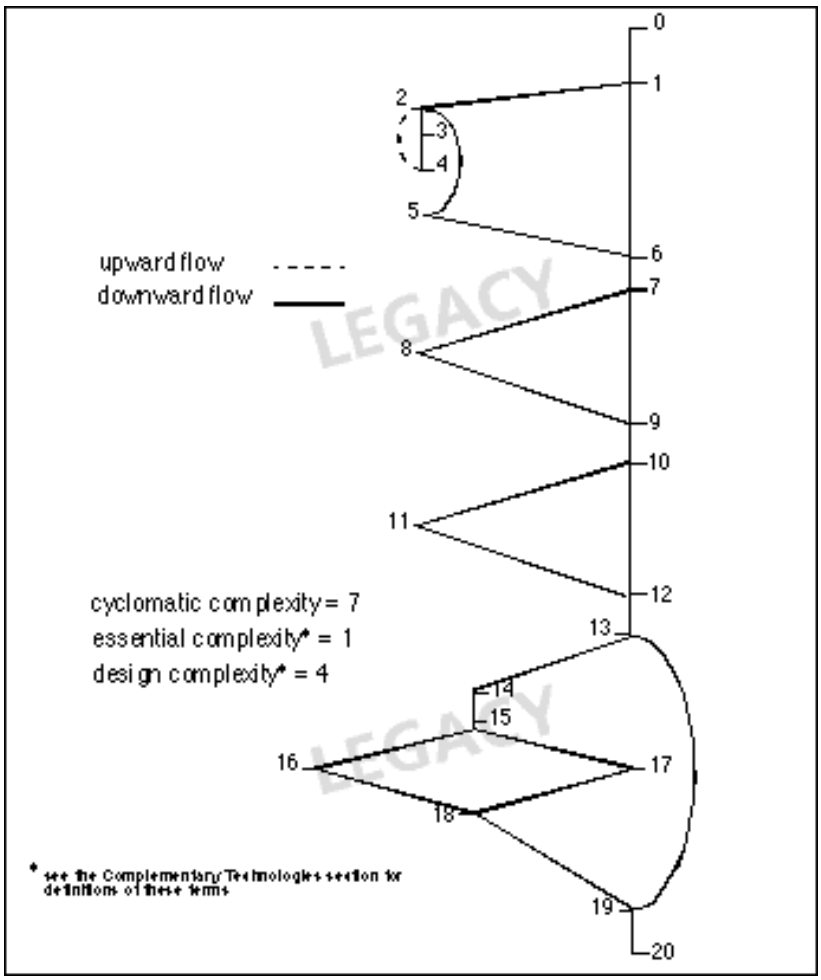


Figure 6: Connected Graph of a Simple Program

A large number of programs have been measured, and ranges of complexity have been established that help the software engineer determine a program's inherent risk and stability. The resulting calibrated measure can be used in development, maintenance, and reengineering situations to develop estimates of risk, cost, or program stability. Studies show a correlation between a program's cyclomatic complexity and its error frequency. A low cyclomatic complexity contributes to a program's *understandability* and indicates it is amenable to *modification* at lower risk than a more complex program. A module's cyclomatic

complexity is also a strong indicator of its *testability* (see Test planning under Usage Considerations).

A common application of cyclomatic complexity is to compare it against a set of threshold values. One such threshold set is in Table 4:

Table 4: Cyclomatic Complexity

Cyclomatic Complexity	Risk Evaluation
1-10	a simple program, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk program
greater than 50	untestable program (very high risk)

Usage Considerations

Cyclomatic complexity can be applied in several areas, including

- *Code development risk analysis*. While code is under development, it can be measured for complexity to assess inherent risk or risk buildup.
- *Change risk analysis in maintenance*. Code complexity tends to increase as it is maintained over time. By measuring the complexity before and after a proposed change, this buildup can be monitored and used to help decide how to minimize the risk of the change.
- *Test Planning*. Mathematical analysis has shown that cyclomatic complexity gives the exact number of tests needed to test every decision point in a program for each outcome. Thus, the analysis can be used for test planning. An excessively complex module will require a prohibitive number of test steps; that number can be reduced to a practical size by breaking the module into smaller, less-complex sub-modules.
- *Reengineering*. Cyclomatic complexity analysis provides knowledge of the structure of the operational code of a system. The risk involved in reengineering a piece of code is related to its complexity. Therefore, cost and risk analysis can benefit from proper application of such an analysis.

Cyclomatic complexity can be calculated manually for small program suites, but automated tools are preferable for most operational environments. For automated graphing and complexity calculation, the technology is language-sensitive; there must be a front-end source parser for each language, with

variants for dialectic differences.

Cyclomatic complexity is usually only moderately sensitive to program change. Other measures (see Complementary Technologies) may be very sensitive. It is common to use several metrics together, either as checks against each other or as part of a calculation set (see Maintainability Index Technique for Measuring Program Maintainability).

Maturity

Cyclomatic complexity measurement, an established but evolving technology, was introduced in 1976. Since that time it has been applied to tens of millions of lines of code in both Department of Defense (DoD) and commercial applications. The resulting base of empirical knowledge has allowed software developers to calibrate measurements of their own software and arrive at some understanding of its complexity. Code graphing and complexity calculation tools are available as part (or as options) of several commercial software environments.

Costs and Limitations

Cyclomatic complexity measurement tools are typically bundled inside commercially-available CASE toolsets. It is usually one of several metrics offered. Application of complexity measurements requires a small amount of training. The fact that a code module has high cyclomatic complexity does not, by itself, mean that it represents excess risk, or that it can or should be redesigned to make it simpler; more must be known about the specific application.

Alternatives

Cyclomatic complexity is one measure of structural complexity. Other metrics bring out other facets of complexity, including both structural and computational complexity, as shown in Table 5.

Table 5: Other Facets of Complexity

Complexity Measurement	Primary Measure of
Halstead Complexity Measures	Algorithmic complexity, measured by counting operators and operands
Henry and Kafura metrics	Coupling between modules (parameters, global variables, calls)
Bowles metrics	Module and system complexity; coupling via parameters and global variables

Troy and Zweben metrics	Modularity or coupling; complexity of structure (maximum depth of structure chart); calls-to and called-by
Ligier metrics	Modularity of the structure chart

Marciniak offers a more complete description of complexity measures and the complexity factors they measure [Marciniak 94].

Complementary Technologies

The following three metrics are specialized measures that are used in specific situations:

1. *Essential complexity*. This measures how much unstructured logic exists in a module (e.g., a loop with an exiting GOTO statement).
2. The program in Figure 6 has no such unstructured logic, so its essential complexity value is one.
3. *Design complexity*. This measures interaction between decision logic and subroutine or function calls.
4. The program in Figure 6 has a design complexity value of 4, which is well within the range of desirability.
5. *Data complexity*. This measures interaction between data references and decision logic.

Other metrics that are "related" to Cyclomatic complexity in general intent are also available in some CASE toolsets.

The metrics listed in Alternatives are also complementary; each metric highlights a different facet of the source code.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Cyclomatic Complexity
Application category	Test (AP.1.4.3) Reapply Software Lifecyle (AP.1.9.3) Reverse Engineering (AP.1.9.4) Reengineering (AP.1.9.5)

Quality measures category	Maintainability (QM.3.1) Testability (QM.1.4.1) Complexity (QM.3.2.1) Structuredness (QM.3.2.3)
Computing reviews category	Software Engineering Metrics (D.2.8) Complexity Classes (F.1.3) Tradeoffs Among Complexity Measures (F.2.3)

References and Information Sources

[Marciniak 94] Marciniak, John J., ed. *Encyclopedia of Software Engineering*, 131-165. New York, NY: John Wiley & Sons, 1994.

[McCabe 89] McCabe, Thomas J. & Butler, Charles W. "Design Complexity Measurement and Testing." *Communications of the ACM 32*, 12 (December 1989): 1415-1425.

[McCabe 94] McCabe, Thomas J. & Watson, Arthur H. "Software Complexity." *Crosstalk, Journal of Defense Software Engineering 7*, 12 (December 1994): 5-9.

[Perry 88] Perry, William E. *A Structured Approach to Systems Testing.* Wellesley, MA: QED Information Sciences, 1988.

[Watson 96] Watson, Arthur H. & McCabe, Thomas J. "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric." [online]. Available WWW, <URL: http://hissa.ncsl.nist.gov/HHRFdata/Artifacts/ITLdoc/235/mccabe.html> (1996).

Current Author/Maintainer

Edmond VanDoren, Kaman Sciences, Colorado Springs

Modifications

12 Jul 2000: Updated reference list

10 Jan 1997 (original)

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html
Last Modified: 24 July 2008

Glossary Term

Testability

the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met [IEEE 90]. Note: Not only is testability a measurement for software, it can also apply to the testing scheme.

References and Information Sources

[Marciniak 94] Marciniak, John J., ed. *Encyclopedia of Software Engineering,* 131-165. New York, NY: John Wiley & Sons, 1994.

Related Topics

Testability (QM.1.4.1)

- Cyclomatic Complexity
- Graphic Tools for Legacy Database Migration
- Halstead Complexity Measures
- Maintainability Index Technique for Measuring Program Maintainability

Glossary Term

Accuracy

a quantitative measure of the magnitude of error [IEEE 90].

References and Information Sources

[ORACLE7 92] "Two-Phase Commit," 22-1-22-21. *ORACLE7 Server Concept Manual* (6693-70-1292). Redwood City, CA: Oracle, 1992.

References and Information Sources

[UCSB 94] *The Performance of Two-Phase Commit Protocols in the Presence of Site Failures* (TRCS94-09). Santa Barbara, CA: University of California, Computer Science Department, April 1994.

Database Two Phase Commit

Software Technology Roadmap

Status

Advanced

Note

We recommend Three Tier Software Architectures *as prerequisite reading for this technology description.*

Purpose and Origin

Since the 1980s, two phase commit technology has been used to automatically control and monitor commit and/or rollback activities for transactions in a distributed database system. Two phase commit technology is used when data updates need to occur simultaneously at multiple databases within a distributed system. Two phase commits are done to maintain data integrity and *accuracy* within the distributed databases through synchronized locking of all pieces of a transaction. Two phase commit is a proven solution when data integrity in a distributed system is a requirement. Two phase commit technology is used for hotel and airline reservations, stock market transactions, banking applications, and credit card systems. For more details on two phase commit see the *ORACLE7 Server Concept Manual* and *The Performance of Two-Phase Commit Protocols in the Presence of Site Failures* [ORACLE7 92, UCSB 94].

Technical Detail

As shown in Figure 7, applying two phase commit protocols ensures that execution of data transactions are synchronized, either all committed or all rolled back (not committed) to each of the distributed databases.

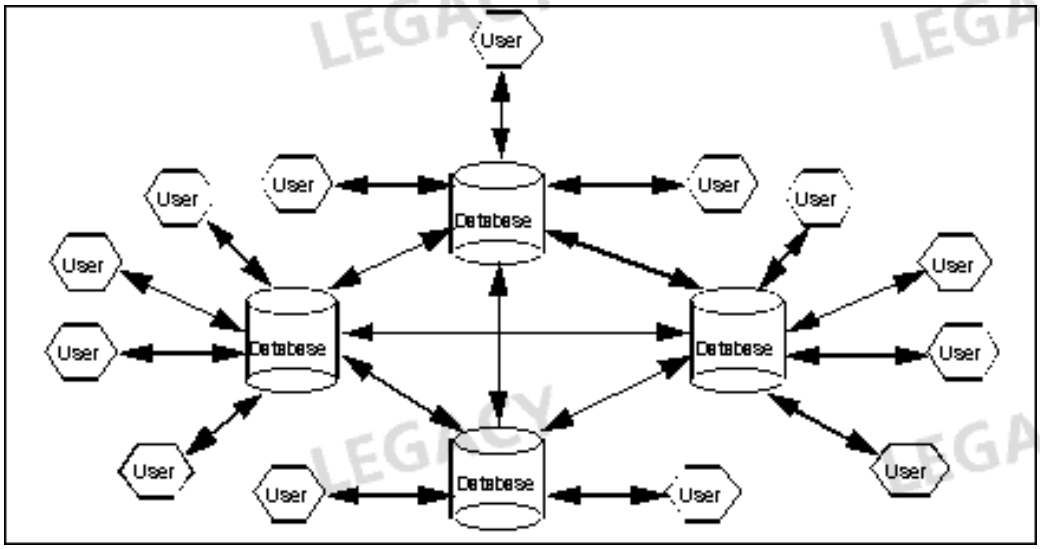


Figure 7: Distributed Databases When Two Phase Commit Happens Simultaneously Through the Network

When dealing with distributed databases, such as in the client/server architecture, distributed transactions need to be coordinated throughout the network to ensure data integrity for the users. Distributed databases using the two phase commit technique update all participating databases simultaneously.

Unlike non-distributed databases (see Figure 8), where a single change is or is not made locally, all participating databases must all commit or all rollback in distributed databases, even if there is a system or network failure at any node. This is how the two phase commit process maintains system data integrity.

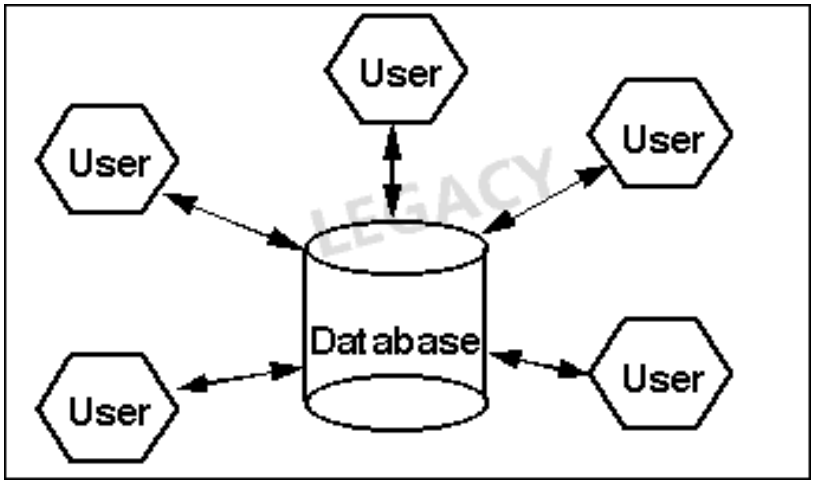


Figure 8: Non-Distributed Databases Make Only Local Updates

Two phase commit has two distinct processes that are accomplished in less than a fraction of a second:

1. The Prepare Phase, where the global coordinator (initiating database) requests that all participants (distributed databases) will promise to commit or rollback the transaction. (Note: Any database could serve as the global coordinator, depending on the transaction.)

2. The Commit Phase, where all participants respond to the coordinator that they are prepared, then the coordinator asks all nodes to commit the transaction. If all participants cannot prepare or there is a system component failure, the coordinator asks all databases to roll back the transaction.

Should there be a machine, network, or software failure during the two phase commit process, the two phase commit protocols will automatically and transparently complete the recovery with no work from the database administrator. This is done through use of pending transaction tables in each database where information about distributed transaction is maintained as they proceed through the two phase commit. Information in the pending transaction table is used by the recovery process to resolve any transaction of questionable status. This information can also be used by the database administrator to override automated recovery procedures by forcing a commit or a rollback to available participating databases.

Usage Considerations

Two phase commit protocols are offered in all modern distributed database products. However, the methods for implementing two phase commits may vary in the degree of automation provided. Some vendors provide a two phase commit implementation that is transparent to the application. Other vendors require specific programming of the calls into an application, and additional programming would be needed should rollback be a requirement; this situation would most likely result in an increase to program cost and schedule.

Maturity

The two phase commit protocol has been used successfully since the 1980s for hotel and airline reservations, stock market transactions, banking applications and credit card systems [Citron 93].

Costs and Limitations

There have been two performance issues with two phase commit:

1. If one database server is unavailable, none of the servers gets the updates. This is correctable if the software administrator forces the commit to the available participants, but if this is a recurring problem the administrator may not be able to keep up, thus causing system and network performance will deteriorate.
2. There is significant demand in network resources as the number of database servers to which data must be distributed increases. This is correctable through network tuning and correctly building the data distribution through database optimization techniques.

Currently, two phase commit procedures are vendor proprietary. There are no standards on how they should be implemented. X/Open has developed a standard that is being implemented in several transaction processing monitors

(see Transaction Processing Monitor Technology), but it has not been adopted by the database vendors [X/Open 96]. Two phase commit proprietary protocols have been published by several vendors.

Alternatives

An alternative to updating distributed databases with a two phase commit mechanism is to update multiple servers using a transaction queuing approach where transactions are distributed sequentially. Distributing transactions sequentially raises the problem of users working with different version of the data. In military usage, this could result in planning sorties for targets that have already been eliminated.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Database Two Phase Commit
Application category	Client-Server (AP.2.1.2.1) Data Management (AP.2.6.1)
Quality measures category	Accuracy (QM.2.1.2.1)
Computing reviews category	Distributed Systems (C.2.4)

References and Information Sources

[Citron 93]	Citron, A., et al. "Two-Phase Commit Optimization and Tradeoffs in the Commercial Environment," 520-529. *Proceedings of the Ninth International Conference on Data Engineering*. Vienna, Austria, April 19-23, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.
[ORACLE7 92]	"Two-Phase Commit," 22-1-22-21. *ORACLE7 Server Concept Manual* (6693-70-1292). Redwood City, CA: Oracle, 1992.
[Schussel 96]	Schussel, G. *Replication, The Next Generation of Distributed Database Technology* [online]. Available WWW <URL: http://www.dciexpo.com/geos/replica.htm> (1996).
[UCSB 94]	*The Performance of Two-Phase Commit Protocols in the Presence of Site Failures* (TRCS94-09). Santa Barbara, CA: University of California, Computer Science Department, April 1994.

[X/Open 96] *X/Open Web Site* [online]. Available WWW <URL: http://www.rdg.opengroup.org/> (1996).

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

David Altieri, GTE

Modifications

20 June 97: updated URLs for [Schussel 96] and [X/Open 96]; changed label for [UCSB 94]
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/dtpc_body.html
Last Modified: 24 July 2008

References and Information Sources

[Citron 93] Citron, A., et al. "Two-Phase Commit Optimization and Tradeoffs in the Commercial Environment," 520-529. *Proceedings of the Ninth International Conference on Data Engineering*. Vienna, Austria, April 19-23, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

References and Information Sources

[X/Open 96] *X/Open Web Site* [online]. Available WWW <URL: http://www.rdg.opengroup.org/> (1996).

Related Topics

Data Management (AP.2.6.1)

- Database Two Phase Commit

Related Topics

Accuracy (QM.2.1.2.1)

- Database Two Phase Commit

References and Information Sources

[DII COE 96a] *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification (I&RTS)* [online]. Available WWW <URL: http://spider.osfl.disa.mil/dii> (1996).

References and Information Sources

[DII COE 96b] *DII COE Style Guide*, Version 2.0 [online]. Available WWW <URL: http://spider.osfl.disa.mil/dii> (1996).

Defense Information Infrastructure Common Operating Environment (DII COE)

Software Technology Roadmap

Status

Advanced

Note

We recommend Reference Models, Architectures, Implementations--An Overview as prerequisite reading for this technology description.

Purpose and Origin

The Defense Information Infrastructure (DII) Common Operating Environment (COE) was developed in late 1993. DII COE was designed to eliminate duplication of development (in areas such as mapping, track management, and communication interfaces) and eliminate design incompatibility among Department of Defense (DoD) systems. Conceptually, the COE is designed to reduce program cost and risk through reusing proven solutions and sharing common functionality, rather than developing systems from "scratch" every time. The purpose of DII COE is to field systems with increasing *interoperability*, *reusability*, *portability*, and operational capability, while reducing development time, technical obsolescence, training requirements, and life-cycle cost.

DII COE reuses proven software components contributed by services and programs to provide common Command, Control, Communication, Computer and Intelligence (C4I) functions. For more details on DII COE see the *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification and the DII COE Style Guide* [DII COE 96a, DII COE 96b].

Technical Detail

DII COE technically is

- an architecture (including a set of guidelines and standards)
- a runtime environment
- software (including reusable components)
- a definition for acceptable application programming interfaces

The four major areas are described in further detail below:

1. *Architecture*. The DII COE architecture is fully compliant with the Department of Defense's Technical Architecture for Information Management (TAFIM Reference Model). The DII COE

architecture, presented in Figure 9, is a "plug and play," client/server architecture (implemented and running) that defines COE interfaces and how system components will fit together and interact.

2. *Runtime environment*. A runtime operating environment that includes a standard user system interface, operating system, and windowing environment. The DII COE architecture facilitates a developer in establishing the environment such that there is no conflict with other developers' products.

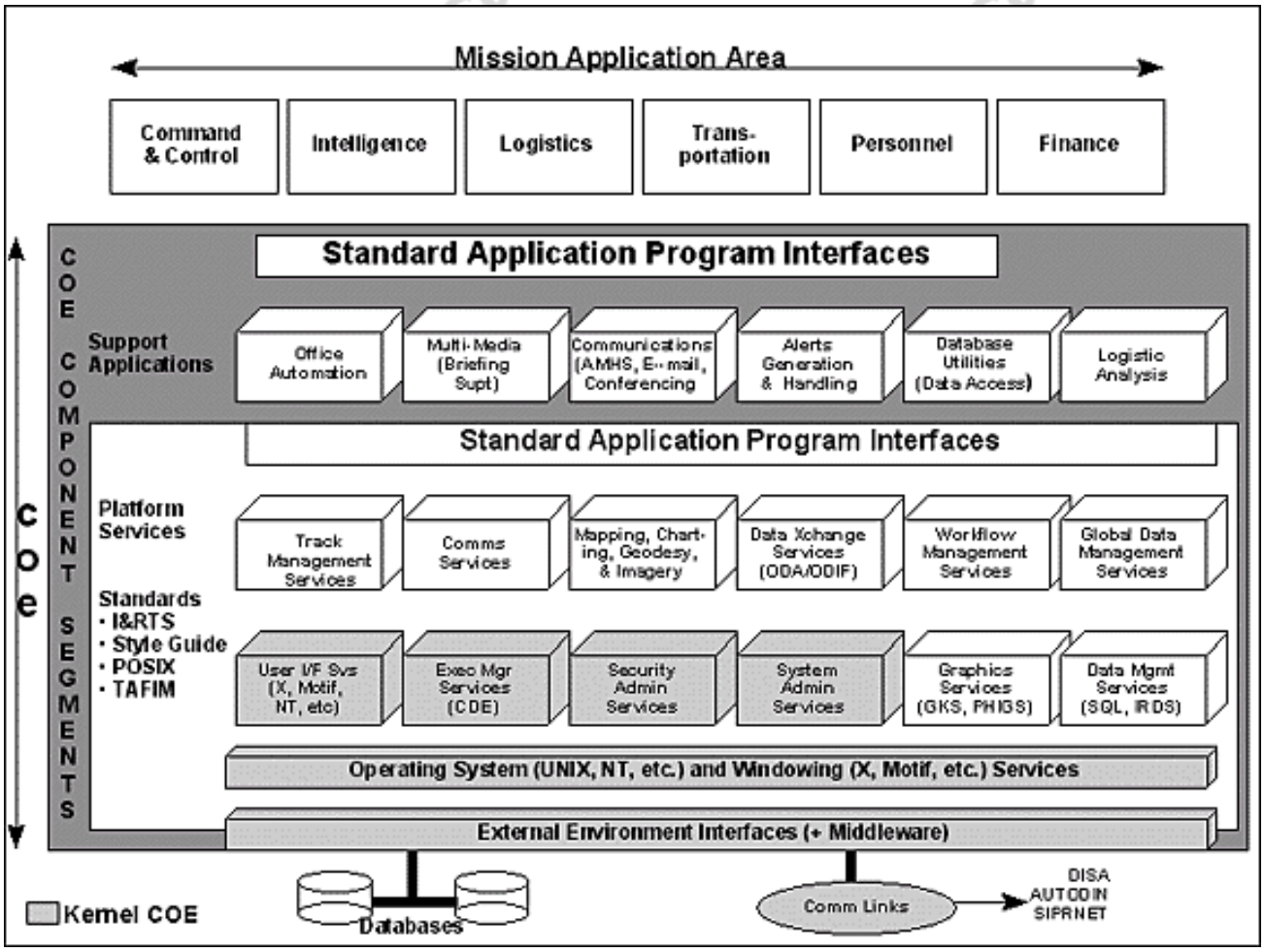


Figure 9: Defense Information Infrastructure Common Operating Environment [DII COE 96]

1. *Software*. A defined set of reusable functions that are already built (available commercially or as government products). Software (with the exception of the operating system and basic windowing software) is packaged in self-contained, manageable units called segments. Segments are the DII COE building block for constructing COE systems. Segments (mission applications and components) may consist of one or more Computer Software Configuration Items (CSCIs). Segments that are part of the reusable (by many mission applications) COE are referred to as COE component segments. Segments are named according to their meaning to operators, rather than internal software structures. Structuring the software into segments allows functionality to be easily added or removed from the target system to meet specific mission and site needs. DII COE databases are divided among segments (as are mission applications) according to the data they contain and the mission applications they support.
2. The kernel COE (light gray shading in Figure 9) is the minimal set of software that is required on every workstation. It includes operating system, windowing services, and external environment interfaces. There are normally five other services also included in the COE kernel: system administration, security administration, executive manager, and two

templates, one for creating privileged operator accounts, and one for creating non-privileged operator accounts. A subset of the kernel COE (defined as Bootstrap COE) is used during initial installation of COE. DII COE is hardware-independent and will run on any open system platform with a standards-based operating system, such as POSIX-compliant UNIX and Windows NT.

3. *APIs*. Two types of Application Programming Interfaces (APIs) are defined for accessing COE segments:
    - public APIs (COE interfaces that will be supported for the COE life cycle)
    - private APIs (interfaces that are supported for a short period of time to allow legacy systems to migrate to full COE compliance)
4. Newly-developed software (segments) must use public APIs to be COE compliant. The incremental implementation strategy for DII COE is to protect legacy system functionality while migrating to fully-compliant COE design by evolving from private APIs to public APIs.

Usage Considerations

There is only one COE available for use by other systems. This COE is currently being used by GCCS (Global Command and Control System) and GCSS (Global Combat Support System). Any system built to the COE infrastructure must access the services using the COE APIs. This improves interoperability between systems because the integration approach, the tool sets, and the segments (software components, not just algorithms) are used by each system [DII COE 96a].

Conceptually, compliance to COE standards ensures that software that is developed or modified for use within COE meets the intended requirements and goals and will evolve with the COE system. Another perspective is that compliance measures the degree to which "plug and play" is possible [Perry 96]. Owners of legacy systems should be familiar with COE compliance requirements to ensure that scoping and planning for future legacy enhancement includes COE requirements and goals.

There are a number of tradeoffs an organization must address when determining evolution of a legacy system to a system that meets COE compliance.

- What are the goals of the legacy system, and will migrating to COE compliance support achievement of the long range goals?
- What level of COE compliance will best and most cost effectively achieve the legacy system's long range goals?
- What is the current state of the legacy system- how compliant is it today?
- Given the current state of the legacy system, what resources are available to begin and follow through on the migration of the code to COE compliance?
- Does the organization want/need to control the legacy system code, and if not, when in the migration to COE is turning it over to DISA desirable?

Based on this analysis, the appropriate level and strategy for compliance can be determined. The four DII COE compliance categories are described in Table 6:

Table 6: DII COE Compliance Categories

Category	Name	Description

1	Runtime Environment	Measures compliance of the proposed segment's fit within the COE executing environment, the amount it reuses COE segments, whether it will run on a COE platform, and whether it will interfere with other segments. This can be done by prototyping within the COE.
2	Style Guide	Measures compliance of the proposed segment's user interface to the Style Guide [DII COE 96b]. This is to ensure that proposed segment will appear consistent with the rest of the COE-based system to minimize training and maintenance cost. Style Guide compliance can be done via a checklist based on the Style Guides requirements.
3	Architectural Compatibility	Measures compliance of the proposed segment's fit within the COE architecture, and the segment's potential life cycle as COE evolves. This can be done by evaluating the segment's use of TAFIM and COE standards and guidelines, and it's internal software structures.
4	Software Quality	Assesses a proposed segment's program risk and software maturity through the use of traditional software metrics. This can be done using measurements such as lines of code and McCabe complexity metrics (see Cyclomatic Complexity).

Category 1 (Runtime) compliance progresses through eight (8) levels of integration from a state of coexistence (agreement on a set of standards and ensure non-interference) with other COE segments, to federated (non-interference when on the same workstation), to fully integrated (share the same software and data). For a segment to be COE compliant, it must be qualified with a category name and compliance level. The following summarizes Category 1's eight levels of compliance; Appendix B of [DII COE 96a] provides a compliance *checklist* for each of the eight levels. Checklists are the current means of assessing progress toward compliance.

- *Standards Compliance Level One* - A proposed segment shares only a common set of standards with the rest of the COE environment, data sharing is undisciplined, and software reuse is minimal other than use of Commercial-Off-The Shelf (COTS) software products. Level 1 allows simultaneous execution of two systems.
- *Network Compliance Level Two* - Two segments will coexist on the same Local Area Network (LAN), but on different CPUs. There is limited data sharing and there may be common user interface "look and feel" if common user interface standards are applied.
- *Workstation Compliance Level Three* - Two applications can reside on the same LAN, share data, and coexist on the same workstation (environmental conflict have been resolved). The kernel COE, or its functional equivalent, resides on the workstation. Some COE components may be reused, but segmenting may not be done. Segments may not interoperate, and do not use the COE services.
- *Bootstrap Compliance Level Four* - Segment formatting is used in all applications. Segments share the bootstrap COE. Some segment conflicts can be automatically checked by the COE system. COE services are not being used. To switch between segments, users may still require separate login accounts. To submit a prototype to DISA for consideration of use,

Bootstrap Compliance is required, although these segments will not be fielded or put in the DISA maintained online library.

- *Minimal COE Compliance Level Five* - All segments share the same kernel COE (equivalent functionality is not acceptable at Level Five). Functionality is available through the COE Executive Manager. Segments may be successfully installed and removed through COE installation tools. Segment descriptor files describe boot, background, and local processes. Segments are registered and available through the online library. Applications appear integrated to the user, but there may be duplication of functionality. Interoperability is not guaranteed. DISA may allow Minimal COE Compliance segments to be installed and used as prototypes at a few sites for evaluation. They can be placed in the library. Currently, Level 5 appears to be the level many legacy systems are targeting.
- *Intermediate COE Compliance Level Six* - Segments use existing account groups, and reuse one or more COE segments. Minor differences may exist between the Style Guide [DII COE 96b] and the segment's graphical user interface implementation.
- *Interoperability Compliance Level Seven* - To ensure interoperability, proposed segments must reuse COE segments, including communication interfaces, message parsers, database tables, track data elements, and logistic services. Public APIs provide access with very few, if any, private APIs. There is no duplicate functionality in the COE segments. DISA requires Interoperability Compliance, for fieldable products and a migration strategy to full COE Compliance (Level 8). A migration strategy is not needed if the proposed segment will be phased out in the near term.
- *Full COE Compliance Level Eight* - All proposed new segments use COE services to the maximum extent possible. New segments are available through the Executive Manager and are completely integrated into the system. All segments fully comply with the Style Guide. [DII COE 96b]. All segments use only public APIs. There is no duplication of functionality any where in the system (as COE or as a mission application).

Two important resources for COE developers and operational sites are the online COE Software Repository System (CSRS) that is used to disseminate and manage software, and the COE Information Server (CINFO) that is used for documentation, meeting notices and general COE information. [DII COE 96a]

Maturity

COE initial proof of concept was created and installed in 1994 with Global Command and Control System (GCCS) Version 1.0. GCCS version 1.1 was used to monitor events during the 1994 Haiti crisis. In 1995, GCCS version 2.0 began fielding to a number of operational sites. There are two systems currently using DII COE: GCCS (developed in 1994 for a near term replacement for World-Wide Military Command and Control System) and GCSS (already fielded at a number of operational CINCs). It is expected that DII COE will be enhanced to include more functionality in such areas as Electronic Commerce/Electronic Data Interchange (EC/EDI), transportation, base support, personnel, health affairs, and finance. [DII COE 96a]

Costs and Limitations

DII COE is relatively new; actual cost, benefit, and risk information is still being collected.

Dependencies

DII COE is dependent of the evolution of TAFIM to ensure compatibility. (see TAFIM Reference Model). An additional dependency could be the Joint Technical Architecture (JTA). The JTA is now being mandated as a set of standards and guidelines for C4I systems, specifically in the area of

interoperability, to supersede TAFIM Volume 7, which did not appear to go far enough to ensure interoperability [JTA 96].

Alternatives

Under conditions where the TAFIM reference model and DII COE compliance is not required, an alternative model would be the Reference Model for Frameworks of Software Engineering Environments (known as the ECMA reference model [ECMA 93]) that is promoted in Europe, and used commercially and world-wide. Commercially-available Hewlett-Packard products use this model [HP 96]. Another alternative would be the Common Object Request Broker Architecture (CORBA) if the design called for object-oriented infrastructure (see Common Object Request Broker Architecture).

Complementary Technologies

Open systems (see COTS and Open Systems--An Overview) would be a complementary technology to DII COE because work done in open system supports the COE goal of achieving interoperable systems.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Defense Information Infrastructure Common Operating Environment
Application category	Software Architecture Models (AP.2.1.1)
Quality measures category	Interoperability (QM.4.1) Reusability (QM.4.4) Portability (QM.4.2)
Computing reviews category	not available

References and Information Sources

[DII COE 96a] *Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification (I&RTS)* [online]. Available WWW <URL: http://spider.osfl.disa.mil/dii> (1996).

[DII COE 96b] *DII COE Style Guide*, Version 2.0 [online]. Available WWW <URL: http://spider.osfl.disa.mil/dii> (1996).

[ECMA 93] *Reference Model for Frameworks of Software Engineering Environments*, 3rd Edition (NIST Special Publication 500-211/Technical Report ECMA TR/55). Prepared jointly by NIST and the European Computer Manufacturers Association (ECMA). Washington, DC: U.S. Government Printing Office, 1993.

[HP 96] *Integrated Solutions Catalog for the SoftBench Product Family*. Palo Alto, CA: Hewlett-Packard, 1996.

[JTA 96] U.S. Department of Defense. *Joint Technical Architecture (JTA)* [online]. Available WWW <URL: http://www-jta.itsi.disa.mil/>(1996).

[Perry 96] Perry, Frank. *Defense Information Infrastructure Common Operating Environment* (briefing). April 17, 1996. Arlington, VA: Defense Information Systems Agency.

Current Author/Maintainer

Darleen Sadoski, GTE

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/diicoe_body.html
Last Modified: 24 July 2008

References and Information Sources

[Perry 96] Perry, Frank. *Defense Information Infrastructure Common Operating Environment* (briefing). April 17, 1996. Arlington, VA: Defense Information Systems Agency.

References and Information Sources

[JTA 96] U.S. Department of Defense. *Joint Technical Architecture (JTA)* [online]. Available WWW <URL: http://www-jta.itsi.disa.mil/>(1996).

References and Information Sources

[ECMA 93] *Reference Model for Frameworks of Software Engineering Environments*, 3rd Edition (NIST Special Publication 500-211/Technical Report ECMA TR/55). Prepared jointly by NIST and the European Computer Manufacturers Association (ECMA). Washington, DC: U.S. Government Printing Office, 1993.

References and Information Sources

[HP 96] *Integrated Solutions Catalog for the SoftBench Product Family*. Palo Alto, CA: Hewlett-Packard, 1996.

Notes

[1] The DAA is the security official with the authority to say a system is secure and is permitted to be used.

Related Topics

System Security (AP.2.4.3)

- Covert Channel Analysis in MLS Systems
- Firewalls and Proxies
- Intrusion Detection
- Message Digest
- Multi-Level Secure One Way Guard with Random Acknowledgment
- Network Auditing Techniques
- Network Security Guards
- Nonrepudiation in Network Communications
- Public Key Cryptography
- Public Key Digital Signatures
- Rule-Based Intrusion Detection
- Statistical-Based Intrusion Detection

Related Topics

Vulnerability (QM.2.1.4.1)

- Firewalls and Proxies
- Multi-Level Secure One Way Guard with Random Acknowledgment

Notes

[1] A voluntary, non-treaty organization founded in 1946 which is responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of the 89 member countries, including ANSI for the U.S.

References and Information Sources

[X.700 96] *X.700 and Other Network Management Services* [online]. Available WWW <URL: http://ganges.cs.tcd.ie/4ba2/x700/index.html> (1996).

Notes

[2] A managed device is any type of node residing on a network, such as a computer, printer or routers that contain a management agent.

Network Management--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

In the early 1980s computer networks began to grow and be interconnected. As the size of these networks grew, they became harder to manage and maintain, thus the need for network management was realized. One of the oldest forms of network management is the use of the remote login to monitor or configure a network device; however, today more sophisticated network management tools are available. Network management is a requirement for anyone who wants to control and monitor their networks.

Technical Detail

Functional Areas of Network Management. Network management is the ability to control and monitor a computer network from a central location. The International Organization for Standardization (ISO)[1] defined a conceptual model for describing the key functional areas of network management which are described below [X.700 96]:

Note: In general, network management systems available from vendors today do *not* support all the key functional areas, and in a supported functional area, the coverage may be incomplete even though support is claimed.

- Fault Management: Provides facilities that allow network managers to discover faults in managed devices,[2] the network, and network operation, to determine their cause and to take remedial action. To enable this, fault management provides mechanisms to:
  - Report the occurrence of faults
  - Log reports
  - Perform diagnostic tests
  - Correct faults (possibly automatically)
- Configuration Management: Monitors network configuration information so that the effects of specific hardware and software can be managed and tracked. It may provide the ability to initialize, reconfigure, operate and shut down managed devices.
- Accounting: Measures network utilization of individual users or groups to:
  - Provide billing information

- Regulate users or groups
- Help keep network performance at an acceptable level

- Performance Management: Measures various aspects of network performance including the gathering and analysis of statistical data about the system so that it may be maintained at an acceptable level. Performance management provides the ability to:
  - Obtain the utilization and error rates of network devices
  - Provide a consistent level of performance by ensuring that devices have a sufficient capacity.
- Security Management: Controls access to network resources so that information can not be obtained without authorization by:
  - Limiting access to network resources
  - Providing notification of security breaches and attempts

Network Management Architecture. In general, network management systems have the same basic architecture, as shown in Figure 27.

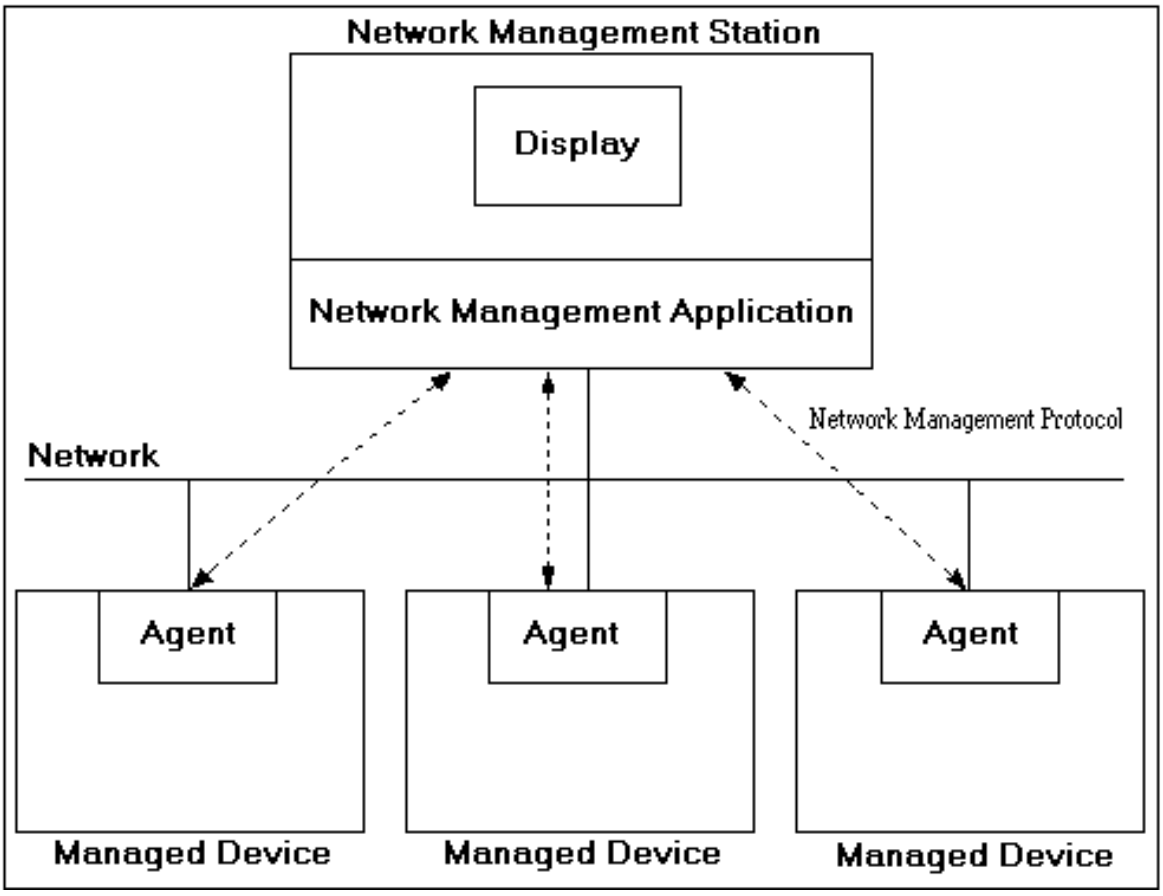


Figure 27: Typical Network Management Architecture [Cisco 96]

The architecture consists of the following elements:

- Network Management Station(s): The network management station[3] runs the network management application[4] that gathers information about managed devices from the management agent[5] which resides within a managed device. The network management application typically must process large amounts of data, react to events, and prepare relevant information for display. It usually has

a control console with a GUI interface which allows the operator to view a graphical representation of the network, control managed devices on the network and program the network management application. Some network management applications can be programmed to react to information collected from management agents and/or set thresholds with the following actions:
    - Perform tests and automatic corrective actions (reconfiguration, shutdown of a managed device)
    - Logging network events
    - Present status information and alerts to operator
- Managed Devices: A managed device can be any type of node residing on a network, such as a computer, printer or router. Managed devices contain a management agent.
- Management agents: Provides information about the managed device to the network management application(s) and may also accept control information.
- Network management protocol: Protocol used by the network management application(s) and the management agent to exchange management information.
- Management Information: The information that is exchanged between the network management application(s) and the management agents that allows the monitoring and control of a managed device.

Network management software (network management applications and agents) is usually based upon a particular network management protocol and the network management capabilities provided with the software are usually based upon the functionality supported by the network management protocol. Most systems use open protocols; however, some network management software is based upon vendor specific proprietary protocols. The selection of network management software is driven by the following factors:

- Network environment (scope and nature of the network)
- Network management requirements
- Cost
- Operating systems involved

The two most common network management protocols are the

- Simple Network Management Protocol
- Common Management Information Protocol

SNMP is by far the most widely used network management protocol and use is widespread in LAN environments. CMIP is used extensively in telecommunication environments, where networks tend to be large and complex.

Usage Considerations

A considerable amount of time is usually required to effectively deploy and learn to use network management software. This is because network managers must be extremely familiar with the network management protocol and the data structures associated with the network management information. Network management protocols and the data structures associated with the network management information are typically complex.

Many network management implementations do not provide support for network devices which use vendor specific protocols.

A network management system for a small isolated network may not be cost effective or needed. This of course depends on functionality, reliability and performance requirements of the network and attached systems.

Maturity

Network management software often lacks the functionality needed to effectively manage a network. Some of this can be attributed to the deficiencies in the network management protocols.

Numerous network management packages are available from a wide variety of vendors. Some packages are simple and provide network management facilities for a single network, others can be complex and handle multiple types of networks. New products and enhancements to existing network management packages are announced frequently.

Costs and Limitations

Network management systems can be quite expensive, and are often complex. Personnel with specialized training are often required to effectively configure, maintain and operate the network management system.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Network Management
Application category	Protocols (AP.2.2.3) Network Management (AP.2.2.2)
Quality measures category	Openness (QM.4.1.2) Interoperability (QM.4.1) Maintainability (QM.3.1) Scalability (QM.4.3) Security (QM.2.1.5)
Computing reviews category	Network Operations (C.2.3) Distributed Systems (C.2.4)

References and Information Sources

[Cisco 96]	*Internetworking Technology Overview / Network Management Basics* [online]. Available WWW <URL: http://cio.cisco.com/univercd/data/doc/cintrnet/ito/55018.htm> (1996).
[Stallings 93]	Stallings, William. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards.* Reading, MA: Addison-Wesley, 1993.
[Vallillee 96]	Vallillee, Tyler. *SNMP & CMIP: An Introduction To Network Management* [online]. Available WWW <URL: http://www.inforamp.net/~kjvallil/t/snmp.html> (1996).
[X.700 96]	*X.700 and Other Network Management Services* [online]. Available WWW <URL: http://ganges.cs.tcd.ie/4ba2/x700/index.html> (1996).

Current Author/Maintainer

Dan Plakosh, SEI

Modifications

9 February 98: Minor modifications

19 June 97 (original)

Footnotes

[1] A voluntary, non-treaty organization founded in 1946 which is responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of the 89 member countries, including ANSI for the U.S.

[2] A managed device is any type of node residing on a network, such as a computer, printer or routers that contain a management agent.

[3] The network management station is the system that hosts the network management application.

[4] The network management application is the application that provides the ability to monitor and control the network.

[5] The network management agent is the software that resides in a managed device that allows the device to be monitored and/or controlled by a network management application.

URL: http://www.sei.cmu.edu/str/descriptions/network_body.html
Last Modified: 24 July 2008

References and Information Sources

[Cisco 96] *Internetworking Technology Overview / Network Management Basics* [online]. Available WWW <URL: http://cio.cisco.com/univercd/data/doc/cintrnet/ito/55018.htm> (1996).

Notes

[3] The network management station is the system that hosts the network management application.

Notes

[4] The network management application is the application that provides the ability to monitor and control the network.

Notes

[5] The network management agent is the software that resides in a managed device that allows the device to be monitored and/or controlled by a network management application.

Related Topics

Integrity (QM.2.1.4.1.1)

- Nonrepudiation in Network Communications

Related Topics

Trustworthiness (QM.2.1.4)

- Java
- Nonrepudiation in Network Communications
- Public Key Digital Signatures

Glossary Term

Productivity

the quality or state of being productive [Webster 87].

References and Information Sources

[Baudoin 96] Baudoin, Claude & Hollowell, Glenn. *Realizing the Object-Oriented Lifecycle*. Upper Saddle River, NJ: Prentice Hall, 1996.

References and Information Sources

[Yourdon 79] Yourdon, E. & Constantine, L. *Structured Design*. Englewood Cliffs, NJ: Prentice Hall, 1979.

References and Information Sources

[Malan 95] Malan, R.; Coleman, D.; & Letsinger, R. "Lessons Learned from the Experiences of Leading-Edge Object Technology Projects in Hewlett-Packard," 33-46. *Proceedings of Tenth Annual Conference on Object-Oriented Programming Systems Languages and Applications*. Austin, TX, October 15-19, 1995. Palo Alto, CA: Hewlett-Packard, 1995.

References and Information Sources

[Kamath 93] Kamath, Y. H.; Smilan, R. E.; & Smith, J. G. "Reaping Benefits With Object-Oriented Technology." *AT&T Technical Journal 72*, 5 (September/October 1993): 14-24.

Related Topics

Define and Develop Requirements (Elicitation Techniques, Specification Techniques, Modeling, Prototyping) (AP.1.2.2.1)

- Algebraic Specification Techniques
- Entity-Relationship Modeling
- Essential Systems Analysis
- Formal Methods
- Functional Decomposition
- Model Checking
- Object-Oriented Analysis
- Specification Construction Techniques
- Structured Analysis and Design

Related Topics

Analyze Functions (AP.1.2.1.1)

- Essential Systems Analysis
- Functional Decomposition
- Object-Oriented Analysis
- Structured Analysis and Design

References and Information Sources

[Martin 93] Martin, James. *Principles of Object-Oriented Analysis and Design*. Englewood Cliffs, NJ: Prentice Hall, 1993.

References and Information Sources

[Vorwerk 94] Vorwerk, Raymond. "Towards a True OBBMS." *Object Magazine 3*, 5 (January 1994): 38-39.

References and Information Sources

[Tkach 94] Tkach, Daniel & Puttick, Richard. *Object Technology in Application Development*. Redwood City, CA: Benjamin/Cummings Publishing Company, 1994.

References and Information Sources

[Desanti 94] Desanti, Mike & Gomsi, Jeff. "A Comparison of Object and Relational Database Technologies." *Object Magazine 3*, 5 (January 1994): 51-57.

References and Information Sources

[Object 96] "Focus on ODBMS Debunking the Myths." *Object Magazine 5*, 9 (February 1996): 21-23.

Related Topics

Database Design (Conceptual, Logical, Physical) (AP.1.3.2)

- Graphic Tools for Legacy Database Migration
- Object-Oriented Database
- Relational DBMS
- SQL

Related Topics

Database Administration (AP.1.9.1)

- Data Mining
- Data Warehousing
- Object-Oriented Database
- Relational DBMS

Related Topics

Databases (AP.2.6)

- Object-Oriented Database
- Relational DBMS

References and Information Sources

[Baudoin 96] Baudoin, Claude & Hollowell, Glenn. *Realizing the Object-Oriented Lifecycle*. Upper Saddle River, NJ: Prentice Hall, 1996.

References and Information Sources

[Maring 96] Maring, B. "Object-Oriented Development of Large Applications." *IEEE Software 13*, 3 (May 1996): 33-40.

References and Information Sources

[Tokar 96] Tokar, Joyce L. "Ada 95: The Language for the 90's and Beyond." *Object Magazine 6*, 4 (June 1996): 53-56.

References and Information Sources

[Malan 95] Malan, R.; Coleman, D.; & Letsinger, R. "Lessons Learned from the Experiences of Leading-Edge Object Technology Projects in Hewlett-Packard," 33-46. *Proceedings of Tenth Annual Conference on Object-Oriented Programming Systems Languages and Applications*. Austin, TX, October 15-19, 1995. Palo Alto, CA: Hewlett-Packard, 1995.

References and Information Sources

[Wade 94] Wade, Andrew E. "Distributed Client-Server Databases." *Object Magazine 4*, 1 (April 1994): 47-52.

References and Information Sources

[Cobb 95] Cobb, Edward E. "TP Monitors and ORBs: A Superior Client/Server Alternative." *Object Magazine 4*, 9 (February 1995): 57-61.

Object Request Broker

Software Technology Roadmap

Status

Complete

Note

We recommend Middleware, as prerequisite reading for this technology description.

Purpose and Origin

An object request broker (ORB) is a middleware technology that manages communication and data exchange between objects. ORBs promote *interoperability* of distributed object systems because they enable users to build systems by piecing together objects- from different vendors- that communicate with each other via the ORB [Wade 94]. The implementation details of the ORB are generally not important to developers building distributed systems. The developers are only concerned with the object interface details. This form of information hiding enhances system *maintainability* since the object communication details are hidden from the developers and isolated in the ORB [Cobb 95].

Technical Detail

ORB technology promotes the goal of object communication across machine, software, and vendor boundaries. The relevant functions of an ORB technology are

- interface definition
- location and possible activation of remote objects
- communication between clients and object

An object request broker acts as a kind of telephone exchange. It provides a directory of services and helps establish connections between clients and these services [CORBA 96, Steinke 95]. Figure 21 illustrates some of the key ideas.

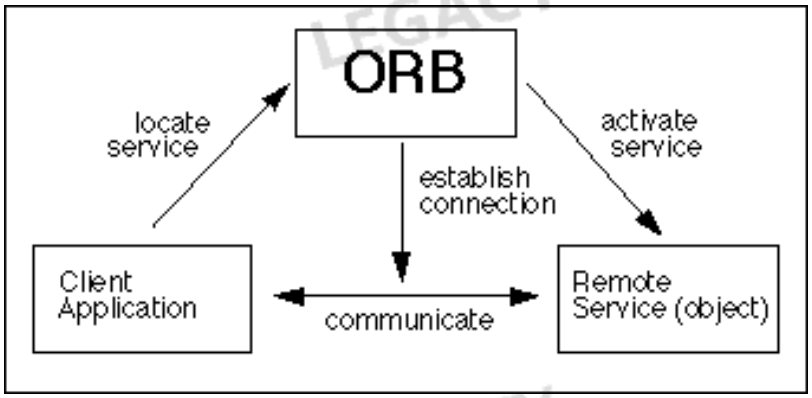


Figure 21: Object Request Broker

The ORB must support many functions in order to operate consistently and effectively, but many of these functions are hidden from the user of the ORB. It is the responsibility of the ORB to provide the illusion of locality, in other words, to make it appear as if the object is local to the client, while in reality it may reside in a different process or machine [Reddy 95]. Thus the ORB provides a framework for cross-system communication between objects. This is the first technical step toward interoperability of object systems.

The next technical step toward object system interoperability is the communication of objects across platforms. An ORB allows objects to hide their implementation details from clients. This can include programming language, operating system, host hardware, and object location. Each of these can be thought of as a "transparency,"[1] and different ORB technologies may choose to support different transparencies, thus extending the benefits of object orientation across platforms and communication channels.

There are many ways of implementing the basic ORB concept; for example, ORB functions can be compiled into clients, can be separate processes, or can be part of an operating system kernel. These basic design decisions might be fixed in a single product; or there might be a range of choices left to the ORB implementer.

There are two major ORB technologies:

- The Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) specification
- Microsoft's Component Object Model (see Component Object Model (COM), DCOM, and Related Capabilities)

An additional, newly-emerging ORB model is Remote Method Invocation (RMI); this is specified as part of the Java language/virtual machine. RMI allows Java objects to be executed remotely. This provides ORB-like capabilities as a native extension of Java [RMI 97].

A high-level comparison of ORB technologies is available in Table 8. Details are available in the referenced technology descriptions.

Usage Considerations

Successful adoption of ORB technology requires a careful analysis of the current and future software architectural needs of the target application and analysis of how a particular ORB will satisfy those needs [Abowd 96]. Among the many things to consider are platform availability, support for various programming languages, as well as implementation choices and product performance parameters. After performing this analysis, developers can make informed decisions in choosing the ORB best suited for their application's needs.

Table 8: Comparison of ORB Technologies

ORB	Platform Availability	Applicable to	Mechanism	Implementations
COM/ DCOM	originally PC platforms, but becoming available on other platforms	"PC-centric" distributed systems architecture	APIs to proprietary system[2]	one[3]
CORBA	platform-independent and interoperability among platforms	general distributed system architecture	specification of distributed object technology	many[4]
Java/ RMI	wherever Java virtual machine (VM) executes	general distributed system architecture and Web-based Intranets	implementation of distributed object technology	various[5]

Maturity

As shown in Table 8, there are a number of commercial ORB products available. ORB products that are not compliant with either CORBA or OLE also exist; however, these tend to be vendor-unique solutions that may affect system interoperability, portability, and maintainability.

Major developments in commercial ORB products are occurring, with life cycles seemingly lasting only four to six months. In addition, new ORB technology (Java/RMI) is emerging, and there are signs of potential "mergers" involving two of the major technologies. The continued trend toward Intranet- and Internet-based applications is another stimulant in the situation. Whether these

commercial directions are fully technically viable and will be accepted by the market is unknown.

Given the current situation and technical uncertainty, potential users of ORB technologies need to determine

- what new features ORB technologies add beyond technologies currently in use in their organizations
- the potential benefits from using these new features
- the key risks involved in adopting the technology as a whole
- how much risk is acceptable to them

One possible path would be to undertake a disciplined and "situated" technology evaluation. Such an evaluation, as described by Brown and Wallnau, focuses on evaluating so-called "innovative" technologies and can provide technical information for adoption that is relative to the current/existing approaches in use by an organization [Brown 96, Wallnau 96]. Such a technology evaluation could include pilot projects focusing on model problems pertinent to the individual organization.

Costs and Limitations

The license costs of the ORB products from the vendors listed above are dependent on the required operating systems and the types of platform. ORB products are available for all major computing platforms and operating systems.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Object Request Broker
Application category	Client/Server (AP.2.1.2.1), Client/Server Communication (AP.2.2.1)
Quality measures category	Interoperability (QM.4.1), Maintainability (QM.3.1)
Computing reviews category	Distributed systems (C.2.4), Object-Oriented programming (D.1.5)

References and Information Sources

[Abowd 96] Abowd, Gregory, et al. "Architectural Analysis of ORBs." *Object Magazine 6*, 1 (March 1996): 44-51.

[Brown 96] Brown, A. & Wallnau, K. "A Framework for Evaluating Software Technology." *IEEE Software 13*, 5 (September 1996): 39-49.

[Cobb 95] Cobb, Edward E. "TP Monitors and ORBs: A Superior Client/ Server Alternative." *Object Magazine 4*, 9 (February 1995): 57-61.

[CORBA 96] *The Common Object Request Broker: Architecture and Specification*, Version 2.0. Framingham, MA: Object Management Group, 1996. Also available [online] WWW <URL: http://www.omg.org> (1996).

[Reddy 95] Reddy, Madhu. "ORBs and ODBMSs: Two Complementary Ways to Distribute Objects." *Object Magazine 5*, 3 (June 1995): 24-30.

[RMI 97] Remote Method Invocation [online]. Available WWW <URL: http://java.sun.com/products/jdk/1.1/docs/guide/rmi> (1997).

[Steinke 95] Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

[Tkach 94] Tkach, Daniel & Puttick, Richard. *Object Technology in Application Development*. Redwood City, CA: Benjamin/ Cummings Publishing Company, 1994.

[Wade 94] Wade, Andrew E. "Distributed Client-Server Databases." *Object Magazine 4*, 1 (April 1994): 47-52.

[Wallnau 96] Wallnau, Kurt & Wallace, Evan. "A Situated Evaluation of the Object Management Group's (OMG) Object Management Architecture (OMA)," 168-178. *Proceedings of the OOPSLA'96*. San Jose, CA, October 6-10, 1996. New York, NY: ACM, 1996. Presentation available [online] FTP. <URL: ftp://ftp.sei.cmu.edu/pub/corba/OOPSLA/present> (1996).

Current Author/Maintainer

Kurt Wallnau, SEI
John Foreman, SEI

External Reviewers

Ed Morris, SEI
Richard Soley, VP, Chief Technical Officer, Object Management Group

Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM; added notes to Table 8
9 April 97: minor edits and reorganization; no meaningful content changes
10 Jan 97 (original): Mike Bray, Lockheed-Martin Ground Systems

Footnotes

[1] transparency: making something invisible to the client

[2] COM/DCOM specifications have been turned over to the Open Group, but the outcome of this standardization activity remains unclear.

[3] Microsoft maintains the only implementation of PC platforms, and is working closely with selected vendors to migrate technology to alternate platforms.

[4] Examples include ORBIX by IONA Technology, NEO by SunSoft, VisiBroker by VisiGenic, PowerBroker by Expersoft, SmallTalkBroker by DNS Technologies, Object Director by Fujitsu, DSOM by IBM, DAIS by ICL, SORBET by Siemens Nixdorf, and NonStop DOM by Tandem.

[5] Implementations of the Java VM have been ported to various platforms.

URL: http://www.sei.cmu.edu/str/descriptions/orb_body.html
Last Modified: 24 July 2008

References and Information Sources

[Reddy 95] Reddy, Madhu. "ORBs and ODBMSs: Two Complementary Ways to Distribute Objects." *Object Magazine 5*, 3 (June 1995): 24-30.

Notes

[1] transparency: making something invisible to the client

References and Information Sources

[RMI 97] Remote Method Invocation [online]. Available WWW <URL: http://java.sun.com/products/jdk/1.1/docs/guide/rmi> (1997).

References and Information Sources

[Abowd 96] Abowd, Gregory, et al. "Architectural Analysis of ORBs." *Object Magazine 6*, 1 (March 1996): 44-51.

Notes

[2] COM/DCOM specifications have been turned over to the Open Group, but the outcome of this standardization activity remains unclear.

Notes

[3] Microsoft maintains the only implementation of PC platforms, and is working closely with selected vendors to migrate technology to alternate platforms.

Notes

[4] Examples include ORBIX by IONA Technology, NEO by SunSoft, VisiBroker by VisiGenic, PowerBroker by Expersoft, SmallTalkBroker by DNS Technologies, Object Director by Fujitsu, DSOM by IBM, DAIS by ICL, SORBET by Siemens Nixdorf, and NonStop DOM by Tandem.

Notes

[5] Implementations of the Java VM have been ported to various platforms.

References and Information Sources

[Brown 96] Brown, A. & Wallnau, K. "A Framework for Evaluating Software Technology." *IEEE Software 13*, 5 (September 1996): 39-49.

References and Information Sources

[Wallnau 96] Wallnau, Kurt & Wallace, Evan. "A Situated Evaluation of the Object Management Group's (OMG) Object Management Architecture (OMA)," 168-178. *Proceedings of the OOPSLA'96*. San Jose, CA, October 6-10, 1996. New York, NY: ACM, 1996. Presentation available [online] FTP. <URL: ftp://ftp.sei.cmu.edu/pub/corba/OOPSLA/present> (1996).

Notes

[1] formerly Unisys Defense Systems, Reston, VA

Notes

[2] Defense Advanced Research Projects Agency (DARPA) Software Technology for Adaptable, Reliable Systems (STARS)

References and Information Sources

[Simos 96] Simos, M., et al. *Software Technology for Adaptable Reliable Systems (STARS) Organization Domain Modeling (ODM) Guidebook* Version 2.0 (STARS-VC-A025/001/00). Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996. Also available [online] WWW <URL: http://www.asset.com/WSRD/abstracts/ABSTRACT_1176.html> (1996).

References and Information Sources

[STARS 93] *Conceptual Framework for Reuse Processes* Volume I, Definition, Version 3.0 (STARS-VC-A018/001/00). Reston, VA: Software Technology for Adaptable Reliable Systems, 1993.

References and Information Sources

[STARS 96c] *Open RLF* (STARS-PA31-AE08/001/00). Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996.

References and Information Sources

[STARS 96a] *Canvas Knowledge Acquisition Guide Book* Version 1.0 (STARS-PA29-AC01/001/00) Reston, VA: Software Technology for Adaptable, Reliable Systems, 1996.

References and Information Sources

[Klinger 96] Klinger, Carol & Solderitsch, James. *DAGAR: A Process for Domain Architecture Definition and Asset Implementation* [online]. Available WWW <URL: http://source.asset.com/stars/darpa/Papers/ArchPapers.html> (1996).

References and Information Sources

[STARS 96b] *Domain Architecture-Based Generation for Ada Reuse (DAGAR) Guidebook* Version 1.0. Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996.

References and Information Sources

[Cornwell 96] Cornwell, Patricia Collins. "HP Domain Analysis: Producing Useful Models for Reusable Software." *HP Journal* (August 1996): 46-55.

References and Information Sources

[Lettes 96] Lettes, Judith A. & Wilson, John. *Army STARS Demonstration Project Experience Report* (STARS-VC-A011/003/02). Manassas, VA: Loral Defense Systems-East, 1996.

Related Topics

Domain Engineering (AP.1.2.4)

- Comparative/Taxonomic Modeling
- Domain Engineering and Domain Analysis
- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Feature-Oriented Domain Analysis
- Organization Domain Modeling
- Visual Programming Techniques

Notes

[1] Personal Software Process and PSP are service marks of Carnegie Mellon University.

References and Information Sources

[Humphrey 95] Humphrey, Watts. *A Discipline for Software Engineering.* Reading, MA: Addison-Wesley Publishing Company, 1995.

Notes

[2] Capability Maturity Model and CMM are service marks of Carnegie Mellon University.

References and Information Sources

[Paulk 95] Carnegie Mellon University, Software Engineering Institute (Principal Contributors and Editors: Mark C. Paulk, Charles V. Weber, Bill Curtis, and Mary Beth Chrissis), The Capability Maturity Model: Guidelines for Improving the Software Process, ISBN 0-201-54664-7, Addison-Wesley Publishing Company, Reading, MA, 1995.

Notes

[3] Team Software Process and TSP are service marks of Carnegie Mellon University.

References and Information Sources

[Humphrey 96b] Humphrey, Watts. "The PSP and Personal Project Estimating." *American Programmer 9*, 6 (June 1996): 2-15.

References and Information Sources

[Humphrey 96a] Humphrey, Watts. "Using a Defined and Measured Personal Software Process." *IEEE Software 13*, 3 (May 1996): 77-88.

References and Information Sources

[McAndrews 00] McAndrews, Donald R. *The Team Software ProcessSM (TSP^{SM}): An Overview and Preliminary Results of Using Disciplined Practices* (CMU/SEI-2000-TR-015). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2000.

Related Topics

Code (AP.1.4.2)

- Graphical User Interface Builders
- Halstead Complexity Measures
- Peer Reviews
- Personal Software Process for Module-Level Development
- Rate Monotonic Analysis
- Nonrepudiation in Network Communications
- Software Walkthroughs

Related Topics

Unit Testing (Code analyzers, Data analyzers, Black-box/Functional Testing, White-box/ Structural Testing) (AP.1.4.3.4)

- Halstead Complexity Measures
- Maintainability Index Technique for Measuring Program Maintainability
- Peer Reviews
- Personal Software Process for Module-Level Development
- Nonrepudiation in Network Communications
- Software Walkthroughs

Related Topics

Maintenance Control (QM.5.1.2.3)

- Personal Software Process for Module-Level Development

Related Topics

Productivity (QM.5.2)

- Function Point Analysis
- Personal Software Process for Module-Level Development

References and Information Sources

[Schneier 96] Schneier, Bruce. *Applied Cryptography*. New York, NY: John Wiley & Sons, 1996.

References and Information Sources

[White 96] White, Gregory B.; Fisch, Eric A.; & Pooch, Udo W. *Computer System and Network Security*. Boca Raton, FL: CRC Press, 1996.

Notes

[1] Of course they are not absolutely unique. We say unique here because it is extremely unlikely statistically for two files to have the same MAC and, more importantly, it is extremely difficult for an attacker/malicious user to create/craft two files having the same MAC.

Public Key Digital Signatures

Software Technology Roadmap

Status

Advanced

Note

We recommend Computer System Security- an Overview as prerequisite reading for this technology description.

Purpose and Origin

Public key digital signature techniques provide data integrity and source authentication capabilities to enhance data trustworthiness in computer networks. This technology uses a combination of a message authentication code (MAC) to guarantee the integrity of data and unique features of paired public and private keys associated with public key cryptography to uniquely authenticate the sender [Schneier 96, Abrams 95]. This technology was first defined in the early 1980s with the development of public key cryptography but has received renewed interest as an authentication mechanism on the Internet.

Technical Detail

Trustworthiness of data received by a computer from another computer is a function of the security capabilities of both computers and the communications between them. One of the fundamental objectives of computer security is data integrity [White 96]. Two aspects of data integrity are improved by public key digital signature techniques. These are sender authentication and data integrity verification. Positive authentication of the message source is provided by the unique relationship of the two encryption keys used in public key cryptography. Positive verification of message integrity is provided by the use of a message authentication code (sometimes called a manipulation detection code or a cryptographic checksum) that is produced by a message digest (sometimes called a data hashing) function. The use of a message authentication code and public key cryptography are combined in the public key digital signature techniques technology.

Sender authentication. Public key cryptography uses two paired keys. These are the public key and the private key (sometimes called the secret key), which are related to each other mathematically. The public key is distributed to anyone that needs to encrypt a message destined for the holder of the private key. The private key is not known to anyone but the holder of the private key. Because of the mathematical relationship of the keys, data encrypted with the public key can only be decrypted with the private key. Another feature of the paired key relationship is that if a message can be successfully decrypted with the public key then it must have been encrypted with the private key. Therefore, any message decrypted by a holder of the public key must have been sent by the holder of the private key. This is used to authenticate the source of a message. Public

key cryptography can use one of several algorithms but the most common one is the Revest, Shamir, and Adleman (RSA) algorithm. It is used to produce the paired keys and to encrypt or decrypt data using the appropriate key.

Data integrity verification. Message digest functions produce a single large number called the message authentication code (MAC) that is *unique*[1] to the total combination and position of characters in the message being digested. The message digest function distributed with RSA is called the MD5 message digest function. It produces a *unique* 128 bit number for each different message digested. If even one character is changed in the message, a dramatically-different 128 bit number is generated.

The overall process for using Public Key Digital Signatures to verify data integrity is shown in Figure 22.

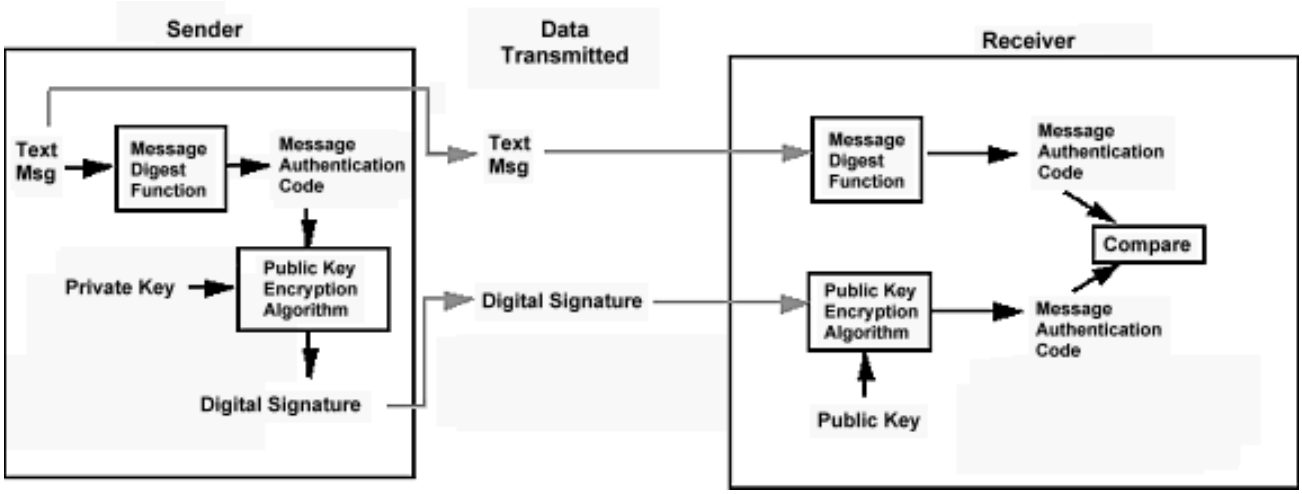


Figure 22: Public Key Digital Signatures

The Digital Signature of a message is produced in two steps:

1. The sender of the message uses the message digest function to produce a message authentication code (MAC).
2. This MAC is then encrypted using the private key and the public key encryption algorithm. This encrypted MAC is attached to the message as the digital signature.

The receiver of the message uses the public key to decrypt the digital signature. If it is decrypted successfully, the receiver of the message knows it came from the holder of the private key. The receiver then uses the message digest function to calculate the MAC associated with the received message contents. If this number compares to the one decrypted from the Digital Signature, the message was received unaltered and data integrity is assured. Together, this technique provides data source authentication and verification of message content integrity.

There are many message digest functions and public key encryption algorithms that may be used in developing the public key digital signature technique. A discussion of these alternative algorithms and their merits is in Schneier [Schneier 96].

Usage Considerations

This technology is most likely to be used in networks of computers where all the communication paths can not be physically protected and where the integrity of data and sender authenticity

aspects of trustability are essential. Military C4I networks and banking networks that are on a widespread local area network or a wide area network are prime examples of this use.

Implementation of the public key digital signature techniques establishes additional requirements on a network. The same message digest functions and public key cryptography algorithm used to process the digital signature must be used by both the sender and receiver. Public/private key pairs must be generated and maintained. Public keys must be distributed (or accessible in a public forum) and private keys protected.

Maturity

The components of this technology, public key encryption and message digest functions, have been in use since the early 1980s. The combined technology is mature and is available in implementations that range from small networks of PCs to protection of data being transferred over the Internet.

The algorithms supporting public key digital signatures have historically consumed large amounts of processing power. However, given recent advances in processors used in PCs and workstations; this is no longer a concern in most circumstances of use.

Costs and Limitations

Using this technology requires network management personnel with knowledge of public key cryptography and the use of software that implements public key cryptography and digital signature algorithms. It also requires security personnel and software that can generate, distribute, and control encryption/decryption keys and respond to the loss or compromise of keys.

Dependencies

Public key cryptography and message digest functions.

Alternatives

Data integrity and authentication can be provided by a combination of dedicated circuits, integrity protocols, and procedural control of sources and destinations. These approaches are not foolproof and can be expensive. Data integrity and authentication can also be provided using private key encryption and a third party arbitrator. This approach has the disadvantage that a third party must be trusted and the data must be encrypted and decrypted twice with two separate private keys.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Public Key Digital Signatures
Application category	System Security (AP.2.4.3)

Quality measures category	Trustworthiness (QM.2.1.4)
Computing reviews category	Computer-Communication Networks Security and Protection (C.2.0) Security and Protection (K.6.5)

References and Information Sources

[Abrams 95]	Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.
[Garfinkel 95]	Garfinkel, Simpson. *PGP: Pretty Good Privacy*. Sebastopol, CA: O'Reilly & Associates, 1995.
[Russel 91]	Russel, Deborah & Gangemi, G.T. Sr. *Computer Security Basics.* Sebastopol, CA: O'Reilly & Associates, Inc., 1991.
[Schneier 96]	Schneier, Bruce. *Applied Cryptography*. New York, NY: John Wiley & Sons, 1996.
[White 96]	White, Gregory B.; Fisch, Eric A.; & Pooch, Udo W. *Computer System and Network Security.* Boca Raton, FL: CRC Press, 1996.

Current Author/Maintainer

Tom Mills, Lockheed Martin

External Reviewers

Jim Ellis, SEI
Scott A. Hissam, SEI

Modifications

4 Nov 03 (typo correction) 26 Jun 00 (references to "secret key" changed to "private key")

10 Jan 97 (original)

Footnotes

[1] Of course they are not absolutely unique. We say unique here because it is extremely unlikely statistically for two files to have the same MAC and, more importantly, it is extremely difficult for an attacker/malicious user to create/craft two files having the same MAC.

URL: http://www.sei.cmu.edu/str/descriptions/pkds_body.html
Last Modified: 24 July 2008

Glossary Term

Dependability

that property of a computer system such that reliance can justifiably be placed on the service it delivers [Barbacci 95].

References and Information Sources

[Serlin 72] Serlin, O. "Scheduling of Time Critical Processes," 925-932. *Proceedings of the Spring Joint Computer Conference*. Atlantic City, NJ, May 16-18, 1972. Montvale, NJ: American Federation of Information Processing Societies, 1972.

References and Information Sources

[Liu 73] Liu, C. L. & Layland, J. W. "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment." *Journal of the Association for Computing Machinery 20*, 1 (January 1973): 40-61.

References and Information Sources

[Sha 91a] Sha, Klein & Goodenough, J. "Rate Monotonic Analysis for Real-Time Systems," 129-155. *Foundations of Real-Time Computing: Scheduling and Resource Management*. Boston, MA: Kluwer Academic Publishers, 1991.

References and Information Sources

[Klein 93] Klein, M.H., et al. *A Practitioners' Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Boston, MA: Kluwer Academic Publishers, 1993.

References and Information Sources

[Rajkumar 91] Rajkumar, Ragunathan. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Boston, MA: Kluwer Academic Publishers, 1991.

References and Information Sources

[Lucas 92] Lucas, L. & Page, B. "Tutorial on Rate Monotonic Analysis." *Ninth Annual Washington Ada Symposium*. McLean, VA, July 13-16, 1992. New York, NY: Association for Computing Machinery, 1992.

References and Information Sources

[Locke 91] Locke, C.D.; Vogel, D.R.; & Mesler, T.J. "Building a Predictable Avionics Platform in Ada: a Case Study," 181-189. *Proceedings of the Twelfth Real-Time Systems Symposium*. San Antonio, TX, December 4-6, 1991. Los Alamitos, CA: IEEE Computer Society Press, 1991.

References and Information Sources

[Ignace 94] Ignace, S. J.; Sedlmeyer, R. L.; & Thuente, D. J. "Integrating Rate Monotonic Analysis into Real-Time Software Development," 257-274. *IFIP Transactions, Diffusion, Transfer and Implementation of Information Technology* (A-45). Pittsburgh, PA, October 11-13, 1993. The Netherlands: International Federation of Information Processing, 1994.

References and Information Sources

[Sha 91b] Sha, L.; Rajkumar, R.; & Lehoczky, J. P. "Real-Time Computing with IEEE Futurebus+." *IEEE Micro 11,* 3 (June 1991): 30-38.

References and Information Sources

[Fowler 93] Fowler, P. & Levine, L. *Technology Transition Push: A Case Study of Rate Monotonic Analysis* Part 1 (CMU/SEI-93-TR-29). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

Related Topics

System Analysis and Optimization (AP.1.3.6)

- Model Checking
- Rate Monotonic Analysis
- Software Reliability Modeling and Analysis

Reference Models, Architectures, Implementations--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

Much confusion exists regarding the definition, applicability, and scope of the terms *reference model*, *architecture*, and *implementation*. Understanding these terms facilitates understanding legacy system designs and how to migrate them to more open systems. The purpose of this technology description is to provide definitions, and more importantly, to describe how the terms are related.

Technical Detail

Reference model. A reference model is a description of all of the possible software components, component services (functions), and the relationships between them (how these components are put together and how they will interact). Examples of commonly-known reference models include the following:

- the Technical Architecture for Information Management (TAFIM) reference model (see TAFIM Reference Model)
- the Reference Model for Frameworks of Software Engineering Environments [ECMA 93]
- Project Support Environment Reference Model (PSERM)
- the Tri-Service Working Group Open Systems Reference Model

Architecture. An architecture is a description of a subset of the reference model's component services that have been selected to meet a specific system's requirements. In other words, not all of the reference model's component services need to be included in a specific architecture. There can be many architectures derived from the same reference model. The associated standards and guidelines for each service included in the architecture form the open systems architecture and become the criteria for implementing the system.

Implementation. The implementation is a product that results from selecting (e.g., commercial-off-the-shelf), reusing, building and integrating software components and component services according to the specified architecture. The selected, reused, and/or built components and component services must comply 100% with the associated standards and guidelines for the implementation to be considered compliant.

Usage Considerations

Figure 23 attempts to show the interrelationships of these concepts using the TAFIM as an example. TAFIM provides the reference model and a number of specific architectures can be derived from the TAFIM reference model based on specific program requirements. From there a number of

implementations may be developed based on the products selected to meet the architecture's services, so long as these products meet the required standards and guidelines. For instance, in one implementation, the product ORACLE might be selected and used to meet some of the data management services. In another implementation, the product Sybase might be selected and used.

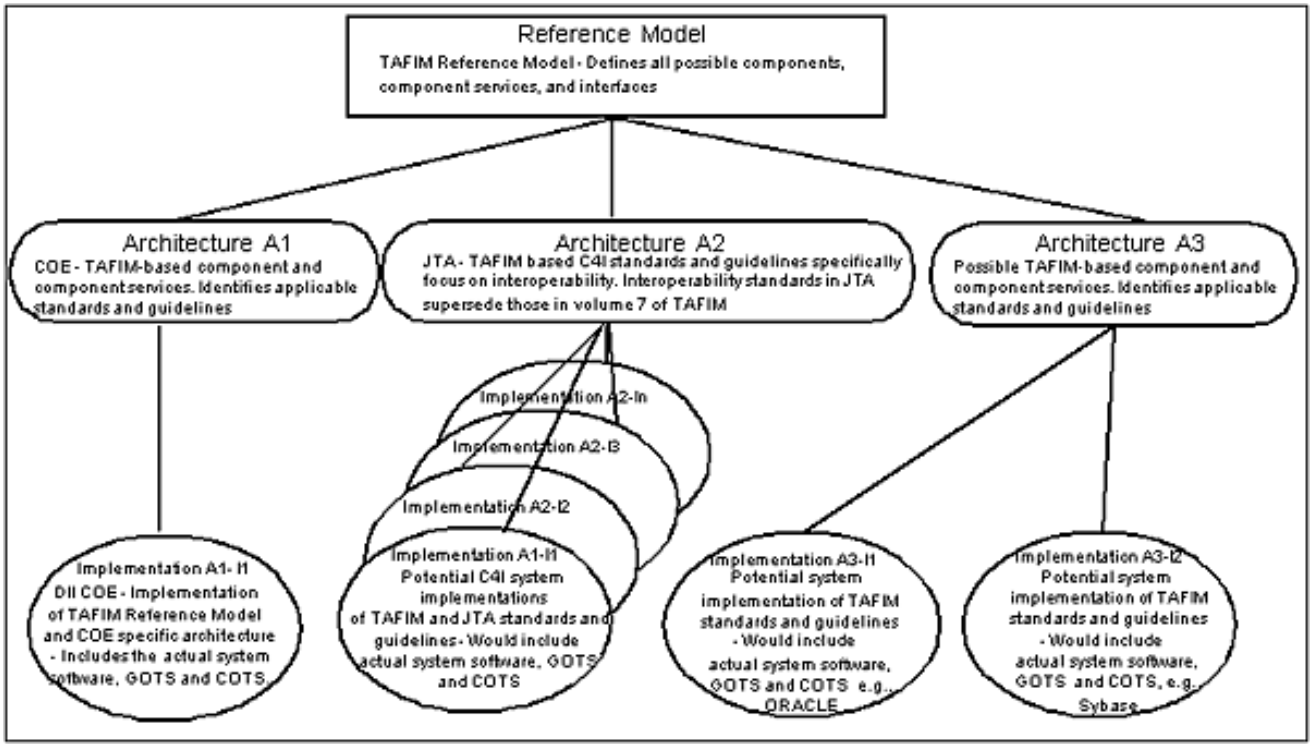


Figure 23: Reference Model, Architecture, and Implementation

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Reference Models, Architectures, Implementations - An Overview
Application category	Software Architecture Models (AP.2.1.1) Software Architecture (AP.2.1)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1) Portability (QM.4.2)
Computing reviews category	Distributed Systems (C.2.4) Software Engineering Design (D.2.10)

References and Information Sources

[ECMA 93] *Reference Model for Frameworks of Software Engineering Environments*, 3rd Edition (NIST Special Publication 500-211/Technical Report ECMA TR/55). Prepared jointly by NIST and the European Computer Manufacturers Association (ECMA). Washington, DC: U.S. Government Printing Office, 1993.

[Meyers 96] Meyers, Craig & Oberndorf, Tricia. *Open Systems: The Promises and the Pitfalls*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Tricia Oberndorf, SEI

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/refmodels_body.html
Last Modified: 24 July 2008

References and Information Sources

[Birrell 84] Birrell, A.D. & Nelson, B.J. "Implementing Remote Procedure Calls." *ACM Transactions on Computer Systems 2,* 1 (February 1984): 39-59.

Remote Procedure Call

Software Technology Roadmap

Status

Advanced

Note

We recommend Middleware *as prerequisite reading for this technology description.*

Purpose and Origin

Remote Procedure Call (RPC) is a client/server infrastructure that increases the *interoperability*, *portability,* and *flexibility* of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the *complexity* of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces--function calls are the programmer's interface when using RPC [Rao 1995].

The concept of RPC has been discussed in literature as far back as 1976, with full-scale implementations appearing in the late 1970s and early 1980s [Birrell 84].

Technical Detail

In order to access the remote server portion of an application, special function calls, RPCs, are embedded within the client portion of the client/server application program. Because they are embedded, RPCs do not stand alone as a discreet middleware layer. When the client program is compiled, the compiler creates a local stub for the client portion and another stub for the server portion of the application. These stubs are invoked when the application requires a remote function and typically support synchronous calls between clients and servers. These relationships are shown in Figure 32 [Steinke 95].

By using RPC, the complexity involved in the development of distributed processing is reduced by keeping the semantics of a remote call the same whether or not the client and server are collocated on the same system. However, RPC increases the involvement of an application developer with the

complexity of the master-slave nature of the client/server mechanism.

RPC increases the flexibility of an architecture by allowing a client component of an application to employ a function call to access a server on a remote system. RPC allows the remote component to be accessed without knowledge of the network address or any other lower-level information. Most RPCs use a synchronous, request-reply (sometimes referred to as "call/wait") protocol which involves blocking of the client until the server fulfills its request. Asynchronous ("call/nowait") implementations are available but are currently the exception.

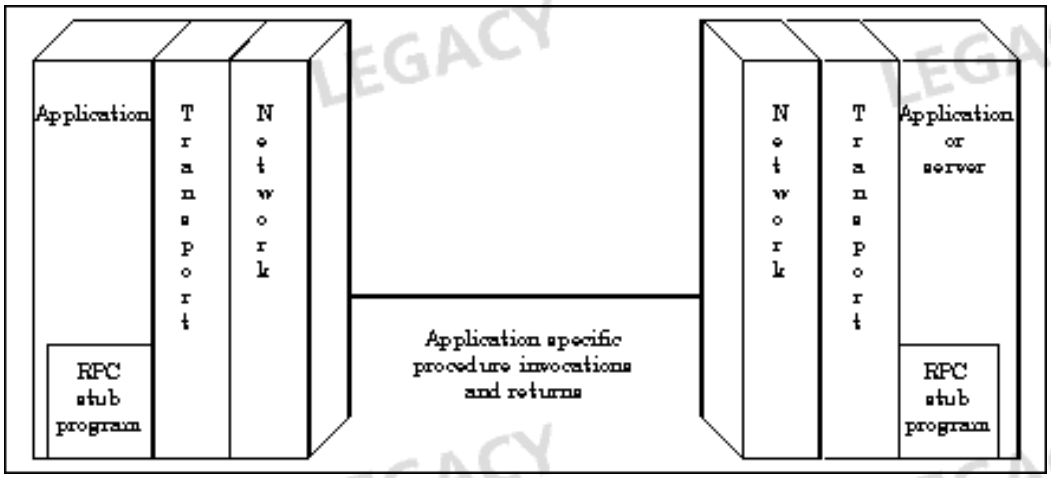


Figure 32: Remote Procedure Calls

RPC is typically implemented in one of two ways:

1. within a broader, more encompassing propriety product
2. by a programmer using a proprietary tool to create client/server RPC stubs

Usage Considerations

RPC is appropriate for client/server applications in which the client can issue a request and wait for the server's response before continuing its own processing. Because most RPC implementations do not support peer-to-peer, or asynchronous, client/server interaction, RPC is not well-suited for applications involving distributed objects or object-oriented programming (see Object-Oriented Programming Languages).

Asynchronous and synchronous mechanisms each have strengths and weaknesses that should be considered when designing any specific application. In contrast to asynchronous mechanisms employed by Message-Oriented Middleware, the use of a synchronous request-reply mechanism in RPC requires that the client and server are always available and functioning (i.e., the client or server is not blocked). In order to allow a client/server application to recover from a blocked condition, an implementation of a RPC is required to provide mechanisms such as error messages, request timers, retransmissions, or redirection to an alternate server. The complexity of the application using a RPC

is dependent on the sophistication of the specific RPC implementation (i.e., the more sophisticated the recovery mechanisms supported by RPC, the less complex the application utilizing the RPC is required to be). RPCs that implement asynchronous mechanisms are very few and are difficult (complex) to implement [Rao 1995].

When utilizing RPC over a distributed network, the performance (or load) of the network should be considered. One of the strengths of RPC is that the synchronous, blocking mechanism of RPC guards against overloading a network, unlike the asynchronous mechanism of Message-Oriented Middleware (MOM). However, when recovery mechanisms, such as retransmissions, are employed by an RPC application, the resulting load on a network may increase, making the application inappropriate for a congested network. Also, because RPC uses static routing tables established at compile-time, the ability to perform load balancing across a network is difficult and should be considered when designing an RPC-based application.

Maturity

Tools are available for a programmer to use in developing RPC applications over a wide variety of platforms, including Windows (3.1, NT, 95), Macintosh, 26 variants of UNIX, OS/2, NetWare, and VMS [Steinke 1995]. RPC infrastructures are implemented within the Distributed Computing Environment (DCE) , and within Open Network Computing (ONC), developed by Sunsoft, Inc. These two RPC implementations dominate the current Middleware market [Rao 1995].

Costs and Limitations

RPC implementations are nominally incompatible with other RPC implementations, although some are compatible. Using a single implementation of a RPC in a system will most likely result in a dependence on the RPC vendor for maintenance support and future enhancements. This could have a highly negative impact on a system's flexibility, maintainability, portability, and interoperability.

Because there is no single standard for implementing an RPC, different features may be offered by individual RPC implementations. Features that may affect the design and cost of a RPC-based application include the following:

- support of synchronous and/or asynchronous processing
- support of different networking protocols
- support for different file systems
- whether the RPC mechanism can be obtained individually, or only bundled with a server operating system

Because of the complexity of the synchronous mechanism of RPC and the proprietary and unique nature of RPC implementations, training is essential even for the experienced programmer.

Alternatives

Other middleware technologies that allow the distribution of processing across multiple processors and platforms are

- Object Request Brokers (ORB)
- Distributed Computing Environment (DCE)
- Message-Oriented Middleware (MOM)
- COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities)
- Transaction Processing Monitor Technology
- Three Tier Software Architectures

Complementary Technologies

RPC can be effectively combined with Message-Oriented Middleware (MOM)- MOM can be used for asynchronous processing.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Remote Procedure Call
Application category	Client/Server (AP.2.1.2.1) Client/Server Communication (AP.2.2.1)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1) Portability (QM.4.2) Complexity (QM.3.2.1)
Computing reviews category	Distributed Systems (C.2.4)

References and Information Sources

[Birrell 84] Birrell, A.D. & Nelson, B.J. "Implementing Remote Procedure Calls." *ACM Transactions on Computer Systems 2,* 1 (February 1984): 39-59.

[Rao 95] Rao, B.R. "Making the Most of Middleware." *Data Communications International 24,* 12 (September 1995): 89-96.

[Steinke 95] Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

[Thekkath 93] Thekkath, C.A. & Levy, H.M. "Limits to Low-Latency Communication on High-Speed Networks." *ACM Transactions on Computer Systems 11*, 2 (May 1993): 179-203.

Current Author/Maintainer

Cory Vondrak, TRW, Redondo Beach, CA

Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/rpc_body.html
Last Modified: 24 July 2008

Glossary Term

Completeness

the degree to which all the parts of a software system or component are present and each of its parts is fully specified and developed [Boehm 78].

Glossary Term

Traceability

the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another [IEEE 90].

References and Information Sources

[SPS 94] *Analysis of Automated Requirements Management Capabilities*. Melbourne, FL: Software Productivity Solutions, 1994.

References and Information Sources

[Gotel 95] Gotel, Orlena. *Contribution Structures for Requirements Traceability*. London, England: Imperial College, Department of Computing, 1995.

Requirements Tracing--An Overview

Software Technology Roadmap

Status

Advanced

Purpose and Origin

The development and use of requirements tracing techniques originated in the early 1970s to influence the *completeness*, *consistency*, and *traceability* of the requirements of a system. They provide an answer to the following questions:

- What mission need is addressed by a requirement?
- Where is a requirement implemented?
- Is this requirement necessary?
- How do I interpret this requirement?
- What design decisions affect the implementation of a requirement?
- Are all requirements allocated?
- Why is the design implemented this way and what were the other alternatives?
- Is this design element necessary?
- Is the implementation compliant with the requirements?
- What acceptance test will be used to verify a requirement?
- Are we done?
- What is the impact of changing a requirement [SPS 94]?

The purpose of this technology description is to introduce the key concepts of requirements tracing. Detailed discussions of the individual technologies can be found in the referenced technology descriptions.

Technical Detail

Requirements traceability is defined as the ability to describe and follow the life of a requirement, in both a forward and backward direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases) [Gotel 95]. It can be achieved by using one or more of the following techniques:

- *Cross referencing*. This involves embedding phrases like "see section x" throughout the project documentation (e.g., tagging, numbering, or

indexing of requirements, and specialized tables or matrices that track the cross references).

- *Specialized templates and integration or transformation documents.* These are used to store links between documents created in different phases of development.
- *Restructuring.* The documentation is restructured in terms of an underlying network or graph to keep track of requirements changes (e.g., assumption-based truth maintenance networks, chaining mechanisms, constraint networks, and propagation) [Gotel 95].

Usage Considerations

For any given project, a key milestone (or step) is to determine and agree upon requirements traceability details. Initially, three important questions need to be answered before embarking on any particular requirements traceability approach:

1. What needs to be traceable?
2. What linkages need to be made?
3. How, when, and who should establish and maintain the resulting database?

Once the questions are answered, then selection of an approach can be made. One approach could be the structured use of general-purpose tools (e.g., hypertext editors, word processors, and spreadsheets) configured to support cross-referencing between documents. For large software development projects, an alternative approach could be the use of a dedicated workbench centered around a database management system providing tools for documenting, parsing, editing, decomposing, grouping, linking, organizing, partitioning, and managing requirements. Table 9 describes the strengths and weaknesses of each of the approaches.

Table 9: Comparing Requirements Tracing Approaches

Approaches	Strengths	Weaknesses
General purpose tools	· readily available · flexible · good for small projects	· need to be configured to support Requirements Traceability (RT) · potential high RT maintenance cost · limited control over RT information · potential limited integration with other software development tools

Workbenches	· fine-grained forward, backward, horizontal, and vertical RT · RT results may facilitate later development activities (i.e., testing) · suitable for large projects	· depend upon stakeholder buy-in · manual intervention may be required · RT in later development phases may be difficult

Regardless of the approach taken, requirements tracing requires a combination of models (i.e., representation forms), methods (i.e., step by step processes), and/or languages (i.e., semiformal and formal) that incorporate the above techniques. Some examples of requirements tracing methods are discussed in the following technology descriptions:

- Feature-Based Design Rationale Capture Method for Requirements Tracing
- Argument-Based Design Rationale Capture Methods for Requirements Tracing

Maturity

Every major office tool manufacturer has spreadsheet and/or database capabilities that can be configured to support requirements tracing. There are at least ten commercial products that fall in the workbench category and support some level of requirements traceability [STSC 98]. At a minimum, they provide

- bidirectional requirement linking to system elements
- capture of allocation rationale, accountability, and test/validation
- identification of inconsistencies
- capabilities to view/trace links
- verification of requirements
- history of requirements changes.

Environments to support requirements traceability past the requirements engineering phase of the system/software life cycle are being researched. Areas include the development of a common language, method, model, and database repository structure, as well as mechanisms to provide data exchange between different tools in the environment. Prototypes exist and at least one commercial product provides support for data exchange through its object-oriented database facilities.

Costs and Limitations

In general, the implementation of requirements tracing techniques within an organization should facilitate reuse and maintainability of the system. However,

additional resources (time and manpower) to initially implement traceability processes (i.e., definition of traceability information, selection of automated tools, training, etc.) will be required. One case study found that the cost was more than twice the normal documentation cost associated with the development of a system of similar size and complexity. However, this was determined to be a one-time cost and the overall costs to maintain the software system are expected to be reduced. Almost immediate return was observed in the reduced amount of time to perform hardware upgrades [Ramesh 95].

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Requirements Tracing
Application category	Requirements Tracing (AP.1.2.3)
Quality measures category	Completeness (QM.1.3.1) Consistency (QM.1.3.2) Traceability (QM.1.3.3) Effectiveness (QM.1.1) Reusability (QM.4.4) Understandability (QM.3.2) Maintainability (QM.3.1)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2) Software Engineering Requirements/ Specifications (D.2.1)

References and Information Sources

[Bailin 90] Bailin, S., et al. "KAPTUR: Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationale," 95-104. *Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference*. Liverpool, NY, September 24-28, 1990. Rome, NY: Rome Air Development Center, 1990.

[Gotel 95] Gotel, Orlena. *Contribution Structures for Requirements Traceability*. London, England: Imperial College, Department of Computing, 1995.

[Ramesh 92] Ramesh, Balasubramaniam & Dhar, Vasant. "Supporting Systems Development by Capturing Deliberations During Requirements Engineering." *IEEE Transactions on Software Engineering 18,* 6 (June 1992): 498-510.

[Ramesh 95] Ramesh, Bala; Stubbs, Lt Curtis; & Edwards, Michael. "Lessons Learned from Implementing Requirements Traceability." *Crosstalk, Journal of Defense Software Engineering 8,* 4 (April 1995): 11-15.

[Shum 94] Shum, Buckingham Simon & Hammond, Nick. "Argumentation-Based Design Rationale: What Use at What Cost?" *International Journal of Human-Computer Studies 40,* 4 (April 1994): 603-652.

[SPS 94] *Analysis of Automated Requirements Management Capabilities.* Melbourne, FL: Software Productivity Solutions, 1994.

[STSC 98] Software Technology Support Center. *Requirements Management Tools* [online]. Available WWW <URL:http://www.stsc.hill.af.mil/RED/LIST.HTML> (1998).

Current Author/Maintainer

Liz Kean, Air Force Rome Laboratory

External Reviewers

Brian Gallagher, SEI

Modifications

4 Feb 98: added reference for [STSC 98]
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/reqtracing_body.html
Last Modified: 24 July 2008

References and Information Sources

[STSC 98] Software Technology Support Center. *Requirements Management Tools* [online]. Available WWW <URL:http://www.stsc.hill.af.mil/RED/LIST.HTML> (1998).

References and Information Sources

[Ramesh 95] Ramesh, Bala; Stubbs, Lt Curtis; & Edwards, Michael. "Lessons Learned from Implementing Requirements Traceability." *Crosstalk, Journal of Defense Software Engineering 8*, 4 (April 1995): 11-15.

Public Key Cryptography

Software Technology Roadmap

Status

Draft

Purpose and Origin

Cryptography is an algorithmic process of converting a plain text (or clear text) message to a cipher text (or cipher) message based on an algorithm that both the sender and receiver know, so that the cipher text message can be returned to its original, plain text form. In its cipher form, a message cannot be read by anyone but the intended receiver. The act of converting a plain text message to its cipher text form is called enciphering. Reversing that act (i.e., cipher text form to plain text message) is deciphering. Enciphering and deciphering are more commonly referred to as *encryption* and *decryption*, respectively.

There are a number of algorithms for performing encryption and decryption, but comparatively few such algorithms have stood the test of time. The most successful algorithms use a *key*. A key is simply a parameter to the algorithm that allows the encryption and decryption process to occur. There are many modern key-based cryptographic techniques [Schneier 96]. These are divided into two classes: symmetric and asymmetric (also called public/private) key cryptography. In *symmetric key cryptography*, the same key is used for both encryption and decryption. In *asymmetric key cryptography*, one key is used for encryption and another, mathematically related key, is used for decryption.

Symmetric Key Cryptography

The most widely used symmetric key cryptographic method is the Data Encryption Standard (DES) [NIST 93]. Although originally published in 1977 by the National Bureau of Standards (reprinted in [Beker+ 82]), DES has not yet been replaced by any other symmetric-key approach. DES uses a fixed length, 56-bit key and an efficient algorithm to quickly encrypt and decrypt messages. DES can be easily implemented in hardware, making the encryption and decryption process even faster. In general, increasing the key size makes the system more secure. A variation of DES, called Triple-DES or DES-EDE (encrypt-decrypt-encrypt), uses three applications of DES and two independent DES keys to produce an effective key length of 168 bits [ANSI 85].

The International Data Encryption Algorithm (IDEA) was invented by James Massey and Xuejia Lai of ETH Zurich, Switzerland in 1991 and is patented and

registered by the Swiss Ascom Tech AG, Solothurn [Lai 92]. IDEA uses a fixed length, 128-bit key (larger than DES but smaller than Triple-DES). It is also faster than Triple-DES. In the early 1990s, Don Rivest of RSA Data Security, Inc., invented the algorithms RC2 and RC4. These use variable length keys and are claimed to be even faster than IDEA. However, implementations may be exported from the U.S. only if they use key lengths of 40 bits or fewer.

Although symmetric key cryptography works, it has a fundamental weak spot-key management. Since the same key is used for encryption and decryption, it must be kept secure. If an adversary knows the key, then the message can be decrypted. At the same time, the key must be available to the sender and the receiver and these two parties may be physically separated. Symmetric key cryptography transforms the problem of transmitting messages securely into that of transmitting keys securely. This is a step forward, because keys are much smaller than messages, and the keys can be generated beforehand. Nevertheless, ensuring that the sender and receiver are using the same key and that potential adversaries do not know this key remains a major stumbling block. This is referred to as the key management problem.

Public/Private Key Cryptography

Asymmetric key cryptography overcomes the key management problem by using different encryption and decryption key pairs. Having knowledge of one key, say the encryption key, is not sufficient enough to determine the other key - the decryption key. Therefore, the encryption key can be made public, provided the decryption key is held only by the party wishing to receive encrypted messages (hence the name public/private key cryptography). Anyone can use the public key to encrypt a message, but only the recipient can decrypt it.

James Ellis, Malcolm Williamson, and Clifford Cocks first investigated public/ private key cryptography at the British Government Communications Headquarters (GCHQ) in the early 1970s [Ellis 87]. The first public discussion of public/private key cryptography was by Whitfield Diffie and Martin Hellman in 1976 [Diffie+ 76].

A widely used public/private key algorithm is RSA, named after the initials of its inventors, Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman [RSA 91]. RSA depends on the difficulty of factoring the product of two very large prime numbers. Although used for encrypting whole messages, RSA is much less efficient than symmetric key algorithms such as DES. ElGamal is another public/ private key algorithm [El Gamal 85]. It uses a different arithmetic algorithm than RSA, called the discrete logarithm problem. An extensive discussion of public/ private key cryptography, including much of the mathematical detail, can be found in the book, *Public Key Cryptography* [Salomaa 96].

Technical Detail

The mathematical relationship between the public/private key pair permits a general rule: any message encrypted with one key of the pair can be successfully decrypted only with that key's counterpart. To encrypt with the public key means you can decrypt only with the private key. The converse is also

true - to encrypt with the private key means you can decrypt only with the public key.

The decision as to which key is kept private and which is made public is not arbitrary. In the case of RSA, the public key uses exponents that are relatively small (in comparison to the private key) making the process of encryption and digital signature verification (discussed later) faster.

Figure 1 illustrates the proper and intended used of public/private key cryptography for sending confidential messages. In the illustration, a user, Bob, has a public/private key pair. The public portion of that key pair is placed in the public domain (for example in a Web server). The private portion is guarded in a private domain, for example, on a digital key card or in a password-protected file.

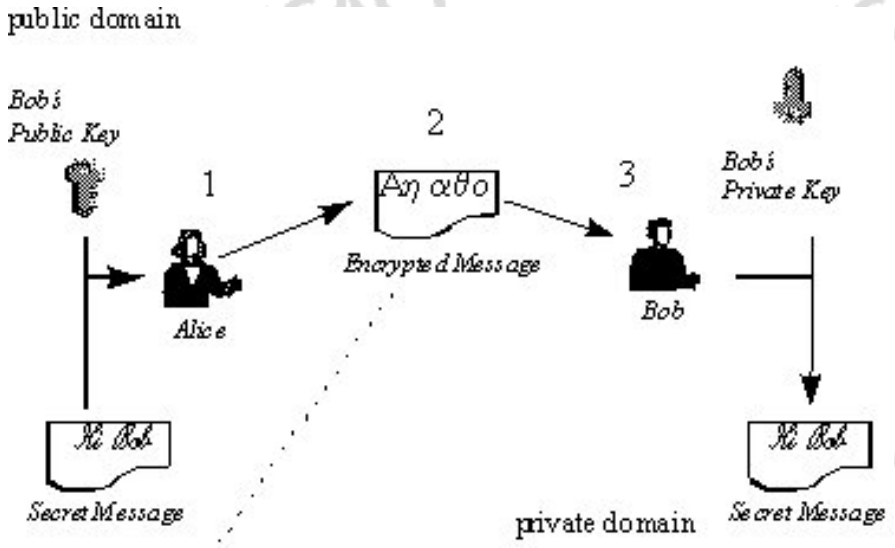


Figure 1: Proper Use of Public Key Cryptography

For Alice to send a secret message to Bob, the following process needs to be followed:

1. Alice passes the secret message and Bob's public key to the appropriate encryption algorithm to construct the encrypted message.
2. Alice transmits the encrypted message (perhaps via e-mail) to Bob.
3. Bob decrypts the transmitted, encrypted message with his private key and the appropriate decryption algorithm.

Bob can be assured that Alice's encrypted secret message was not seen by anyone else since only his private key is capable of decrypting the message.

Since we know that a private key can also be used to encrypt messages, Bob could technically respond in secret to Alice's original message by using the same public/private key pair as illustrated in Figure 2.

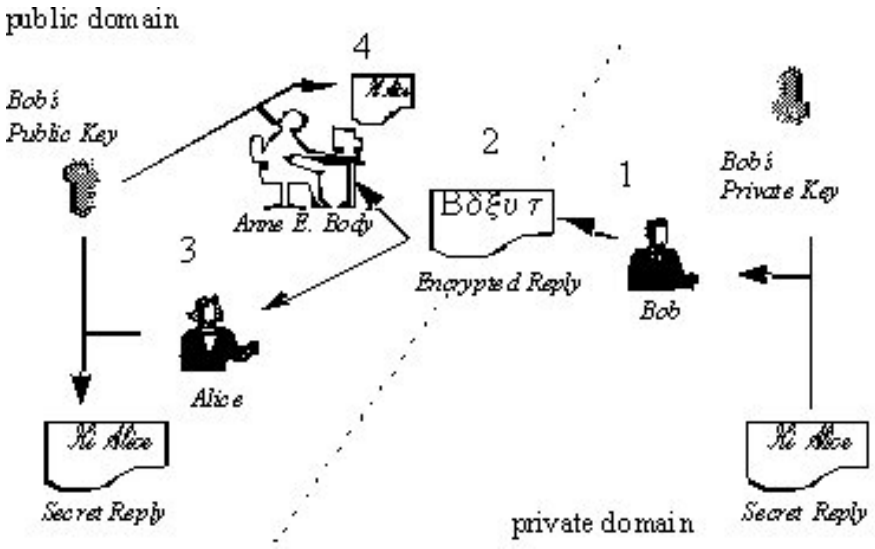


Figure 2: Improper Use of Public Key Cryptography

In this scenario:

1. Bob passes the secret reply and his private key to the encryption algorithm to construct the encrypted reply.
2. Bob transmits the encrypted reply to Alice.
3. Alice decrypts the transmitted, encrypted reply with Bob's public key and the decryption algorithm to read this reply.

Unfortunately, Bob's message will not be confidential because anyone with access to the encrypted reply and Bob's public key (which is in the public domain) can decrypt the reply and see the text of the message. However, if Alice had her own public/private key pair, then Bob and Alice could communicate confidentially. In this case, Bob would send messages encrypted with Alice's public key (which only Alice could decrypt by using her private key), and Alice would send messages to Bob encrypted with Bob's public key (which only he could decrypt using his private key).

Usage Considerations

Public key cryptography is especially useful in situations where there is a need for confidentiality, integrity, and non-repudiation. That is, in situations where the messages being passed are intended to only be shared by the sending and receiving parties. Further, public key cryptography is used in situations where the recipient of a message must have confidence that the message received was received as intended by the sender and has not been altered or forged in any manner.

Confidentiality assures that unintended third parties can not view information sent between two communicating parties. Encryption is the most widely used mechanism for providing confidentiality over an insecure medium.

Integrity is knowing that the message you receive was exactly what was sent and it was unaltered or damaged during transmission. Digital signatures are

used to seal a message as a means to warn if the integrity of a message has been compromised. Today, Web content that executes on local workstations is commonly downloaded. Knowing that the content has not been surreptitiously modified is critical if you are to trust the content. If the content is from a trusted source and it is unmodified, your confidence in that content is higher - because the content has integrity. If the content is from an unknown source or you cannot tell if it has been modified, the content cannot be trusted. Mechanisms such as digital signatures and certificates help maintain the integrity of exchanged products and services.

Non-repudiation is the inability to disavow an act. In other words, evidence exists that prevents a person from denying an act. For example, you log in to a computer system by presenting a user name and password. Most software applications consider this sufficient evidence to permit access, but could it be proved that it was really you that was logged in? You could argue that someone else obtained your password, possibly using snooping techniques. Now, suppose that a computer system requires a fingerprint or retinal image to gain access. Contesting the fact now becomes more difficult.

Finally, as opposed to symmetric key cryptography, public key cryptography is a useful means of getting around issues dealing with key distribution and management.

Maturity

Public key cryptography has been in use for more than 30 years. Secure Sockets Layer (SSL) defined by Netscape is a popular application of public key cryptography found in Web-enabled applications requiring secure communications and authentication. Pretty Good Privacy (or PGP) is another popular application of public key cryptography used to send confidential electronic mail and digitally signing electronic documents.

Further, a number of commercial companies have become third party providers of public key cryptography software including, but not limited to, RSA Security, Inc, Sun Microsystems, Microsoft, Entrust, Inc., and VeriSign, Inc.

Costs and Limitations

Cost to implement public key cryptography in a system vary according to size and scope. Characteristics that can determine costs include the number of pair-wise keys that need to be created for the purposes of confidentiality and integrity. For example, securing all corporate email will require that employers to issue public keys to all of its employees and enforce the use of those key when communicating corporate ideas and correspondence. Systems are available to support such wide use but come at a cost. A counter-example of this would be a corporate "portal" or web site available to the public from which the public may be asked to place orders. In such a case, the corporation may only be required to acquire public key cryptography for the one or more server(s) that will be used to interact with the public, this is typically a annual cost from security providers such as VeriSign, Inc.

Using this technology may require network management personnel with knowledge of public key cryptography and the use of software that implements public key cryptography and digital signature algorithms especially if an outside provider for public key infrastructures is NOT used. It also requires security personnel and software that can generate, distribute, and control encryption/ decryption keys and respond to the loss or compromise of keys.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Public Key Cryptography
Application category	Information Security (AP.2.4)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating Systems Security & Protection (D.4.6), Security & Protection (K.6.5), Computer-Communications Networks Security and Protection (C.2.0)

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Schneier 96] Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd editon by , John Wiley & Sons, ISBN 0471128457, 1996.

[NIST 93] Data Encryption Standard (DES) (FIPS PUB 46-2). Gaithersburg, Md.: National Institute of Standards and Technology, January, 1993. Available WWW: <URL: http://www.nist.gov/itl/div897/pubs/fip46-2.htm>.

[Beker+ 82] Beker, H. & Piper, F. Cipher Systems. London: Northwood Books, 1982.

[ANSI 85] ANSI X9.17-1985, American National Standard, Financial Institution Key Management (Wholesale), American Bankers Association, Section 7.2. New York: American National Standards Institute, 1985.

[Lai 92] Lai, X. ETH Series on Information Processing (J.L. Massey, ed.). Vol. 1, On the Design and Security of Block Ciphers. Konstanz, Switzerland: Hartung-Gorre Verlag, 1992.

[Ellis 87] Ellis, J.H. ìThe Story of Non-Secret Encryption.î Cheltenham, UK: Communications Electronics Security Group, 1987. Available WWW: <URL: http://www.cesg.gov.uk/about/nsecret/ellis.htm>

[Diffie+] Diffie, W. & Hellman, M.E. ìNew Directions in Cryptography.î IEEE Transactions on Information Theory, IT-22, Vol. 6, pp. 644-654, 1976.

[RSA 91] PKCS #1: RSA Encryption Standard, Version 1.4. San Mateo, Ca.: RSA Data Security, Inc., 1991.

[El Gamal 85] El Gamal, T. ìA Public Key Cryptosystem and Signature Scheme Based on Discrete Logarithms.î IEEE Transactions on Information Theory, IT-31, pp. 469-473, 1985.

[Salomaa 96] Salomaa, A. Public-Key Cryptography, 2nd edition. Berlin: Springer-Verlag, 1996.

Current Author/Maintainer

Scott A. Hissam, SEI

External Reviewers

Modifications

9 Dec 01 (original)

URL: http://www.sei.cmu.edu/str/descriptions/publickey_body.html
Last Modified: 24 July 2008

Notes

[1] An enterprise is defined as a system comprised of multiple business systems or multiple subsystems.

References and Information Sources

[Shelton 93] Shelton, Robert E. "The Distributed Enterprise (Shared, Reusable Business Models the Next Step in Distributed Object Computing)." *Distributed Computing Monitor 8*, 10 (October 1993): 1.

References and Information Sources

[Adler 95] Adler, R. M. "Distributed Coordination Models for Client/Sever Computing." *Computer 28,* 4 (April 1995): 14-22.

References and Information Sources

[OSF 96a] Open Software Foundation. *The OSF Distributed Computing Environment* [online]. Available WWW <URL: http://www.osf.org/dce/> (1996).

References and Information Sources

[Schill 93] Schill, Alexander. "DCE-The OSF Distributed Computing Environment Client/Server Model and Beyond," 283. *International DCE Workshop*. Karlsruhe, Germany, October 7-8, 1993. Berlin, Germany: Springer-Verlag, 1993.

Distributed Computing Environment

Software Technology Roadmap

Status

Advanced

Note

We recommend Middleware *as prerequisite reading for this technology description.*

Purpose and Origin

Developed and maintained by the Open Systems Foundation (OSF), the Distributed Computing Environment (DCE) is an integrated distributed environment which incorporates technology from industry. The DCE is a set of integrated system services that provide an *interoperable* and *flexible* distributed environment with the primary goal of solving interoperability problems in heterogeneous, networked environments.

OSF provides a reference implementation (source code) on which all DCE products are based [OSF 96a].The DCE is *portable* and flexible- the reference implementation is independent of both networks and operating systems and provides an architecture in which new technologies can be included, thus allowing for future enhancements. The intent of the DCE is that the reference implementation will include mature, proven technology that can be used in parts- individual services- or as a complete integrated infrastructure.

The DCE infrastructure supports the construction and integration of client/server applications while attempting to hide the inherent *complexity* of the distributed processing from the user [Schill 93]. The OSF DCE is intended to form a comprehensive software platform on which distributed applications can be built, executed, and maintained.

Technical Detail

The DCE architecture is shown in Figure 10 [Schill 93].

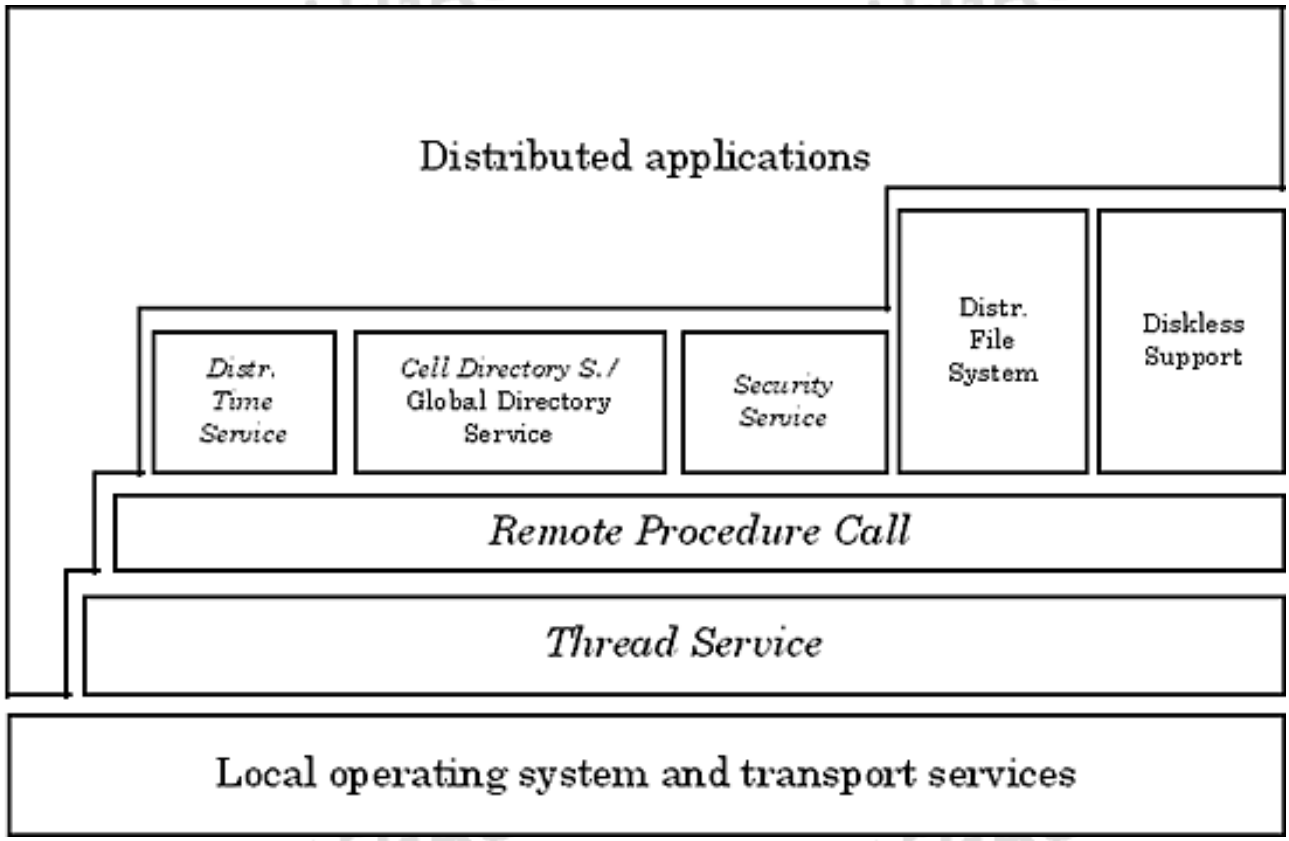


Figure 10: Distributed Computing Environment Architecture

DCE services are organized into two categories:

1. *Fundamental distributed services* provide tools for software developers to create the end-user services needed for distributed computing. They include
    - Remote Procedure Call, which provides portability, network *independence*, and *secure* distributed applications.
    - Directory services, which provide full X.500 support and a single naming model to allow programmers and maintainers to identify and access distributed resources more easily.
    - Time service, which provides a mechanism to monitor and track clocks in a distributed environment and accurate time stamps to reduce the load on system administrator.
    - Security service, which provides the network with authentication, authorization, and user account management services to maintain the *integrity*, privacy, and authenticity of the distributed system.
    - Thread service, which provides a simple, portable, programming model for building concurrent applications.
2. *Data-sharing services* provide end users with capabilities built upon the fundamental distributed services. These services require no programming on the part of the end user and facilitate better use of information. They include
    - Distributed file system, which interoperates with the network file system to provide a high-performance, *scalable*, and secure file access system.
    - Diskless support, which allows low-cost workstations to use disks on servers, possibly reducing the need/cost for local disks, and provides performance enhancements to reduce network overhead.

The DCE supports International Open Systems Interconnect (OSI) standards, which are critical to global interconnectivity. It also implements ISO standards such as CCITT X.500, Remote Operations Service Element (ROSE), Association Control Service Element (ACSE), and the ISO session and presentation services. The DCE also supports Internet standards such as the TCP/IP transport and network protocols, as well as the Domain Name System and Network Time Protocol provided by the Internet.

Usage Considerations

The DCE can be used by system vendors, software developers, and end users. It can be used on any network hardware and transport software, including TCP/IP, OSI, and X.25. The DCE is written in standard C and uses standard operating system service interfaces like POSIX and X/Open guidelines. This makes the DCE portable to a wide variety of platforms. DCE allows for the extension of a network to large numbers of nodes, providing an environment capable of supporting networks of numerous low-end computers (i.e., PCs and Macintosh machines), which is important if downsizing and distributing of processing is desired. Because DCE is provided in source form, it can be tailored for specific applications if desired [OSF 96a].

DCE works internally with the client/server model and is well-suited for development of applications that are structured according to this model. Most DCE services are especially optimized for a structuring of distributed computing systems into a "cell" (a set of nodes/platforms) that is managed together by one authority.

For DCE, intra-cell communication is optimized and relatively secure and transparent. Inter-cell communication, however, requires more specialized processing and more complexity than its intra-cell counterpart, and requires a greater degree of programming expertise.

When using the thread services provided by DCE, the application programmer must be aware of thread synchronization and shared data across threads. While different threads are mutually asynchronous up to a static number defined at initialization, an individual thread is synchronous. The complexity of thread programming should be considered if these services are to be used.

DCE is being used or is planned for use on a wide variety of applications, including the following:

- *The Common Operating Environment*. DCE has been approved by DISA (Defense Information Systems Agency) as the distributed computing technology for the Common Operating Environment (COE) (see Defense Information Infrastructure Common Operating Environment).
- *The Advanced Photon Source (APS) system*. This is a synchrotron radiation facility under construction at Argonne National Laboratory.
- *The Alaska Synthetic Aperture Radar Facility (ASF)*. This is the ground station for a set of earth-observing radar spacecraft, and is one of the first NASA projects to use DCE in an operational system.
- *The Deep Space Network's Communications Complexes Monitor and Control Subsystem*. This project is deploying DCE for subsystem internal communications, with the expectation that DCE will eventually form the infrastructure of the entire information system.
- *The Multimission Ground Data System Prototype*. This project evaluated the applicability of DCE technology to ground data systems for support of JPL flight projects

(Voyager, Cassini, Mars Global Surveyor, Mars Pathfinder).
- *Earth Observing Systems Data Information System*. This NASA system is one of the largest information systems ever implemented. The system is comprised of legacy systems and data, computers of many varieties, networks, and satellites in space.
- *Command and control prototypes*. MITRE has prototyped command and control (C2) applications using DCE technology. These applications provide critical data such as unit strength, supplies, and equipment, and allow staff officers to view maps of areas of operation [OSF 96b].

Maturity

In early 1992, the OSF released the source code for DCE 1.0. Approximately 12 vendors had ported this version to their systems and had DCE 1.0 products available by June 1993. Many of these original products were "developer's kits" that were not robust, did not contain the entire set of DCE features (all lacked distributed file services), and were suited mostly for UNIX platforms [Chappell 93].

The DCE continues to evolve, but many large organizations have committed to basing their next generation systems on the DCE- over 14 major vendors provided DCE implementations by late 1994, when DCE 1.1 was released.

DCE 1.2.1, released in March 1996, provided the following new features:

- Interface definition language (IDL) support for C++ to include features such as inheritance and object references in support of object-oriented applications. This feature supports adoption of any object model or class hierarchy, thus providing developers with additional flexibility.
- Features to provide for coexistence with other application environments.
- Improvements over DCE 1.1 including enhancements to achieve greater reliability and better performance [OSF 96a].

Two other approaches to supporting objects are being considered besides the approach described for DCE 1.2:

1. Installing a CORBA-based product over DCE to provide additional support for distributed object technologies and a wide range of standardized service interfaces.
2. Integrating Network COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities) into the DCE infrastructure.

Costs and Limitations

DCE was not built to be completely object-oriented. The standard interfaces used by the DCE, as well as all the source code itself, are defined only in the C programming language. For object-oriented applications (i.e., applications being developed using an object-oriented language (see Object-Oriented Programming Languages) such as C++ or Ada 95, it may be more complex, less productive (thus more expensive), and less maintainable to use a non-object-oriented set of services like the DCE [Chappell 96].

Object-oriented extensions of the DCE have been developed by industry, but an agreed to vendor-neutral standard was still being worked in 1996.

Dependencies

Dependencies include Remote Procedure Call (RPC).

Alternatives

Alternatives include CORBA (see Common Object Request Broker Architecture), COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities), and message-oriented middleware (see Message-Oriented Middleware).

Complementary Technologies

DCE, in-part, has been used in building CORBA-compliant (see Common Object Request Broker Architecture) products as early as 1995. OSF is considering support for objects using COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities).

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Distributed Computing Environment
Application category	Distributed Computing (AP.2.1.2)
Quality measures category	Interoperability (QM.4.1) Portability (QM.4.2) Scalability (QM.4.3) Security (QM.2.1.5) Maintainability (QM.3.1) Complexity (QM.3.2.1) Throughput (QM.2.2.3)
Computing reviews category	Distributed Systems (C.2.4)

References and Information Sources

[Brando 96] Brando, T. "Comparing CORBA & DCE." *Object Magazine 6,* 1 (March 1996): 52-7.

[Chappell 93] Chappell, David. "OSF's DCE and DME: Here Today?" *Business Communications Review 23,* 7 (July 1993): 44-8.

[Chappell 96] Chappell, David. *DCE and Objects* [online]. Available WWW <URL: http://www.opengroup.org/dce/info/dce_objects.htm> (1996).

[OSF 96a] Open Software Foundation. *The OSF Distributed Computing Environment* [online]. Available WWW <URL: http://www.osf.org/dce/> (1996).

[OSF 96b] Open Software Foundation. *The OSF Distributed Computing Environment: End-User Profiles* [online]. Available WWW URL: < http://www.osf.org/comm/lit/dce-eup/> (1996).

[Product 96] *DCE Product Survey Report* [online]. Available WWW <URL: http://nsdir.cards.com/Libraries/HTML/PDLC/DCE_prod_surv_rpt.html> (1996).

[Schill 93] Schill, Alexander. "DCE-The OSF Distributed Computing Environment Client/Server Model and Beyond," 283. *International DCE Workshop*. Karlsruhe, Germany, October 7-8, 1993. Berlin, Germany: Springer-Verlag, 1993.

Current Author/Maintainer

Cory Vondrak, TRW, Redondo Beach, CA

Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM
10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/dce_body.html
Last Modified: 24 July 2008

Glossary Term

Security

the ability of a system to manage, protect, and distribute sensitive information.

References and Information Sources

[OSF 96b] Open Software Foundation. *The OSF Distributed Computing Environment: End-User Profiles* [online]. Available WWW URL: < http://www.osf.org/comm/lit/dce-eup/> (1996).

References and Information Sources

[Chappell 93] Chappell, David. "OSF's DCE and DME: Here Today?" *Business Communications Review 23*, 7 (July 1993): 44-8.

References and Information Sources

[Chappell 96] Chappell, David. *DCE and Objects* [online]. Available WWW <URL: http://www.osf.org/dce/3rd-party/ChapRpt1.html> (1996).

Related Topics

Distributed Computing (AP.2.1.2)

- Distributed Computing Environment
- Java
- TAFIM Reference Model

References and Information Sources

[Foreman 96] Foreman, John. *Product Line Based Software Development- Significant Results, Future Challenges*. Software Technology Conference, Salt Lake City, UT, April 23, 1996.

References and Information Sources

[Katz 94] Katz, S., et al. *Glossary of Software Reuse Terms*. Gaithersburg, MD: National Institute of Standards and Technology, 1994.

Domain Engineering and Domain Analysis

Software Technology Roadmap

Status

Advanced

Purpose and Origin

The term domain is used to denote or group a set of systems or functional areas, within systems, that exhibit similar functionality. Domain engineering is the foundation for emerging "product line" software development approaches [Foreman 96], and affects the *maintainability*, *understandability*, *usability*, and *reusability* characteristics of a system or family of similar systems.

The purpose of this technology description is to introduce the key concepts of domain engineering and provide overview information about domain analysis. Detailed discussions of individual domain analysis methods can be found in the referenced technology descriptions.

Technical Detail

Domain engineering and domain analysis are often used interchangeably and/or inconsistently. Although domain analysis as a term may pre-date domain engineering, domain engineering is the more inclusive term, and is the process of

- defining the scope (i.e., domain definition)
- analyzing the domain (i.e., domain analysis)
- specifying the structure (i.e., domain architecture development)
- building the components (e.g., requirements, designs, software code, documentation)

for a class of subsystems that will support reuse [Katz 94].

Figure 11 [Foreman 96] shows the process and products of the overall domain engineering activity, and shows the relationships and interfaces of domain engineering to the conventional (individual) system development (application engineering) process. This has come to be known as the two life cycle model.

Domain engineering is related to system engineering, which is an integrated set of engineering disciplines that supports the design, development, and operation of large-scale systems [Eisner 94]. Domain engineering is distinguished from system engineering in that it involves designing assets[1] for a *set or class of multiple applications* as opposed to designing the best solution for a single application. In addition, system engineering provides the "whole solution," whereas domain engineering defines (i.e., limits) the scope of functionality addressed across multiple systems [Simos 96].

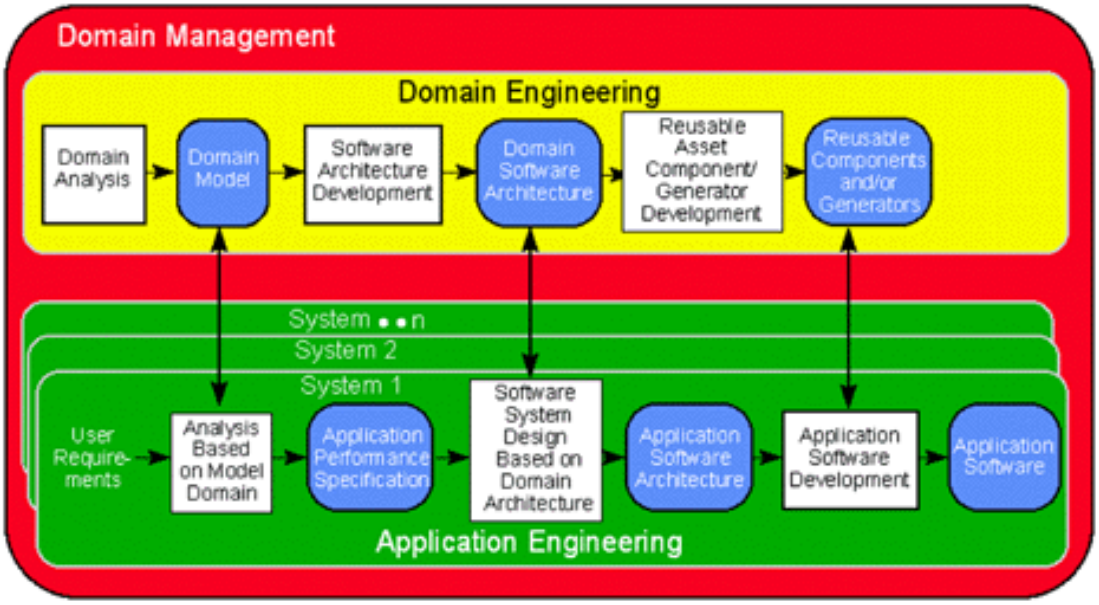


Figure 11: Domain Engineering and Application Engineering (Two Life Cycles)

Domain engineering supports systems engineering for individual systems by enabling coherent solutions across a family of systems: simplifying their construction, and improving the ability to analyze and predict the behavior of "systems of systems" composed of aggregations of those systems [Randall 96].

Domain analysis. Domain analysis (first introduced in the 1980s) is an activity within domain engineering and is the process by which information used in developing systems in a domain is identified, captured, and organized with the purpose of making it reusable when creating new systems [Prieto-Diaz 90]. Domain analysis focuses on supporting systematic and large-scale reuse (as opposed to opportunistic reuse, which suffers from the difficulty of adapting assets to fit new contexts) by capturing both the commonalities and the variabilities[2] of systems within a domain to improve the efficiency of development and maintenance of those systems. The results of the analysis, collectively referred to as a domain model, are captured for reuse in future development of similar systems and in maintenance planning of legacy systems (i.e., migration strategy) as shown in Figure 12 [Foreman 96].

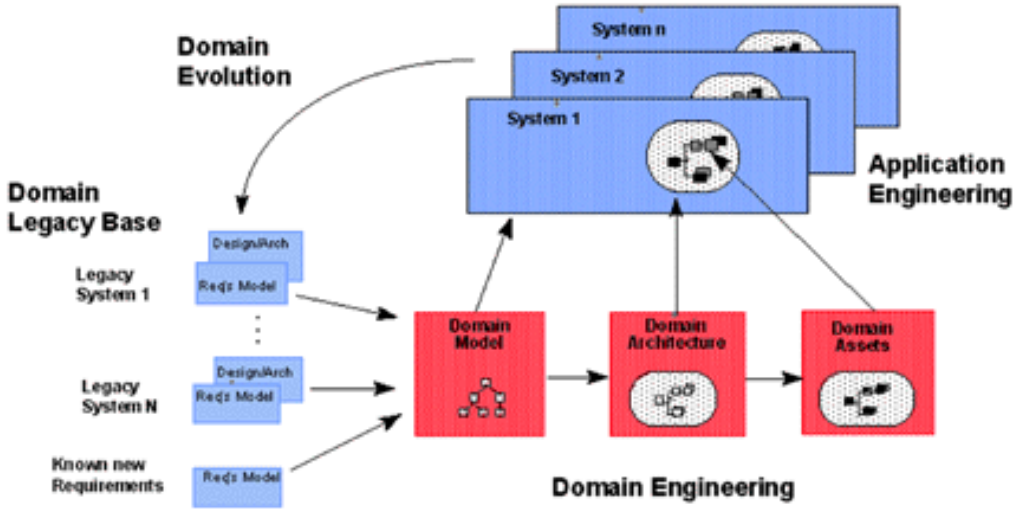


Figure 12: Domain Engineering and Legacy System Evolution

One of the major historical obstacles to reusing a software asset has been the uncertainty surrounding the asset. Questions to be answered included

- How does the software asset behave in its original context?
- How will it behave in a new context?
- How will adaptation affect its behavior [Simos 96]?

Design for reuse techniques (e.g., documentation standards, adaptation techniques) were developed to answer these questions; however, they did not provide the total solution, as a software asset's best scope needed to be determined (i.e., In which set of systems would the software asset be most likely reused?). Domain engineering and analysis methods were developed to answer more global questions, such as:

- Who are the targeted customers for the asset base (the designed collection of assets targeted to a specific domain)?
- Who are the other stakeholders in the domain?
- What is the domain boundary?
- What defines a feature of the domain?
- When is domain modeling complete?
- How do features vary across different usage contexts?
- How can the asset base be constructed to adapt to different usage contexts?

Goals of domain analysis include the following:

- Gather and correlate all the information related to a software asset. This will aid domain engineers in assessing the reusability of the asset. For example, if key aspects of the development documentation (e.g., chain of design decisions used in the development process) are available to a potential reuser, a more cost-effective reuse decision can be made.
- Model commonality and variability across a set of systems. This comparative analysis can reveal hidden contextual information in software assets and lead to insights about underlying rationale that would not have been discovered by studying a single system in isolation. It would answer questions like the following:
    - Why did developers make different design tradeoffs in one system than another?
    - What aspects of the development context influenced these decisions?
    - How can this design history be transformed into more prescriptive guidance to new developers creating systems within this domain?
- Derive common architectures and specialized languages that can leverage the software development process in a specific domain.

There is no standard definition of domain analysis; several domain analysis methods exist. Common themes among the methods include mechanisms to

- define the basic concepts (boundary, scope, and vocabulary) of the domain that can be used to generate a domain architecture
- describe the data (e.g., variables, constants) that support the functions and state of the system or family of systems
- identify relationships and constraints among the concepts, data, and functions within the domain
- identify, evaluate, and select assets for (re-)use
- develop adaptable architectures

Wartik provides criteria for comparing domain analysis methods [Wartik 92]. Major differences between the methods fall into three categories:

- *Primary product of the analysis.* In the methods, the results of the analysis and modeling activities may be represented differently. Examples include: different types of reuse library infrastructures (e.g., structured frameworks for cataloging the analysis results), application engineering processes, etc.
- *Focus of the analysis.* The methods differ in the extent they provide support for
    - context analysis: the process by which the scope of the domain is defined and analyzed to identify variability
    - stakeholder analysis: the process of modeling the set of stakeholders of the domain, which is the initial step in domain planning
    - rationale capture: the process for identifying and recording the

reasoning behind the design of an artifact
    - scenario definition: mechanisms to capture the dynamic aspects of the system
    - derivation histories: mechanisms for replaying the history of design decisions
    - variability modeling: the process for identifying the ways in which two concepts or entities differ
    - legacy analysis: the process for studying and analyzing an existing set of systems
    - prescriptive modeling: the process by which binding decisions and commitments about the scope, architecture, and implementation of the asset base are made
- *Representation techniques.* An objective of every domain analysis method is to represent knowledge in a way that is easily understood and machine-processable. Methods differ in the type of representation techniques they use and in the ease with which new representation techniques can be incorporated within the method.

Examples of domain analysis methods include

- Feature-Oriented Domain Analysis (FODA), a domain analysis method based upon identifying the features of a class of systems, defines three basic activities: context analysis, domain modeling, and architecture modeling [Kang 90].
- Organization Domain Modeling (ODM), a domain engineering method that integrates organizational and strategic aspects of domain planning, domain modeling, architecture engineering and asset base engineering [Simos 96].

Randall, Arango, Prieto-Diaz, and the Software Productivity Consortium offer other domain engineering and analysis methods [Randall 96, Arango 94, Prieto-Diaz 91, SPC 93].

Usage Considerations

Domain analysis is best suited for domains that are mature and stable, and where context and rationale for legacy systems can be rediscovered through analysis of legacy artifacts and through consultation with domain experts. In general, when applying a domain analysis method, it is important to achieve independence from architectural and design decisions of legacy systems. Lessons learned from the design and implementation of the legacy system are essential; however, the over-reliance on precedented features and legacy implementations may bias new developments.

Maturity

See individual technologies.

Costs and Limitations

See individual technologies.

Complementary Technologies

Use of visual programming techniques can provide better understanding of key software assets like execution patterns, specification and design animations, testing plans, and systems simulation. Other complementary technologies include comparative/taxonomic modeling and techniques for the development of adaptable architectures/implementations (e.g., generation, decision-based composition).

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Domain Engineering and Domain Analysis
Application category	Domain Engineering (AP.1.2.4)
Quality measures category	Reusability (QM.4.4) Maintainability (QM.3.1) Understandability (QM.3.2)
Computing reviews category	Software Engineering Tools and Techniques (D.2.2)

References and Information Sources

[Arango 94] Arango, G. "Domain Analysis Methods," 17-49. *Software Reusability*. Chichester, England: Ellis Horwood, 1994.

[Eisner 94] Eisner, H. "Systems Engineering Sciences," 1312-1322. *Encyclopedia of Software Engineering*. New York, NY: John Wiley and Sons, 1994.

[Foreman 96] Foreman, John. *Product Line Based Software Development-Significant Results, Future Challenges*. Software Technology Conference, Salt Lake City, UT, April 23, 1996.

[Hayes 94] Hayes-Roth, F. *Architecture-Based Acquisition and Development of Software: Guidelines and Recommendations from the ARPA Domain-Specific Software Architecture (DSSA) Program*. Palo Alto, CA: Teknowledge Federal Systems, 1994.

[IESE 98] Fraunhofer Institute for Experimental Software Engineering. *Domain Engineering Bibliography* [online]. Originally available WWW <URL:http://www.iese.fhg.de/ISE/DEbib/domain.html> (1998).

[Kang 90] Kang, K., et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-21, ADA 235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[Katz 94] Katz, S., et al. *Glossary of Software Reuse Terms*. Gaithersburg, MD: National Institute of Standards and Technology, 1994.

[Prieto-Diaz 90] Prieto-Diaz, R. "Domain Analysis: An Introduction." *Software Engineering Notes 15*, 2 (April 1990): 47-54.

[Prieto-Diaz 91] Prieto-Diaz, R. *Domain Analysis and Software Systems Modeling.* Los Alamitos, CA: IEEE Computer Society Press, 1991.

[Randall 96] Randall, Rick. *Space and Warning C2Product Line Domain Engineering Guidebook*, Version 1.0 [online]. Originally available WWW <URL: http://source.asset.com/stars/loral/domain/guide/delaunch.htm>

[Simos 96] Simos, M., et al. *Software Technology for Adaptable Reliable Systems (STARS) Organization Domain Modeling (ODM) Guidebook* Version 2.0 (STARS-VC-A025/001/00). Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996.

[SPC 93] *Reuse-Driven Software Processes Guidebook* Version 2.00.03 (SPC-92019-CMC). Herndon, VA: Software Productivity Consortium, 1993.

[Svoboda 96] Svoboda, Frank. *The Three "R's" of Mature System Development: Reuse, Reengineering, and Architecture* [online]. Available WWW <URL: http://source.asset.com/stars/darpa/Papers/ArchPapers.html> (1996).

[Wartik 92] Wartik, S. & Prieto-Diaz, R. "Criteria for Comparing Reuse-Oriented Domain Analysis Approaches." *International Journal of Software Engineering and Knowledge Engineering 2*, 3 (September 1992): 403-431.

Current Author/Maintainer

Liz Kean, Air Force Rome Laboratory

External Reviewers

Jim Baldo, MITRE, Washington, DC
Dick Creps, Lockheed Martin, Manassas, VA
Teri Payton, Lockheed Martin, Manassas, VA
Spencer Peterson, SEI
Rick Randall, Kaman Sciences, Colorado Springs, CO
Mark Simos, Organon Motives, Belmont, MA

Modifications

4 Feb 98: added reference for [IESE 98]
7 Oct 97: minor edits
10 Jan 97: (original)

Footnotes

[1] Examples include requirements, design, history of design decisions, source code, and test information.

[2] Commonality and variability refer to such items as functionality, data items, performance attributes, capacity, and interface protocols.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/deda_body.html
Last Modified: 24 July 2008

References and Information Sources

[Eisner 94] Eisner, H. "Systems Engineering Sciences," 1312-1322. *Encyclopedia of Software Engineering*. New York, NY: John Wiley and Sons, 1994.

Notes

[1] Examples include requirements, design, history of design decisions, source code, and test information.

References and Information Sources

[Simos 96] Simos, M., et al. *Software Technology for Adaptable Reliable Systems (STARS) Organization Domain Modeling (ODM) Guidebook* Version 2.0 (STARS-VC-A025/001/00). Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996. Also available [online] WWW <URL: http://www.asset.com/WSRD/abstracts/ABSTRACT_1176.html> (1996).

References and Information Sources

[Randall 96] Randall, Rick. *Space and Warning C2Product Line Domain Engineering Guidebook*, Version 1.0 [online]. Available WWW <URL: http://source.asset.com/stars/loral/domain/guide/delaunch.htm>.

References and Information Sources

[Prieto-Diaz 90] Prieto-Diaz, R. "Domain Analysis: An Introduction." *Software Engineering Notes 15*, 2 (April 1990): 47-54.

Notes

[2] Commonality and variability refer to such items as functionality, data items, performance attributes, capacity, and interface protocols.

References and Information Sources

[Wartik 92] Wartik, S. & Prieto-Diaz, R. "Criteria for Comparing Reuse-Oriented Domain Analysis Approaches." *International Journal of Software Engineering and Knowledge Engineering 2*, 3 (September 1992): 403-431.

References and Information Sources

[Kang 90] Kang, K., et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-21, ADA 235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

References and Information Sources

[Arango 94] Arango, G. "Domain Analysis Methods," 17-49. *Software Reusability*. Chichester, England: Ellis Horwood, 1994.

References and Information Sources

[Prieto-Diaz 91] Prieto-Diaz, R. *Domain Analysis and Software Systems Modeling.* Los Alamitos, CA: IEEE Computer Society Press, 1991.

References and Information Sources

[SPC 93] *Reuse-Driven Software Processes Guidebook* Version 2.00.03 (SPC-92019-CMC). Herndon, VA: Software Productivity Consortium, 1993.

References and Information Sources

[Bailin 90] Bailin, S., et al. "KAPTUR: Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationale," 95-104. *Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference*. Liverpool, NY, September 24-28, 1990. Rome, NY: Rome Air Development Center, 1990.

References and Information Sources

[Kang 90] Kang, K., et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-21, ADA 235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

References and Information Sources

[Cohen 92] Cohen, Sholom G., et al. *Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain* (CMU/SEI-91-TR-28, ADA 256590). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.

References and Information Sources

[Krut 93] Krut, Robert W. Jr. *Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology* (CMU/SEI-93-TR-11, ESC-TR-93-188) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

References and Information Sources

[Petro 95] Petro, James J.; Peterson, Alfred S.; & Ruby, William F. *In-Transit Visibility Modernization Domain Modeling Report Comprehensive Approach to Reusable Defense Software* (STARS-VC-H002a/001/00). Fairmont, WV: Comprehensive Approach to Reusable Defense Software, 1995.

References and Information Sources

[Devasirvatham 94] Devasirvatham, Josiah, et al. *In-Transit Visibility Modernization Domain Scoping Report Comprehensive Approach to Reusable Defense Software* (STARS-VC-H0002/001/00). Fairmont, WV: Comprehensive Approach to Reusable Defense Software, 1994.

References and Information Sources

[Schnell 96] Schnell, K.; Zalman, N.; & Bhatt, Atul. *Transitioning Domain Analysis: An Industry Experience* (CMU/SEI-96-TR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

Notes

[1] Refer to "Intrusion Detection FAQ: What is a bastion host?," available at http://www.sans.org/newlook/resources/IDFAQ/bastion.htm.

Notes

[2] Refer to "Screened Host Firewall," available at http://csrc.nist.gov/publications/nistpubs/800-10/node57.html.

Notes

[3] Refer to "Dual Homed Gateway Firewall," available at http://csrc.nist.gov/publications/nistpubs/800-10/node56.html

Notes

[4] Refer to "User Datagram Protocol RFC 768," available at http://www.ietf.org/rfc/rfc0768.txt.

Notes

[5] Refer to "Internet Control Message Protocol RFC 792," available at http://www.ietf.org/rfc/rfc0792.txt.

Notes

[6] Refer to "Simple Mail Transfer Protocol RFC 821," available at http://www.ietf.org/rfc/rfc0821.txt.

Notes

[7] Refer to "Hypertext Transfer Protocol RFC 2616," available at http://www.ietf.org/rfc/rfc2616.txt.

Notes

[8] Refer to "File Transfer Protocol RFC 959," available at http://www.ietf.org/rfc/rfc0959.txt.

Notes

[9] Refer to "Understanding TCP/IP," available at: http://www.cisco.com/univercd/cc/td/doc/product/iaabu/centri4/user/scf4ap1.htm

References and Information Sources

[Ogletree 00]	Ogletree, Terry William. Practical Firewalls. Que, June 2000.

References and Information Sources

[Smith 01] Smith, Gary. "A Brief Taxonomy of Firewalls-Great Walls of Fire." May 18, 2001. Available at http://www.sans.org/infosecFAQ/firewall/taxonomy.htm

References and Information Sources

[Tyson] Tyson, Jeff. "How Network Address Translation Works." Online: http://www.howstuffworks/nat.htm and Cisco Systems Inc. "How NAT Works." Online: http://www.cisco.com/warp/public/556/nat-cisco.shtml

References and Information Sources

[Marciniak 94] Marciniak, John J., ed. *Encyclopedia of Software Engineering*, 518-524. New York, NY: John Wiley & Sons, 1994.

References and Information Sources

[IFPUG 96] *The International Function Point Users' Group (IFPUG) Web site* [online]. Available WWW <URL: http://www.ifpug.org/> (1996).

References and Information Sources

[Umholtz 94] Umholtz, Donald C. & Leitgeb, Arthur J. "Engineering Function Points and Tracking Systems."*Crosstalk, Journal of Defense Software Engineering 7*, 11 (November 1994): 9-14.

References and Information Sources

[DeMarco 82] DeMarco, Tom. *Controlling Software Projects: Management, Measurement, and Estimation*. New York, NY: Yourdon Press, 1982.

References and Information Sources

[Rehesaar 96] Rehesaar, Hugo. "ISO/IEC Functional Size Measurement Standards," 311-318. *Proceedings of the GUFPI/IFPUG Conference on Software Measurement and Management*. Rome, Italy, February 5-9, 1996. Westerville, OH: International Function Point Users Group, 1996.

References and Information Sources

[Wittig 94] Wittig, G. E. & Finnie, G. R. "Software Design for the Automation of Unadjusted Function Point Counting," 613-623. *Business Process Re-Engineering Information Systems Opportunities and Challenges, IFIP TC8 Open Conference*. Gold Coast, Queensland, Australia, May 8-11, 1994. The Netherlands: IFIP, 1994.

References and Information Sources

[Kemerer 93] Kemerer, Chris. "Reliability of Function Points Measurement: A Field Experiment." *Communications of the ACM 36*, 2 (February 1993): 85-97.

References and Information Sources

[Siddiqee 93] Siddiqee, M. Waheed. "Function Point Delivery Rates Under Various Environments: Some Actual Results," 259-264. *Proceedings of the Computer Management Group's International Conference*. San Diego, CA, December 5-10, 1993. Chicago, IL: Computer Management Group, 1993.

References and Information Sources

[Jones 95] Jones, Capers. "Backfiring: Converting Lines of Code to Function Points." *IEEE Computer 28*, 11 (November 1995): 87-8.

References and Information Sources

[Selfridge 94] Selfridge, Peter G. & Heineman, George T. "Graphical Support for Code-Level Software Understanding," 114-24. *Ninth Knowledge-Based Software Engineering Conference*. Monterey, CA, September 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

References and Information Sources

[Bennett 95] Bennett, K. "Legacy Systems: Coping With Stress." *IEEE Software 12*, 1 (January 1995): 19-23.

Glossary Term

Compatibility

the ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment [IEEE 90].

References and Information Sources

[Gray 94] Gray, W. A.; Wikramanayake, G. N.; & Fiddian, N. J. "Assisting Legacy Database Migration," 5/1-3. *IEE Colloquium: Legacy Information System- Barriers to Business Process Re-Engineering* (1994/246). London, UK, December 13, 1994. London, UK: IEE, 1994.

References and Information Sources

[Ning 94] Ning, Jim Q.; Engberts, Andre; & Kozaczynski, W. "Automated Support for Legacy Code Understanding." *Communications of the ACM 37,* 5 (May 1994): 50-57.

Related Topics

Compatibility (QM.4.1.1)

- Graphic Tools for Legacy Database Migration

References and Information Sources

[Myers 95] Myers, Brad A. "User Interface Software Tools." *ACM Transactions on Computer-Human Interaction 2*, 1 (March 1995): 64-108.

References and Information Sources

[OSF 96] OSF Home Page [online]. Available WWW <URL: http://www.osf.org> (1996).

References and Information Sources

[Halstead 77] Halstead, Maurice H. *Elements of Software Science, Operating, and Programming Systems Series* Volume 7. New York, NY: Elsevier, 1977.

References and Information Sources

[Jones 94] Jones, Capers. "Software Metrics: Good, Bad, and Missing." *Computer 27, 9* (September 1994): 98-100.

References and Information Sources

[Oman 91] Oman, P. *HP-MAS: A Tool for Software Maintainability*, *Software Engineering* (#91-08-TR). Moscow, ID: Test Laboratory, University of Idaho, 1991.

References and Information Sources

[Szulewski 84] Szulewski, Paul, et al. *Automating Software Design Metrics* (RADC-TR-84-27). Rome, NY: Rome Air Development Center, 1984.

References and Information Sources

[Marciniak 94] Marciniak, John J., ed. *Encyclopedia of Software Engineering*, 131-165. New York, NY: John Wiley & Sons, 1994.

References and Information Sources

[Oman 94] Oman, P. & Hagemeister, J. "Constructing and Testing of Polynomials Predicting Software Maintainability." *Journal of Systems and Software 24*, 3 (March 1994): 251-266.

Related Topics

Debugger (AP.1.4.2.4)

- Halstead Complexity Measures
- Maintainability Index Technique for Measuring Program Maintainability

References and Information Sources

[Ware 79] Ware, W. H. *Security Controls for Computer Systems: Report of Defense Science Board, Task Force on Computer Security*. Santa Monica, CA: The Rand Corporation, 1979.

References and Information Sources

[Spafford 88] Spafford, Eugene H. *The Internet Worm Program: An Analysis* (CSD-TR-823). West Lafayette, IN: Purdue University, 1988.

References and Information Sources

[Kemmerer 94]	Kemmerer, Richard A. "Computer Security," 1153-1164. *Encyclopedia of Software Engineering.* New York, NY: John Wiley and Sons, 1994.

Glossary Term

Availability

the degree to which a system or component is operational and accessible when required for use [IEEE 90].

Glossary Term

Integrity

the degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data [IEEE 90].

Glossary Term

Confidentiality

the nonoccurrence of the unauthorized disclosure of information [Barbacci 95].

References and Information Sources

[Lunt 93] Lunt, Teresa F. "A Survey of Intrusion Detection Techniques." *Computers and Security 12*, 4 (June 1993): 405-418.

References and Information Sources

[Mukherjee 94] Mukherjee, Biswanath, L.; Heberlein, Todd; & Levitt, Karl N. "Network Intrusion Detection." *IEEE Network 8,* 3 (May/June 1994): 26-41.

Glossary Term

Throughput

the amount of work that can be performed by a computer system or component in a given period of time [IEEE 90].

References and Information Sources

[Sun 97e] Overview of Java [online]. Available WWW <URL: http://java.sun.com/docs/Overviews/java/java-overview-1.html> (1997).

References and Information Sources

[van Hoff 96] van Hoff, A. *Hooked on Java*. Reading, MA: Addison-Wesley, 1996.

Glossary Term

Trustworthiness

the degree to which a system or component avoids compromising, corrupting, or delaying sensitive information.

References and Information Sources

[Sun 99d] JavaBeans Home Page [online]. Available WWW <URL:http://java.sun.com/beans/index.html>

References and Information Sources

[Yourdon 96] Yourdon, Edward. "Java, the Web, and Software Development." *Computer 29*, 8 (August 1996): 25-30.

Java(TM)

Software Technology Roadmap

Status

Advanced

Purpose and Origin

Java™ is an object-oriented programming language (see Object-Oriented Programming Languages) developed by a small team of people headed by James Gosling at Sun Microsystems (development began in 1991) [Sun 97e]. It was originally intended for use in programming consumer devices, but when the explosion of interest in the Internet began in 1995 it became clear that Java was an ideal programming language for Internet applications [van Hoff 96]. Java addresses many of the issues of software distribution over a network, including *interoperability*, *security*, *portability*, and *trustworthiness*. When they are embedded in a Web page, Java programs are called "applets." Applets, in conjunction with JavaBeans(tm)[Sun 99d] provide a developer the flexibility to develop a more sophisticated user interface on a Web page [Yourdon 96]. Java applets provide executable content, such as event-driven pop-up windows and graphical user interface (GUI) widgets (see Graphical User Interface Builders) via peer classes (see Figure 19), which can support a variety of applications. Java applets are the dominant player of client side Internet computing. However, the server side computing, i.e. the code that generates the HTML contents, was considered a stronghold of better performance languages as C++ or script languages as PERL. This situation is changing with the release of Java 2 Enterprise Edition(tm) (J2EE) [Sun 99a]. J2EE is a new Java platform specifically designed to address the needs of enterprise server side computing. J2EE provides *scalability*, *interoperability*, *reliability*, *security*. Java is also re-addressing its original purpose (consumer devices) through JINI(tm) connection technology [Sun 97b]. JINI enables devices to work together without the burden of setting up networks, loading drivers and so on. JINI devices such as TVs, DVDs and cameras will be able to self-install, self-organize into communities, self-configure, and self-diagnose. Jini technology reduces dependence on system administrators, potentially lowering support costs and allowing impromptu device communities to assemble in places far from the traditional office. JINI mainly addresses *usability*, *cost of ownership* and *interoperability*.

Technical Detail

Java is a high-level programming language similar in flavor to Smalltalk and similar in syntax to C and C++. However, the Java language is far less complex than C++. It is an object-oriented, statically typed language that is architecture-neutral, multi-threaded, and robust. It provides built-in garbage collection, supports a single-inheritance class hierarchy, and does not use pointers, thereby eliminating three of the primary sources of errors in many C++ programs. Because it borrows its syntax from the widely known C and C++, the Java language feels familiar to most developers. Java provides flexibility in that it provides dynamic functionality. Classes are linked in as required and can be downloaded from across networks. Incoming code is verified before execution. Such flexibility is a paradigm shift from the normal model of computing, which usually requires the entire suite of possible functionality to be installed onto a user's platform prior to execution [Yourdon 96]. Java programs start as Java source code, which is then compiled to bytecode and stored on a server or a local computer in ".class" files. In order to execute a Java program, a user invokes a Java Virtual Machine (JVM) that executes the Java bytecode. Unlike most other programming languages, Java bytecode is *not* platform-specific or native to any particular processor; it is a "write once, run anywhere" approach. This platform-neutrality at both source and binary levels means Java is inherently portable. The Java system also provides an extensive library of classes that provides access to the underlying operating system. All of today's popular Web browsers contain a Java Virtual Machine (JVM), including Netscape Navigator, Microsoft Internet Explorer, and Sun's HotJava Browser. Desktop platforms such as Microsoft Windows, MacOS, OS/2 Warp, and Sun Solaris also provide a standalone JVM which can execute Java code. A new generation of so-called Network Computers[1] executes Java code directly. Sun is extending the availability of the JVM to enable Java programs to be deployed on a wide range of consumer devices, such as pagers, telephones, and televisions [Clark 97].

The relationships of code, Java Virtual Machines, and platform independence/ neutrality is shown in Figure 19.

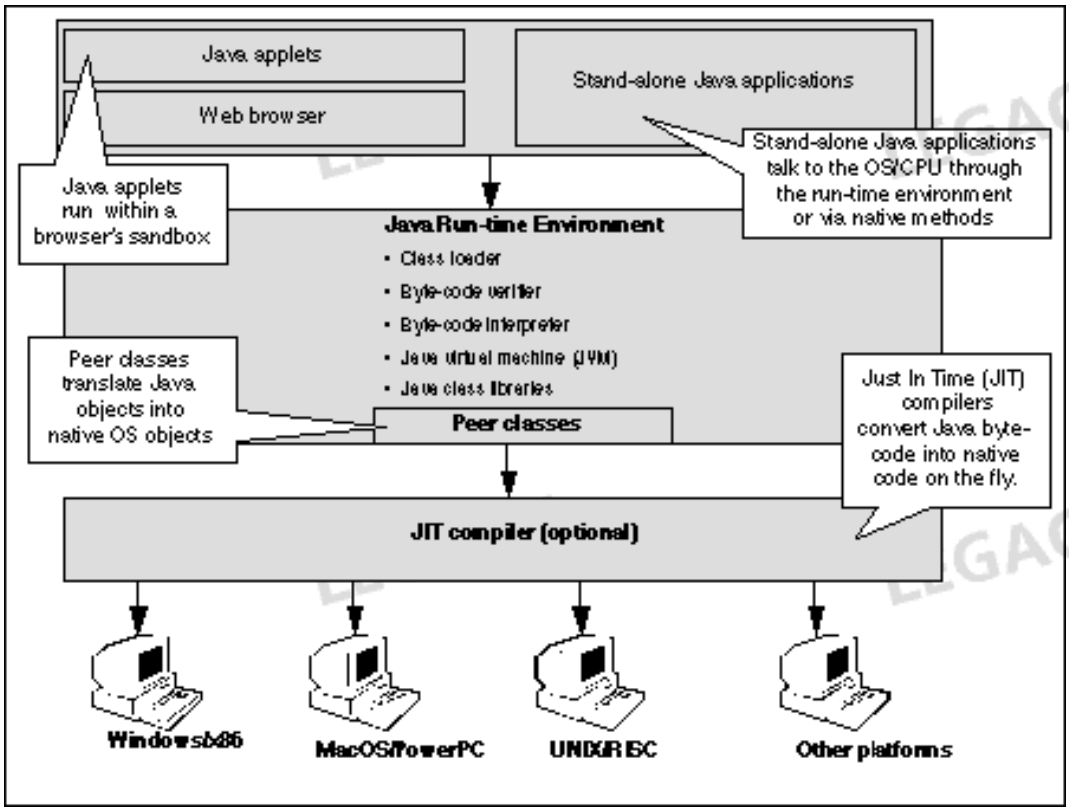


Figure 19: Multiple-Platform Application [Halfhill 97]

The Java platform provides portability, a measure of security, and inherent trustworthiness, including strong memory protection, encryption and signatures, rules enforcement, and runtime verification. Java is designed to allow applets to be downloaded and executed without introducing viruses or misbehaved code. It does this by placing strict limits on applets to prevent malicious actions. For example, applets cannot read from or write to the local disk. Unfortunately, while the Java model is theoretically secure, the various *implementations* of the JVM continue to show signs of weakness. Exploitation of security flaws in the implementations is still alarmingly common [Sun 97a]. An applet's actions are restricted to its "sandbox," an area of the Web browser dedicated to that applet and within which it may do anything it wants. But a Java applet can't read or alter any data outside its sandbox. Hence users can run untrusted Java code without compromising their trusted computing environments. Standalone windows created by Java applications are clearly labeled as being owned by untrusted software. Java applications are also prohibited from making network connections to other computers on a corporate Intranet, so malicious code can't exploit undiscovered security holes. Applets are not enough to build enterprise systems. Applets excel delivering functionality to remote clients, but enterprise applications need much more than remote access, like scalability and transactions. To address those needs Sun has developed Java(tm) 2 Platform, Enterprise Edition (J2EE). J2EE is a standard set of Java APIs that define a multi tier architecture (see Three Tier Software Architectures) suitable for the development, deployment, and management of enterprise applications written in the Java(tm) programming language. J2EE is functionally complete in the sense that it is possible to develop a large class of enterprise applications using only the J2EE APIs. Figure 20 illustrates the architecture of a J2EE application.

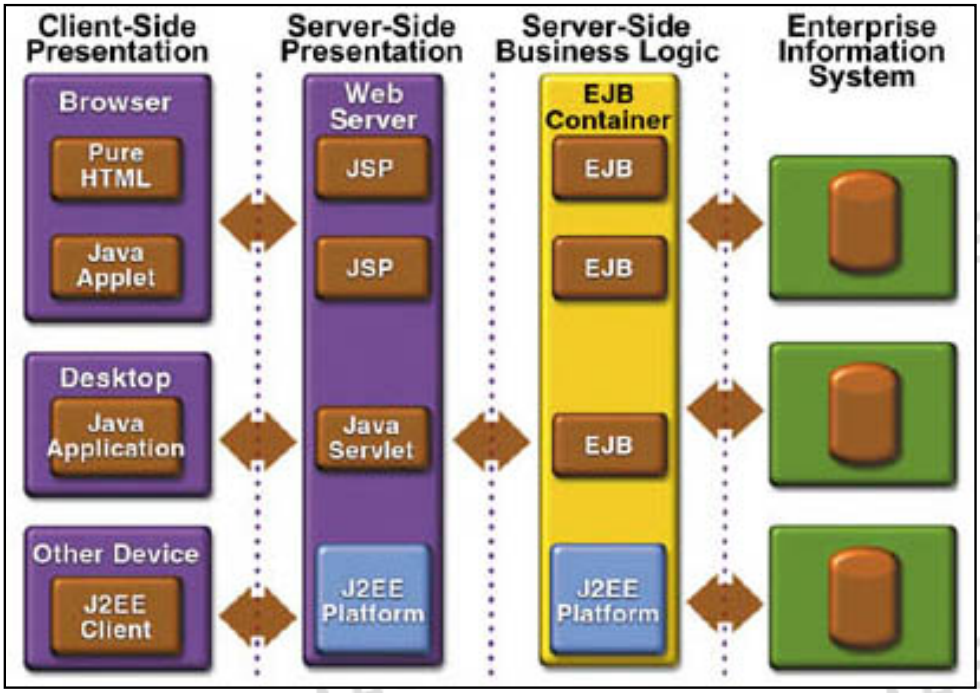


Figure 20: J2EE architecture [Sun 99f]

Remote clients are implemented as a combination of html pages and applets (or as Java Applications if Internet access is not required). The middle tier is split in two, the *Enterprise JavaBeans framework(tm)* (*EJB) [Sun 99e]* containing Enterprise Beans, which are reusable units that contain transactional business logic and the *Web Server* containing *JSP Pages and servlets that are* software entities that provide services in response to HTTP requests. The persistence layer can be implemented in any commercial database.

Usage Considerations

APIs. Java specifies a core set of Application Programming Interfaces (APIs) required in all Java implementations and an extended set of APIs covering a much broader set of functionality. The core set of APIs include interfaces for

- basic language types
- file and stream I/O
- network I/O
- container and utility classes
- abstract windowing toolkit

The extended set of APIs includes interfaces for 2D-rendering and 2D-animation; a 3D-programming model; telephony, time-critical audio, video, and MIDI[2] data; network and systems management; electronic commerce; and encryption and authentication [Hamilton 96]. J2EE introduces additional APIs to address specific need of enterprise environments. These APIs provide similar functionality to CORBA services including:

- Asynchronous communication through the Java Message Service (JMS)
- A naming service through the Java Naming and Directory Interface (JNDI)

- A transaction service through the Java Transaction API (JTA)
- Tabular data access though the JDBC API.

Platform-specific implementations. Recently, there has been some debate about the use of platform-specific APIs and the affect on the future of Java. For example, Microsoft's Internet Explorer 4.0 includes technology for J/Direct, which will provide a connection between Java and the Windows programming environment. Applications that make use of the J/Direct API will run only on the Windows platform, thereby curtailing one of Java's inherent benefits: platform neutrality. Providing Java developers with direct access to the Win32 API also breaks Java's security model and makes it more like Microsoft's platform-dependent ActiveX technology [Levin 97]. Sun has sued Microsoft for this practice, there is not a definitive resolution (by November 1999) but Microsoft has already been banned from using Java trademark with their modified versions.

Traning/education. The Java syntax for expressions and statements are almost identical to ANSI C, thus making the language easy to learn for C or C++ programmers. Because Java is a programming language, it requires a higher skill level for content developers than hypertext markup language (HTML). Programmers need to learn the Java standard library, which contains objects and methods for opening sockets, implementing the HTTP protocol, creating threads, writing to the display, and building a user interface. Java provides mechanisms for interfacing with other programming languages such as C and existing libraries such as Xlib, Motif, or legacy database software.

Performance. Performance is a major consideration when deciding to use Java. In most cases, interpreted Java is much slower than compiled C or C++ (as much as 10-15 times slower). However, most recent versions of the popular Web browsers and Java development environments provide Just In Time (JIT) compilers that produce native binary code (while the program is loaded and executed) that is beginning to rival that of optimized C++. The Java 2 platform also provides the Java HotSpot(tm) Performance Engine [Sun 97c] that combines the functionality of a JIT with runtime optimizations that further improve Java performance. For real-time applications, the performance implications of the Java garbage collector should also be considered. Garbage collection may make it difficult to easily bound timing properties of the application.

Language migration. A number of items should be considered if migrating from C or C++ to Java, including the following:

- Java is totally object-oriented; thus everything must be done via a method invocation.
- Java has no pointers or parameterized types.
- Java supports multithreading and garbage collection.

Maturity

Java was made available to the general public in May 1995, and has enjoyed unprecedented rapid transition into practice. Web sites such as the Java Applet Rating Service (JARS) [JARS 97] and Gamelan [Gamelan 97] contain literally thousands of Java-based applications available for downloading. All of today's

leading Web browsers provide support for Java by including a JVM as part of their product. There are multitudes of books available that describe all aspects of Java programming. Many commercial uses of Java have also appeared in a relatively short period of time. Sun provides a series of "customer success stories" at their web sites [Sun 97b, Sun 97c]. Some of the many commercial applications written in Java include

- TWSNet, a shipment tracking and processing application for CSX Corporation [Sun 96a]
- OC://WebConnect, a Web-based terminal emulation package for connecting to legacy SNA networks, from OpenConnect Systems
- via World Network, an online travel reservation system, from Andersen Consulting

There are now several development environments that support Java programming. These include IBM's Visual Age for Java, Symantec's Visual Café, Microsoft's J++, and Sun's Java Development Kit (JDK) [Sun 97d]. Most of these products provide integrated editors, debuggers, JIT compilers, and other tools commonly associated with computer-aided software engineering (CASE) tools. J2EE is one of the newest and less mature parts of Java. In fact, by November 1999 there is only a beta release of J2EE. Some constituents of the platform are quite stable but others are undertaking deep changes. EJB, for example, was released in Dec 1997 and there already are more than thirty implementations [EJB-SIG 99] (including from IBM, BEA, Oracle and IONA). However, EJB has suffered important changes from the 1.1 to the 1.2 release and that volatility is expected to continue with subsequent releases. In summary, J2EE is currently usable but there are several important issues to be solved and some time is needed until it delivers all its potential.

Costs and Limitations

Java and the source for the Java interpreter are freely available for noncommercial use. Some restrictions exist for incorporating Java into commercial products. Sun Microsystems licenses Java to hardware and software companies that are developing products to run the Java virtual machine and execute Java code. Developers, however, can write Java code without a license. A complete Java Development Kit (JDK), including a Java compiler, can be downloaded for free [Sun 97d]. Although a J2EE reference implementation will be provided by Sun, this implementation is not expected to be usable in industrial deployments. Several vendors are providing J2EE solutions ranging from free open source distributions to industrial strength distributions with per developer fees and per server fees. Yourdon discusses the potential impact of Java on the cost of software applications in the future- purchased software packages could be replaced with transaction-oriented rental of Java applets attached to Web pages [Yourdon 96].

Alternatives

From a programming-language point of view, alternatives to Java include C/C++, Perl, and Tcl/Tk. Scripting languages often used in Web browsers, such as

JavaScript and Visual Basic Scripting Edition (VB Script), can also be used to perform some of the tasks that Java can do, but not all of them. Perhaps the biggest challenge to client side Java's success is Microsoft's ActiveX technology. ActiveX is built on top of COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities). Microsoft provides tools for developers to create "ActiveX controls" that can serve a similar purpose to Java applets. The primary difference is that ActiveX is a proprietary technology that only runs on the Windows platform at present. It also provides a different security model based on its "Authenticode" certificate technology, security zones, and encrypted signatures. The ActiveX model itself is not secure in the way Java is; ActiveX controls have unlimited access to the user's machine when they are executing. This gives them more power to perform operations, but also makes them potentially more dangerous to the user's computing environment. The alternatives for the server side Java computing are CORBA and MTS. As CORBA and Java are basically complimentary technologies, only MTS can be considered as a J2EE's competitor. MTS is a product that provides to specifically designed COM objects with enterprise services as transactions and security. MTS and COM will converge into a single technology called COM+ that will be released with Windows 2000.

Complementary Technologies

The entire distributed object technology area (CORBA, COM/DCOM, ActiveX, etc.) offers technologies that can inter-operate with Java. There are standards available that let Java objects talk to CORBA objects, thus extending the capabilities of both technologies. Of particular relevance is the RMI-IIOP mapping that enable interoperability between RMI objects and IIOP objects. Also relevant is the CORBA 3 Component Model [OMG 99], this model is strongly based in EJB and has EJB interoperability as one of its main goals. Java provides solid foundations to component-based development (see Component-Based Software Development/COTS Integration). The additions of Remote Method Invocation (RMI), the JavaBeans component architecture [JavaSoft 97] and EJB component framework to Java facilitate the reuse of other people's software. Java components developed in this manner can have their interfaces examined, can communicate with one another over a network, and can be integrated with other components all without needing the source code. Java has evolved to become a serious option to implement three tier architectures (see Three Tier Software Architectures). Mobile and light clients can be implemented as Applets or JavaBeans, Enterprise JavaBeans are perfect for transactional business logic and Java Server Pages can be used to generate the html representation. The ability to deploy component-oriented enterprise multi-tiers systems, in a platform-neutral manner, can give fast moving enterprises a significant and measurable competitive edge.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Java
Application category	Distributed Computing (AP.2.1.2) Application Program Interfaces (AP.2.7) Programming Language (AP.1.4.2.1) Compiler (AP.1.4.2.3)
Quality measures category	Complexity (QM.3.2.1) Cost of Ownership (QM.5.1) Interoperability (QM.4.1) Maintainability (QM.3.1) Portability (QM.4.2) Reliability (QM.2.1.2) Scalability (QM.4.3) Trustworthiness (QM.2.1.4) Usability (QM.2.3)
Computing reviews category	Programming Languages (D.3) Distributed Systems (C.2.4)

References and Information Sources

[Clark 97] Clark, Don. "Sun Microsystems Pours Some New Java." *Wall Street Journal*, Page B-4. April 1, 1997.

[EJB-SIG 99] Special Interest Group - Enterprise JavaBeans [online] Available WWW, <URL: http://www.mgm-edv.de/ejbsig/ejbservers_tabled.html>

[Gamelan 97] Gamelan Web site [online]. Available WWW, <URL: http://www.gamelan.com> (1997).

[Gosling 96] Gosling, James & McGilton, Henry. *The Java Language Environment: A White Paper* [online]. Available WWW, <URL: http://java.sun.com/docs/white/langenv/> (1996).

[JARS 97] *Java Applet Rating Service* (JARS) [online]. Available WWW, <URL: http://www.jars.com> (1997).

[Halfhill 97] Halfhill, Tom R. "Today the Web, Tomorrow the World." *Byte 22*, 1 (January 1997): 68-80.

[Hamilton 96] Hamilton, Marc. "Java and the Shift to Net-Centric Computing." *Computer 29*, 8 (August 1996): 31-39.

[JavaSoft 97] *JavaBeans: The Only Component Architecture for Java* [online]. Available WWW, <URL: http://splash.javasoft.com/beans/> (1997).

[Levin 97] Levin, Rich and Patrizio, Andy. "Breaking Point" [online]. *Information Week*, June 23, 1997. Available WWW, <URL: http://techweb.cmp.com/iw/636/36iujav.htm> (1997).

[OMG 99] CORBA Component Model RFP [online]. Originally available WWW, <URL http://www.omg.org/techprocess/meetings/schedule/CORBA_Component_Model_RFP.html>

[Sun 96a] *CSX Gets on Track With Java* [online]. Available WWW, <URL: http://java.sun.com/features/1996/october/csx102996.html> (1996).

[Sun 96b] *Java Computing in the Enterprise. Strategic Overview: Java* [online]. Available WWW, <URL: http://www.sun.com/javacomputing> (1996).

[Sun 97a] *Frequently Asked Questions- Applet Security* [online]. Available WWW, <URL: http://java.sun.com/sfaq> (1997).

[Sun 97b] *Java in the Real World* [online]. Available WWW, <URL: http://java.sun.com/nav/used/> (1997).

[Sun 97c] *Customer Successes* [online]. Available WWW, <URL: http://www.sun.com/javastation/customersuccesses/> (1997).

[Sun 97d] *Java Development Kit 1.1.2* [online]. Available WWW, <URL: http://java.sun.com/products/jdk/1.1/> (1997).

[Sun 97e] Overview of Java [online]. Available WWW, <URL: http://java.sun.com/docs/Overviews/java/java-overview-1.html> (1997).

[Sun 99a] Java(tm) 2 Platform, Enterprise Edition [online]. Available WWW, <URL:http://java.sun.com/j2ee/>

[Sun 99b] Jini(tm) connection technology [online]. Available WWW, <URL: http://www.sun.com/jini/>

[Sun 99c] Java HotSpot(tm) Performance Engine [online]. Available WWW, <URL:http://java.sun.com/products/hotspot/>

[Sun 99d] JavaBeans Home Page [online]. Available WWW, <URL:http://java.sun.com/beans/index.html>

[Sun 99e] Enterprise JavaBeans Home Page [online]. Available WWW, <URL:http://java.sun.com/products/ejb/index.html>

[Sun 99f] J2EE Sun BluePrints(TM) [online]. Available WWW, <URL:http://java.sun.com/j2ee/blueprints/>

[van Hoff 96] van Hoff, A. *Hooked on Java*. Reading, MA: Addison-Wesley, 1996.

[Yourdon 96] Yourdon, Edward. "Java, the Web, and Software Development." *Computer 29*, 8 (August 1996): 25-30.

Current Author/Maintainer

Santiago Comella-Dorda, SEI
Scott Tilley, SEI

External Reviewers

Alan Brown, Texas Instruments
Hausi Müller, University of Victoria

Modifications

24 Feb 2000: Updated to cover new Java developments

More material added to cover Java 2, Jini, Java Beans, and Enterprise Java Beans. Many new references added.

30 June 1997: Substantially rewritten to reflect developments of the past 5-6 months.

More material added in Usage Considerations, Maturity, and Alternatives. Added figure to show applets, applications, Virtual Machines, and platform independence/neutrality. Many new references added.

10 Jan 1997 (original); author: Cory Vondrak, TRW, Redondo Beach, CA

Footnotes

[1] The network computer (NC) does not have an agreed-upon definition. Some NCs are new devices designed to run software written in Java, with gateways to existing programs and data. These are the official Network Computers (an Oracle trademark) and JavaStations (a Sun trademark). Other NCs are more like terminals in the classic sense: they don't execute programs at the desktop. Instead, applications run remotely on a server, and the client handles only the graphics locally. The generic term for these and the true NC alternatives to the personal computer (PC) is "thin client". They are referred to as "thin" because they are generally less complex and less expensive than a PC. However, recent developments by Microsoft and others have muddied the waters a bit with the "NetPC," which is essentially a stripped-down and sealed PC that is meant to be centrally administered. [2] MIDI stands for "Musical Instrument Digital Interface". It is a hardware specification and protocol used to communicate note and effect information between synthesizers, computers, keyboards, controllers and other electronic music devices.

Java™ Copyright

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/java_body.html
Last Modified: 24 July 2008

References and Information Sources

[Sun 99a] Java(tm) 2 Platform, Enterprise Edition [online]. Available WWW <URL:http://java.sun.com/j2ee/>

References and Information Sources

[Sun 97b] *Java in the Real World* [online]. Available WWW <URL: http://java.sun.com/nav/used/> (1997).

Glossary Term

Cost of ownership

the overall cost of a computer system to an organization to include the costs associated with operating and maintaining the system, and the lifetime of operational use of the system.

Notes

[1] The network computer (NC) does not have an agreed-upon definition. Some NCs are new devices designed to run software written in Java, with gateways to existing programs and data. These are the official Network Computers (an Oracle trademark) and JavaStations (a Sun trademark). Other NCs are more like terminals in the classic sense: they don't execute programs at the desktop. Instead, applications run remotely on a server, and the client handles only the graphics locally. The generic term for these and the true NC alternatives to the personal computer (PC) is "thin client". They are referred to as "thin" because they are generally less complex and less expensive than a PC. However, recent developments by Microsoft and others have muddied the waters a bit with the "NetPC," which is essentially a stripped-down and sealed PC that is meant to be centrally administered.

References and Information Sources

[Clark 97] Clark, Don. "Sun Microsystems Pours Some New Java." *Wall Street Journal*, Page B-4. April 1, 1997.

References and Information Sources

[Halfhill 97] Halfhill, Tom R. "Today the Web, Tomorrow the World." *Byte 22*, 1 (January 1997): 68-80.

References and Information Sources

[Sun 97a] *Frequently Asked Questions- Applet Security* [online]. Available WWW <URL: http://java.sun.com/sfaq> (1997).

References and Information Sources

[Sun 99f] J2EE Sun BluePrints(TM) [online]. Available WWW, <URL:http://java.sun.com/j2ee/blueprints/>

References and Information Sources

[Sun 99e] Enterprise JavaBeans Home Page [online]. Available WWW <URL:http://java.sun.com/products/ejb/index.html>

Notes

[2] MIDI stands for "Musical Instrument Digital Interface". It is a hardware specification and protocol used to communicate note and effect information between synthesizers, computers, keyboards, controllers and other electronic music devices.

References and Information Sources

[Hamilton 96] Hamilton, Marc. "Java and the Shift to Net-Centric Computing." *Computer 29*, 8 (August 1996): 31-39.

References and Information Sources

[Levin 97] Levin, Rich and Patrizio, Andy. "Breaking Point" [online]. *Information Week*, June 23, 1997. Available WWW <URL: http://techweb.cmp.com/iw/636/36iujav.htm> (1997).

References and Information Sources

[Sun 97c] *Customer Successes* [online]. Available WWW <URL: http://www.sun.com/javastation/customersuccesses/> (1997).

References and Information Sources

[JARS 97] *Java Applet Rating Service* (JARS) [online]. Available WWW <URL: http://www.jars.com> (1997).

References and Information Sources

[Gamelan 97]	Gamelan Web site [online]. Available WWW <URL: http://www.gamelan.com> (1997).

References and Information Sources

[Sun 97c] *Customer Successes* [online]. Available WWW <URL: http://www.sun.com/javastation/customersuccesses/> (1997).

References and Information Sources

[Sun 96a] *CSX Gets on Track With Java* [online]. Available WWW <URL: http://java.sun.com/features/1996/october/csx102996.html> (1996).

References and Information Sources

[Sun 97d] *Java Development Kit 1.1.2* [online]. Available WWW <URL: http://java.sun.com/products/jdk/1.1/> (1997).

References and Information Sources

[EJB-SIG 99] Special Interest Group - Enterprise JavaBeans [online] Available WWW <URL: http://www.mgm-edv.de/ejbsig/ejbservers_tabled.html>

References and Information Sources

[OMG 99] CORBA Component Model RFP [online]. Available WWW, <URL: http://www.omg.org/techprocess/meetings/schedule/CORBA_Component_Model_RFP.html>

References and Information Sources

[JavaSoft 97] *JavaBeans: The Only Component Architecture for Java* [online]. Available WWW <URL: http://splash.javasoft.com/beans/> (1997).

References and Information Sources

[Siwolp 95] Siwolp, Sana. "Not Your Father's Mainframe." *Information Week* 546 (Sept 25, 1995): 53-58.

References and Information Sources

[Data 96] *Data Warehousing* [online]. Available WWW <URL: http://www-db.stanford.edu/warehousing/publications.html> and <URL: http://www-db.stanford.edu/warehousing/warehouse.html> (1996).

Mainframe Server Software Architectures

Software Technology Roadmap

Status

Complete

Note

We recommend Client/Server Software Architectures *as prerequisite reading for this technology description.*

Purpose and Origin

Since 1994 mainframes have been combined with distributed architectures to provide massive storage and to improve system security, *flexibility*, *scalability*, and *reusability* in the client/server design. In a mainframe server software architecture, mainframes are integrated as servers and data warehouses in a client/server environment. Additionally, mainframes still excel at simple transaction-oriented data processing to automate repetitive business tasks such as accounts receivable, accounts payable, general ledger, credit account management, and payroll. Siwolp and Edelstein provide details on mainframe server software architectures see [Siwolp 95, Edelstein 94].

Technical Detail

While client/server systems are suited for rapid application deployment and distributed processing, mainframes are efficient at online transactional processing, mass storage, centralized software distribution, and data warehousing [Data 96]. Data warehousing is information (usually in summary form) extracted from an operational database by data mining (drilling down into the information through a series of related queries). The purpose of data warehousing and data mining is to provide executive decision makers with data analysis information (such as trends and correlated results) to make and improve business decisions.

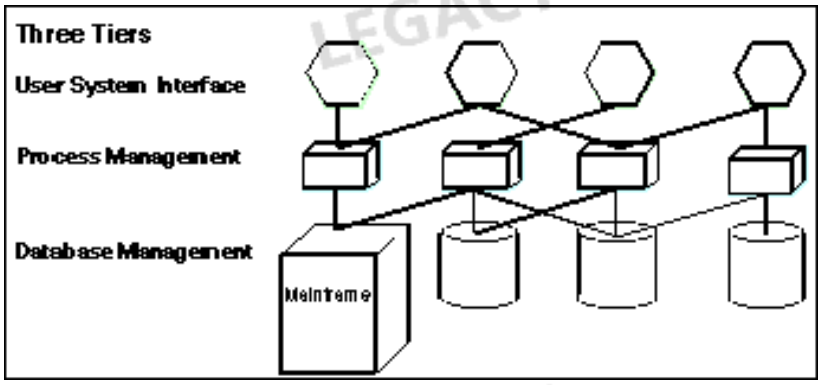


Figure 20: Mainframe in a Three Tier Client/Server Architecture

Figure 20 shows a mainframe in a three tier client/server architecture. The combination of mainframe horsepower as a server in a client/server distributed architecture results in a very effective and efficient system. Mainframe vendors are now providing standard communications and programming interfaces that make it easy to integrate mainframes as servers in a client/server architecture. Using mainframes as servers in a client/server distributed architecture provides a more modular system design, and provides the benefits of the client/server technology.

Using mainframes as servers in a client/server architecture also enables the distribution of workload between major data centers and provides disaster protection and recovery by backing up large volumes of data at disparate locations. The current model favors "thin" clients (contains primarily user interface services) with very powerful servers that do most of the extensive application and data processing, such as in a two tier architecture. In a three tier client/server architecture, process management (business rule execution) could be off-loaded to another server.

Usage Considerations

Mainframes are preferred for big batch jobs and storing massive amounts of vital data. They are mainly used in the banking industry, public utility systems, and for information services. Mainframes also have tools for monitoring performance of the entire system, including networks and applications not available today on UNIX servers [Siwolp 95].

New mainframes are providing parallel systems (unlike older bipolar machines) and use complementary metal-oxide semiconductor (CMOS) microprocessors, rather than emitter-coupler logic (ECL) processors. Because CMOS processors are packed more densely than ECL microprocessors, mainframes can be built much smaller and are not so power-hungry. They can also be cooled with air instead of water [Siwolp 95].

While it appeared in the early 1990s that mainframes were being replaced by client/server architectures, they are making a comeback. Some mainframe vendors have seen as much as a 66% jump in mainframe shipments in 1995

due to the new mainframe server software architecture [Siwolp 95].

Given the cost of a mainframe compared to other servers, UNIX workstations and personal computers (PCs), it is not likely that mainframes would replace all other servers in a distributed two or three tier client/server architecture.

Maturity

Mainframe technology has been well known for decades. The new improved models have been fielded since 1994. The new mainframe server software architecture provides the distributed client/server design with massive storage and improved security capability. New technologies of data warehousing and data mining data allow extraction of information from the operational mainframe server's massive storage to provide businesses with timely data to improve overall business effectiveness. For example, stores such as Wal-Mart found that by placing certain products in close proximity within the store, both products sold at higher rates than when not collocated.[1]

Costs and Limitations

By themselves, mainframes are not appropriate mechanisms to support graphical user interfaces. Nor can they easily accommodate increases in the number of user applications or rapidly changing user needs [Edelstein 94].

Alternatives

Using a client/server architecture without a mainframe server is a possible alternative. When requirements for high volume (greater than 50 gigabit), batch type processing, security, and mass storage are minimal, three tier or two tier architectures without a mainframe server may be viable alternatives. Other possible alternatives to using mainframes in a client/server distributed environment are using parallel processing software architecture or using a database machine.

Complementary Technologies

A complementary technology to mainframe server software architectures is open systems . This is because movement in the industry towards interoperable heterogeneous software programs and operating systems will continue to increase reuse of mainframe technology and provide potentially new applications for mainframe capabilities.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Mainframe Server Software Architectures
Application category	Client/Server (AP.2.1.2.1)
Quality measures category	Maintainability (QM.3.1) Scalability (QM.4.3) Reusability (QM.4.4)
Computing reviews category	Distributed Systems (C.2.4)

References and Information Sources

[Data 96]	*Data Warehousing* [online]. Available WWW <URL: http://www-db.stanford.edu/warehousing/publications.html> and <URL: http://www-db.stanford.edu/warehousing/warehouse.html> (1996).
[Edelstein 94]	Edelstein, Herb. "Unraveling Client/Server Architecture." *DBMS 7*, 5 (May 1994): 34(7).
[Siwolp 95]	Siwolp, Sana. "Not Your Father's Mainframe." *Information Week* 546 (Sept 25, 1995): 53-58.

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Frank Rogers, GTE

Modifications

10 Jan 97 (original)

Footnotes

[1] Source: Stodder, David. Open Session Very Large Data Base (VLDB) Summit, New Orleans, LA 23-26 April, 1995.

sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/mssa_body.html
Last Modified: 24 July 2008

Notes

[1] Source: Stodder, David. Open Session Very Large Data Base (VLDB) Summit, New Orleans, LA 23-26 April, 1995.

Notes

[1] In an expert system, knowledge about a problem domain is represented by a set of rules. These rules consist of two parts:

1. The antecedent, which defines when the rule should be applied. An expert system will use pattern matching techniques to determine when the observed data matches or satisfies the antecedent of a rule.
2. The consequent, which defines the action(s) that should be taken if its antecedent is satisfied.

A rule is said to be "fired" when the action(s) defined in its consequent are executed. For RBID systems, rule antecedents will typically be defined in terms of audit trail data, while rule consequents may be used to increase or decrease the level of monitoring of various entities, or they may be used to notify system administration personnel about significant changes in system state.

References and Information Sources

[Ilgun 93] Ilgun, Koral. "USTAT: A Real-time Intrusion Detection System for UNIX," 16-28. *Proceedings of the 1993 Computer Society Symposium on Research in Security and Privacy*. Oakland, California, May 24-26, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

References and Information Sources

[Tener 86] Tener, W. T. "Discovery: An Expert System in the Commercial Data Security Environment." *Computer Security Journal 6*, 1 (Summer 1990): 45.

References and Information Sources

[Denning 87] Denning, Dorothy E., et al. "Views for Multilevel Database Security." *IEEE Transactions on Software Engineering SE-13,* 2 (February 1987): 129-140.

References and Information Sources

[Vaccarro 89] Vaccarro, H. S. & Liepins, G. E. "Detection of Anomalous Computer Session Activity," 208-209. *Proceedings of the IEEE Symposium on Research in Security and Privacy*. Oakland, California, May 1-3, 1989. Washington, DC: IEEE Computer Society Press, 1989.

Notes

[1] The IETF is a large open community of network designers, operators, vendors, and researchers whose purpose is to coordinate the operation, management and evolution of the Internet, and to resolve short- and mid-range protocol and architectural issues. It is a major source of proposed protocol standards which are submitted to the Internet Engineering Steering Group for final approval. The IETF meets three times a year and extensive minutes of the plenary proceedings are issued.

Notes

[2] The IAB is a technical advisory group of the Internet Society. The IAB provides oversight of the architecture for the protocols and procedures used by the Internet, the process used to create Internet Standards and serves as an appeal board for complaints of improper execution of the standards process.

References and Information Sources

[IETF 96] Internet Engineering Task Force home page [online]. Available WWW <URL: http://www.ietf.cnri.reston.va.us/> (1996).

References and Information Sources

[Henderson 95] Henderson and Erwin. "SNMP Version 2: Not So Simple." *Business Communications Review 25*, 5 (May 1995): 44-48.

References and Information Sources

[SNMPv1 Specs] The following RFC's identify the major components of SNMPv1 online]. Available WWW
<URL: http://www.cis.ohio-state.edu/htbin/rfc/rfcXXXX.html> (1996).

Historical

RFC 1156 - Management Information Base Network Management of TCP/IP based internets
RFC 1161 - SNMP over OSI

Informational

RFC 1215 - A Convention for Defining Traps for use with the SNMP
RFC 1270 - SNMP Communication Services
RFC 1303 - A Convention for Describing SNMP-based Agents
RFC 1470 - A Network Management Tool Catalog

Standard and Draft

RFC 1089 - SNMP over Ethernet
RFC 1140 - IAB Official Protocol Standards
RFC 1155 - Structure and Identification of Management Information for TCP/IP based internets.
RFC 1157 - A Simple Network Management Protocol
RFC 1158 - Management Information Base Network Management of TCP/IP based internets: MIB-II
RFC 1187 - Bulk Table Retrieval with the SNMP
RFC 1212 - Concise MIB Definitions
RFC 1213 - Management Information Base for Network Management of TCP/IP-based internets: MIB-II
RFC 1224 - Techniques for Managing Asynchronously-Generated Alerts
RFC 1418 - SNMP over OSI
RFC 1419 - SNMP over AppleTalk
RFC 1420 - SNMP over IPX

References and Information Sources

[SNMPv2 Specs] The following RFC's identify the major components of SNMPv2 online]. Available WWW
<URL: http://www.cis.ohio-state.edu/htbin/rfc/rfcXXXX.html> (1996).

Historical

RFC 1441 - Introduction to SNMP v2
RFC 1442 - SMI For SNMP v2
RFC 1443 - Textual Conventions for SNMP v2
RFC 1444 - Conformance Statements for SNMP v2
RFC 1445 - Administrative Model for SNMP v2
RFC 1446 - Security Protocols for SNMP v2
RFC 1447 - Party MIB for SNMP v2
RFC 1448 - Protocol Operations for SNMP v2
RFC 1449 - Transport Mappings for SNMP v2
RFC 1450 - MIB for SNMP v2
RFC 1451 - Manager to Manager MIB
RFC 1452 - Coexistence between SNMP v1 and SNMP v2

Draft

RFC 1902 - SMI for SNMPv2
RFC 1903 - Textual Conventions for SNMPv2
RFC 1904 - Conformance Statements for SNMPv2
RFC 1905 - Protocol Operations for SNMPv2
RFC 1906 - Transport Mappings for SNMPv2
RFC 1907 - MIB for SNMPv2
RFC 1908 - Coexistence between SNMPv1 and SNMPv2

Experimental

RFC 1901 - Introduction to Community-based SNMPv2
RFC 1909 - An Administrative Infrastructure for SNMPv2
RFC 1910 - User-based Security Model for SNMPv2

Simple Network Management Protocol

Software Technology Roadmap

Status

Advanced

Note

We recommend Network Management -- An Overview *as prerequisite reading for this technology description.*

Purpose and Origin

Simple Network Management Protocol (SNMP) is a network management specification developed by the Internet Engineering Task Force (IETF),[1] a subsidiary group of the Internet Activities Board (IAB),[2] in the mid 1980s to provide standard, simplified, and extensible management of LAN-based internetworking products such as bridges, routers, and wiring concentrators [IETF 96, Henderson 95]. SNMP was designed to reduce the complexity of network management and minimize the amount of resources required to support it. SNMP provides for centralized, robust, interoperable network management, along with the flexibility to allow for the management of vendor-specific information.

Technical Detail

SNMP is a communication specification that defines how management information is exchanged between network management applications and management agents. There are several versions of SNMP, two of the most common are SNMPv1 [SNMPv1 Specs] and SNMPv2 [SNMPv2 Specs]. SNMPv2 and some of the less common versions will be discussed later in this text.

The architecture of SNMPv1 is shown in Figure 33, which is a more detailed version of the managed device and network management application shown in Figure 27 of Network Management-An Overview. SNMPv1 is a simple message based request/response application-layer protocol which typically uses the User Datagram Protocol (UDP) [RFC 96] for data delivery. The SNMPv1 network management architecture contains:

- Network Management Station (NMS) - Workstation that hosts the network

management application.

- SNMPv1 network management application - Polls management agents for information and provides control information to agents.
- Management Information Base (MIB) - Defines the information that can be collected and controlled by the management application.
- SNMPv1 management agent(s) - Provides information contained in the MIB to management applications and may accept control information.

A MIB is basically a database of managed objects[3] that resides on the agent. Managed objects are a characteristic of a managed device that can be monitored, modified or controlled, such as a threshold, network address or counter. The management application or user can define the relationship between the SNMPv1 manager and the management agent.

Attributes of managed objects may be monitored or set by the network management application using the following operations:

- GET_NEXT_REQUEST - Requests the next object instance from a table or list from an agent
- GET_RESPONSE - Returned answer to *get_next_request*, *get_request*, or *set_request*
- GET_REQUEST - Requests the value of an object instance from the agent
- SET_REQUEST - Set the value of an object instance within an agent
- TRAP - Send trap (event) asynchronously to *network management application.* Agents can send a trap when a condition has occurred, such as change in state of a device, device failure or agent initialization/restart.

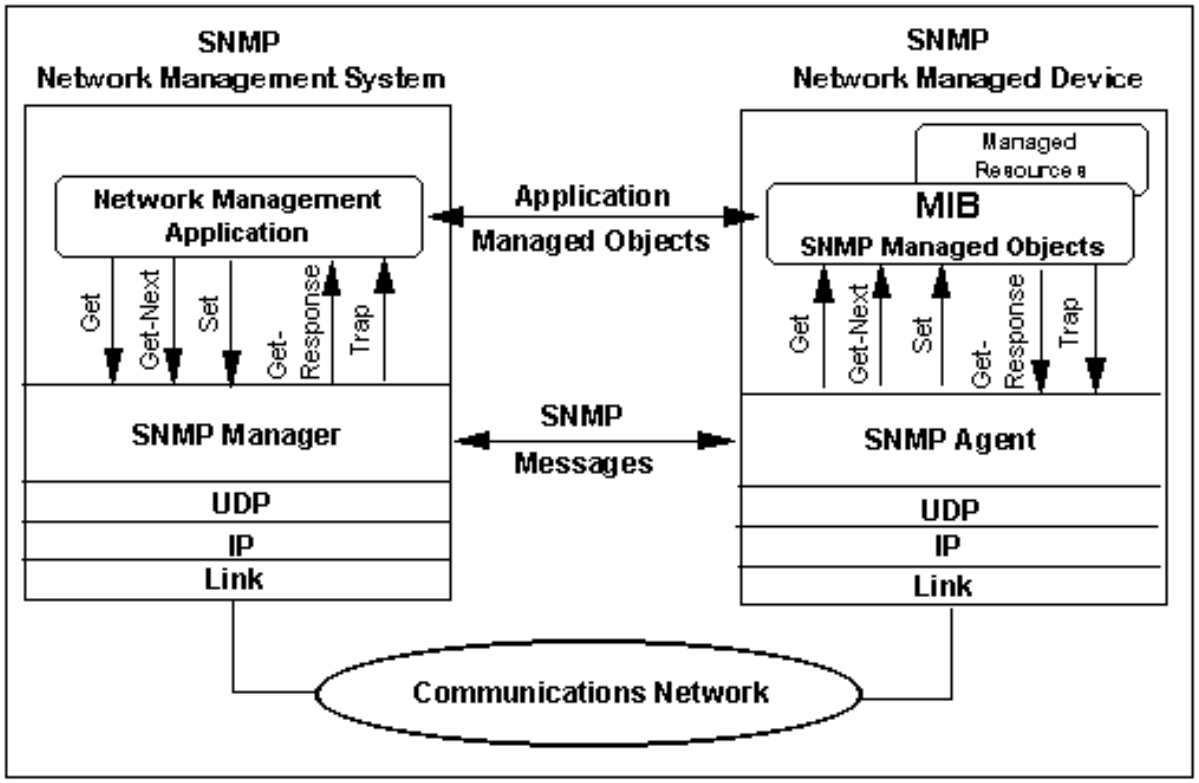


Figure 33: The SNMPv1 Architecture [Lake 96]

By specifying the protocol to be used between the network management application and management agent, SNMP allows products (software and managed devices) from different vendors (and their associated management agents) to be managed by the same SNMP network management application. A "proxy function" is also specified by SNMP to enable communication with non-SNMP devices to accommodate legacy equipment.

The main attributes of SNMP are as follows [Moorhead 95]:

- It is simple to implement, making it easy for a vendor to accommodate it into its device.
- It does not require large computational or memory resources from the devices that do accommodate it.

Network management, as defined by SNMP, is based on polling and asynchronous events. The SNMP manager polls for information gathered by each of the agents. Each agent has the responsibility of collecting information (e.g., performance statistics) pertaining to the device it resides within and storing that information in the agent's own management information base (MIB). This information is sent to the SNMP manager in response to the manager's polling.

SNMP events (alerts) are driven by trap messages generated as a result of certain device parameters. These parameters can be either generic or vendor device specific. Enterprise-specific trap messages are vendor proprietary and generally provide more device-specific detail.

The SNMPv2 [SNMPv2 Specs] (SNMP Version 2) specification included the following new capabilities:

- manager to manager communication to support the coexistence of multiple/ distributed managers and mid-level managers, increasing the flexibility and scalability of the network being managed
- enhanced security (known as "Secure SNMP") by specifying three layers of security
  - encryption: Used to keep content of messages private. Encryption is based on the Data Encryption Standard (DES) [DES 93] defined by the National Institute of Standards and Technology (NIST) and the American National Standards Institute (ANSI)[4].
  - authentication: Proof of the identity of the sender of a message.
  - authorization: Provides access restrictions thru access control lists.
- improved efficiency and performance through the addition of bulk transfers of data. This means that in some cases, using SNMPv2 instead of SNMPv1, network management can be provided over low-bandwidth, wide-area links.
- support for additional network protocols besides UDP/IP, for example, OSI, NetWare IPX/SPX and Appletalk [Broadhead 95]

Usage Considerations

Problem isolation. Neither version of SNMP does an effective job at helping network managers isolate problem devices in large, complex networks. It sometimes becomes

difficult for an SNMP manager to determine which network events/alarms are significant-- all are treated equally.

Focus. SNMPv1 provides information only on individual devices, not on how the devices work as a system.

Incompatibilities. SNMPv1 and SNMPv2 are incompatible with each other and can not interact, however, some SNMP network management applications packages support both specifications.

Performance. The performance impact on the network being managed should be considered when using the polling scheme that SNMP uses for collecting information from distributed agents. A higher frequency of polling, which may be required to manage a network effectively, will increase the overhead on a network, possibly resulting in a need for additional networking or processor resources. The frequency of polling can be controlled by the SNMP manager, but can be dependent on what kind of messages (generic or enterprise-specific) a device vendor supports. Many vendors offer generic trap messages on their devices rather than enterprise-specific messages, because it is easier and takes less time for the vendor to implement. Devices that provide only generic trap information must be polled frequently to obtain the granularity of information to manage the device effectively.

Maturity

SNMPv1 has been incorporated into many products and management platforms. It has been deployed by virtually all internetworking vendors. It has been widely adopted for the enterprise (business organization) networks and may be the manager of choice for the internetworking arena in the future because it is well-suited for managing TCP/IP networks. Limitations are discussed below in Costs and Limitations.

SNMPv2 has many unresolved issues and was supported by few vendors as of January 1998. The members of IETF subcommittee can not agree upon several parts of the SNMPv2 specification (primarily the security and administrative needs of the protocol); as a result only certain parts of SNMPv2 specification have reached draft standard status within the IETF [SNMP FAQ 98]. There has been several attempts to achieve acceptance of SNMPv2 through the release of experimental modified versions commonly known as SNMPv2*, SNMPv2c, SNMPv2u, SNMPv1+ and SNMP1.5 that do not contain the contentious parts.

SNMPv3 is the latest proposed version for the next generation of SNMP functionality. It is based upon the protocol operations, data types, and proxy support from SNMPv2 with user-based seucurity from SNMPv2u and SNMPv2*. It may take years before a new version is accepted.

Costs and Limitations

The attractiveness of SNMP is its simplicity and relative ease of implementation. With this comes a price: e.g., the more fine grained information that is need or required, such as the variance in interarrival time (jitter) of packets sent to a particular local address, the less likely it is that it will be available.

SNMPv1 uses the underlying User Datagram Protocol (UDP) for data delivery, which does not ensure reliability of data transfer. The loss of data may be a limitation to a network manager, depending on the criticality of the information being gathered and the frequency at which the polling is being performed.

SNMP is best suited for network monitoring and capacity planning. SNMP does not provide even the basic troubleshooting information that can be obtained from simple network troubleshooting tools [Wellens 96]. SNMP agents do not analyze information, they just collect information and provide it to the network management application.

SNMPv1 has minimal security capability. Because SNMPv1 lacks the control of unauthorized access to critical network devices and systems, it may be necessary to restrict the use of SNMP management to non-critical networks. Lack of authentication in SNMPv1 has led many vendors to not include certain commands, thus reducing extensibility and consistency across managed devices. SNMPv2 addresses these security problems but is difficult and expensive to set up and administer (e.g., each MIB must be locally set up).

Vendors often include SNMP agents with their software and public domain agents are available. Management applications are available from a variety of vendors as well as the public domain, however they can differ greatly in terms of functionality, plots and visual displays.

SNMP out-of-the-box can not be used to track information contained in application/user level protocols (e.g., radar track message, http, mail). However these might be accomplished through the use of a extensible (customized) SNMP agent that has user defined MIB.[5] It is important to note that a specialized or extensible network manager may be required for use with the customized agents.

There are also concerns about the use of SNMP in the real-time domain where bounded response, deadlines, and priorities are required.

SNMPv2 is intended to be able to coexist with existing SNMPv2, but in order to use SNMPv2 as the SNMP manager or to migrate from SNMPv1 to SNMPv2, all SNMPv1 compliant agents must be entirely replaced with SNMPv2 compliant agents-gateways or bilingual managers and proxy agents were not available to support the gradual migration as of early-1995. Since SNMPv1 and SNMPv2 are incompatible with each other and SNMPv2 is not stable, it is important when procuring a managed device to determine which network management protocol(s) is supported.

Alternatives

Common Management Information Protocol (CMIP) may be a better alternative for large, complex networks or security-critical networks.

CMIP is similar to SNMP and was developed to address SNMP's shortcomings. However, CMIP takes significantly more system resources than SNMP, is difficult to program, and is designed to run on the ISO protocol stack [X.700 96]. (However, the technology standard used today in most systems is TCP/IP.)

The biggest feature in CMIP is that an agent can perform tasks or trigger events based upon the value of a variable or a specific condition. For example, when a computer can not reach its network fileserver for a predetermined number of times, an event can be generated to notify the appropriate personnel [Vallillee 96]. With SNMP, this task would have to be performed by a user, because an SNMP agent does not analyze information.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Simple Network Management Protocol
Application category	Protocols (AP.2.2.3) Network Management (AP.2.2.2)
Quality measures category	Maintainability (QM.3.1) Simplicity (QM.3.2.2) Complexity (QM.3.2.1) Efficiency/ Resource Utilization (QM.2.2) Scalability (QM.4.3) Security (QM.2.1.5)
Computing reviews category	Network Operations (C.2.3) Distributed Systems (C.2.4)

References and Information Sources

[Broadhead 95] Broadhead, Steve. "SNMP Too Simple for Security?" *Secure Computing* (April 1995): 24-29.

[DES 93] Federal Information Processing Standards Publication 46-2 DATA ENCRYPTION STANDARD, 1993 [online]. Available WWW <URL: http://csrc.ncsl.nist.gov/fips/fips46-2.txt> (1996).

[Feit 94] Feit, Sidnie. *A Guide to Network Management*. New York, NY: McGraw Hill, 1994.

[Henderson 95] Henderson and Erwin. "SNMP Version 2: Not So Simple." *Business Communications Review 25,* 5 (May 1995): 44-48.

[Herman 94] Herman, James. "Network Computing Inches Forward." *Business Communications Review 24,* 5 (May 1994): 45-50.

[IETF 96] Internet Engineering Task Force home page [online]. Available WWW <URL: http://www.ietf.cnri.reston.va.us/> (1996).

[Kapoor 94] Kapoor, K. "SNMP Platforms: What's Real, What Isn't." *Data Communications International 23,* 12 (September 1994): 115-18.

[Lake 96] Lake, Craig. *Simple Network Management Protocol (SNMP)* [online]. Available WWW <URL: http://www.sei.cmu.edu/str/docs/SNMP.html> (1996).

[MIB 96] *Development of an MIB for http* [online]. Available WWW <URL: http://http-mib.onramp.net/bof/> (1996).

[Moorhead 95] Moorhead, R.J. & Amirthalingam, K. "SNMP- An Overview of its Merits and Demerits," 180-3. *Proceedings of the Twenty-Seventh Southeastern Symposium on System Theory*. Starkvill, MS, March 12-14, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Phifer 94] Phifer, L.A. "Tearing Down the Wall: Integrating ISO and Internet Management." *Journal of Network and Systems Management 2*, 3 (September 1994): pp. 317-22.

[RFC 96] Postel T. *User Datagram Protocol* (RFC 768) [online]. Available WWW <URL: http://ds.internic.net/rfc/rfc768.txt> (1996).

[Rose 94] Rose, Marshall T. *The Simple Book: An Introduction to Internet Management.* Englewood Cliffs, NJ: Prentice-Hall, 1994.

[SNMP 98] *Simple Network Management Protocol* [online]. Available WWW <URL: http://www.snmp.com> and <URL: http://www.snmp.com/snmppages.html> (1998).

[SNMP FAQ 98] Simple Network Management Protocol FAQ [online]. Available WWW <URL: http://www.snmp.com/FAQs/snmp-faq-part1.txt> and <URL: http://www.snmp.com/FAQs/snmp-faq-part2.txt> (1998).

[SNMPv1 Specs] The following RFC's identify the major components of SNMPv1 online]. Available WWW <URL: http://www.cis.ohio-state.edu/htbin/rfc/rfcXXXX.html> (1996).

Historical

RFC 1156 - Management Information Base Network Management of TCP/IP based internets

RFC 1161 - SNMP over OSI

Informational

RFC 1215 - A Convention for Defining Traps for use with the SNMP

RFC 1270 - SNMP Communication Services

RFC 1303 - A Convention for Describing SNMP-based Agents

RFC 1470 - A Network Management Tool Catalog

Standard and Draft

RFC 1089 - SNMP over Ethernet

RFC 1140 - IAB Official Protocol Standards
RFC 1155 - Structure and Identification of Management Information for TCP/IP based internets.
RFC 1157 - A Simple Network Management Protocol
RFC 1158 - Management Information Base Network Management of TCP/IP based internets: MIB-II
RFC 1187 - Bulk Table Retrieval with the SNMP
RFC 1212 - Concise MIB Definitions
RFC 1213 - Management Information Base for Network Management of TCP/IP-based internets: MIB-II
RFC 1224 - Techniques for Managing Asynchronously-Generated Alerts
RFC 1418 - SNMP over OSI
RFC 1419 - SNMP over AppleTalk
RFC 1420 - SNMP over IPX

[SNMPv2 Specs]

The following RFC's identify the major components of SNMPv2 online]. Available WWW <URL: http://www.cis.ohio-state.edu/htbin/rfc/rfcXXXX.html> (1996).

Historical

RFC 1441 - Introduction to SNMP v2
RFC 1442 - SMI For SNMP v2
RFC 1443 - Textual Conventions for SNMP v2
RFC 1444 - Conformance Statements for SNMP v2
RFC 1445 - Administrative Model for SNMP v2
RFC 1446 - Security Protocols for SNMP v2
RFC 1447 - Party MIB for SNMP v2
RFC 1448 - Protocol Operations for SNMP v2
RFC 1449 - Transport Mappings for SNMP v2
RFC 1450 - MIB for SNMP v2
RFC 1451 - Manager to Manager MIB
RFC 1452 - Coexistence between SNMP v1 and SNMP v2

Draft

RFC 1902 - SMI for SNMPv2
RFC 1903 - Textual Conventions for SNMPv2
RFC 1904 - Conformance Statements for SNMPv2
RFC 1905 - Protocol Operations for SNMPv2
RFC 1906 - Transport Mappings for SNMPv2
RFC 1907 - MIB for SNMPv2
RFC 1908 - Coexistence between SNMPv1 and SNMPv2

Experimental

RFC 1901 - Introduction to Community-based SNMPv2

RFC 1909 - An Administrative Infrastructure for SNMPv2
RFC 1910 - User-based Security Model for SNMPv2

[Stallings 93] Stallings, William. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards.* Reading, MA: Addison-Wesley, 1993.

[Vallillee 96] Vallillee, Tyler. *SNMP & CMIP: An Introduction To Network Management* [online]. Available WWW <URL: http://www.inforamp.net/~kjvallil/t/snmp.html> (1996).

[Wellens 96] Wellens, Chris & Auerbach, Karl. "Towards Useful Management" [online]. *The Quarterly Newsletter of SNMP Technology, Comment, and Events(sm) 4,* 3 (July 1996). Available WWW <URL: http://www.iwl.com/Press/thefuture.html> (1996).

[X.700 96] *X.700 and Other Network Management Services* [online]. Available WWW <URL: http://ganges.cs.tcd.ie/4ba2/x700/index.html> (1996).

Current Author/Maintainer

Dan Plakosh, SEI

External Reviewers

Craig Meyers, SEI
Patrick Place, SEI

Modifications

16 Jan 98: Changes included

- Increased the consistency of terminology
- Minor change to the SNMPv1 architecture figure
- Updated status of SNMPv2 and added information about other SNMP versions
- Clarified some areas
- Updated references

19 Jun 97: Changes included

- Creating an overview technical description on network management, which includes overview material and figures applicable to all network management techniques
- Clarifying the discussion of SNMPv1 and SNMPv2
- Minor changes to the SNMPv1 architecture figure
- Increased the consistency of terminology
- added many new references

10 Jan 97 (original); author for this version: Cory Vondrak, TRW, Redondo Beach, CA

Footnotes

[1] The IETF is a large open community of network designers, operators, vendors, and researchers whose purpose is to coordinate the operation, management and evolution of the Internet, and to resolve short- and mid-range protocol and architectural issues. It is a major source of proposed protocol standards which are submitted to the Internet Engineering Steering Group for final approval. The IETF meets three times a year and extensive minutes of the plenary proceedings are issued.

[2] The IAB is a technical advisory group of the Internet Society. The IAB provides oversight of the architecture for the protocols and procedures used by the Internet, the process used to create Internet Standards and serves as an appeal board for complaints of improper execution of the standards process.

[3] Managed objects: a characteristic of a managed device that can be monitored, modified or controlled.

[4] This organization is responsible for approving U.S. standards in many areas, including computers and communications. Standards approved by this organization are often called ANSI standards (e.g., ANSI C is the version of the C language approved by ANSI).

[5] There is an MIB being developed for http [MIB 96], and the MIB for mail monitoring is now a proposed standard.

URL: http://www.sei.cmu.edu/str/descriptions/snmp_body.html
Last Modified: 24 July 2008

References and Information Sources

[RFC 96] Postel T. *User Datagram Protocol* (RFC 768) [online]. Available WWW <URL: http://ds.internic.net/rfc/rfc768.txt> (1996).

Notes

[3] Managed objects: a characteristic of a managed device that can be monitored, modified or controlled.

References and Information Sources

[Lake 96] Lake, Craig. *Simple Network Management Protocol (SNMP)* [online]. Available WWW <URL: http://www.sei.cmu.edu/str/docs/SNMP.html> (1996).

References and Information Sources

[Moorhead 95] Moorhead, R.J. & Amirthalingam, K. "SNMP- An Overview of its Merits and Demerits," 180-3. *Proceedings of the Twenty-Seventh Southeastern Symposium on System Theory*. Starkvill, MS, March 12-14, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

References and Information Sources

[DES 93] Federal Information Processing Standards Publication 46-2 DATA ENCRYPTION STANDARD, 1993 [online]. Available WWW <URL: http://csrc.ncsl.nist.gov/fips/fips46-2.txt> (1996).

Notes

[4] This organization is responsible for approving U.S. standards in many areas, including computers and communications. Standards approved by this organization are often called ANSI standards (e.g., ANSI C is the version of the C language approved by ANSI).

References and Information Sources

[Broadhead 95] Broadhead, Steve. "SNMP Too Simple for Security?" *Secure Computing* (April 1995): 24-29.

References and Information Sources

[SNMP FAQ 98]	Simple Network Management Protocol FAQ [online]. Available WWW <URL: http://www.snmp.com/FAQs/snmp-faq-part1.txt> and <URL: http://www.snmp.com/FAQs/snmp-faq-part2.txt> (1998).

References and Information Sources

[Wellens 96]	Wellens, Chris & Auerbach, Karl. "Towards Useful Management" [online]. *The Quarterly Newsletter of SNMP Technology, Comment, and Events(sm) 4*, 3 (July 1996). Available WWW <URL: http://www.iwl.com/Press/thefuture.html> (1996).

Notes

[5] There is an MIB being developed for http [MIB 96], and the MIB for mail monitoring is now a proposed standard.

References and Information Sources

[X.700 96] *X.700 and Other Network Management Services* [online]. Available WWW <URL: http://ganges.cs.tcd.ie/4ba2/x700/index.html> (1996).

References and Information Sources

[Vallillee 96] Vallillee, Tyler. *SNMP & CMIP: An Introduction To Network Management* [online]. Available WWW <URL: http://www.inforamp.net/~kjvallil/t/snmp.html> (1996).

References and Information Sources

[MIB 96] *Development of an MIB for http* [online]. Available WWW <URL: http://http-mib.onramp.net/bof/> (1996).

References and Information Sources

[Harry 00] Harry, Mikel. "Six Sigma: The Breakthrough Management Strategy Revolutionizing the World's Top Corporations." New York, N.Y. Random House Publishers, 2000.

References and Information Sources

[Arnold 99] Arnold, Paul V. *Pursuing the Holy Grail* [online]. Available WWW <URL: http://www.progressivedistributor.com/mro/archives/editorials/editJJ1999.html> (1999).

References and Information Sources

[Harrold 99] Harrold, Dave. *Designing for Six Sigma Capability* [Online]. Available WWW <URL: http://www.controleng.com/archives/1999/ctl0101.99/01a103.htm> (1999).

References and Information Sources

[Pyzdek 01] Pyzdek, Thomas. *The Six Sigma Handbook.* New York, N.Y.: McGraw-Hill Professional Publishing, 2001.

References and Information Sources

[ASQ 00] ASQ Statistics Division. *Improving Performance Through Statistical Thinking.* Milwaukee, WI: ASQ Quality Press, 2000.

References and Information Sources

[ASA 01] American Statistical Association, Quality & Productivity Section. *Enabling Broad Application of Statistical Thinking* [online]. Available WWW <URL: http://web.utk.edu/~asaqp/thinking.html> (2001).

References and Information Sources

[Bylinsky 98] Bylinsky, Gene. *How to Bring Out Better Products Faster* [online]. Available WWW <URL: http://www.amsup.com/media/fortune.htm> (1998).

References and Information Sources

[Pyzdek 2-01] Pyzdek, Thomas. *Six Sigma and Beyond: Why Six Sigma Is Not TQM* [online]. Available WWW <URL: http://www.qualitydigest.com/feb01/html/sixsigma.html> (2001).

References and Information Sources

[Marash 99] Marash, Stanley A. *Six Sigma: Passing Fad or a Sign of Things to Come?* [online]. Available WWW <URL: http://www.thesamgroup.com/sixsigmaarticle.htm> (1999).

Simplex Architecture

Software Technology Roadmap

Status

complete

Purpose and Origin

Real-time applications that play a mission-critical role are prevalent throughout the DoD and industry. The complexity of these systems make them expensive to design, *maintain*, and support. Their mission critical nature requires assurance of operational *availability*. These systems are often *safety-critical*, requiring a high degree of *reliability*. The long life cycles of these systems usually result in multiple capability upgrades as well as platform migrations. As the use of COTS products increases, upgrade cycles will become shorter.

Simplex architecture is a paradigm and an engineering framework that permits the quick, easy, and reliable insertion of new capabilities and technologies into mission critical real-time systems [Sha 96]. Simplex is the synthesis of selected best practices in several technology areas that support the safe, online upgrade of hardware and software, in spite of residual errors in the new components. Through the use of Simplex, it becomes possible to shift resources from static design and extensive testing to reliable incremental evolution.

Technical Detail

Software is pervasive within the critical systems that form the infrastructure of modern society, both military and civilian. These systems are often large and complex and require periodic and extensive upgrading. The important technical problems include the following:

- *Integration of new and revised components.* The need for periodic and extensive upgrading and technology refreshment of systems challenges developers to integrate new or changed components into systems without compromising the strict reliability and availability requirements of the applications. There are significant strategic and tactical advantages afforded by the ability to adapt quickly to changing situations. These potential advantages challenge developers to find ways of modifying, upgrading, or adding system components more quickly while reducing the possibility of error.
- *Vendor driven upgrade.* To cut costs and gain leverage from technical

advances in the commercial sector, the DoD has encouraged more frequent use of COTS components in its software. For similar reasons, industry is often following suit. COTS components have a short life cycle (roughly one year.) DoD platforms change at a much slower rate and typically have longer life cycles (often 25-30 years or more). This make the DoD platform susceptible to a problem that occurs when the vendor releases a new version of the COTS component. The upgrade can either be ignored or incorporated into the system. Ignoring it will eventually result in a system that is burdened with unsupported and obsolete components. Incorporating it forces the DoD platform to change on a schedule determined by the vendor, rather than the system developer, maintainer, or customer. New releases usually add features and fix existing bugs, but in the process they also often introduce new bugs. So upgrading is risky; a way to manage the risk is needed.

- *Upgrade paradox.* The upgrade paradox results from the use of replication or functional redundancy and majority voting. A minority upgrade will have no effect because it will be voted out of the system by the majority. A majority upgrade with residual errors can cause the system to fail.

Collectively, these technical problems present a formidable challenge to the developers and maintainers of systems with long life cycles.

Simplex is a framework for system integration and evolution. It integrates a number of technologies, including:

- *Analytic Redundancy.* These technologies are used for integrated availability and reliability management. They employ sophisticated monitoring and switching logic which includes a *simple leadership protocol.* Analytic redundancy allows high-performance, but possibly less-reliable, components to be used in systems demanding a high degree of reliability. This is accomplished without sacrificing the performance and reliability levels provided by existing highly reliable components.
- *Replaceable Units.* These technologies (dynamic binding) allow the replacement of software modules at runtime without having to shut down and restart the system.
- *Publish/Subscribe.* These are flexible real-time group communication technologies that allow components to dynamically publish and subscribe to needed information [Rajkumar 95].
- *Rate Monotonic Scheduling.* These technologies for real-time computing (see Rate Monotonic Analysis) allow components to be replaced or modified in real time, transparently to the applications, while still meeting deadlines. These technologies are integrated into the real-time operating system.

The above technologies are shown in the context of the overall structure of a Simplex-based application in Figure 33.

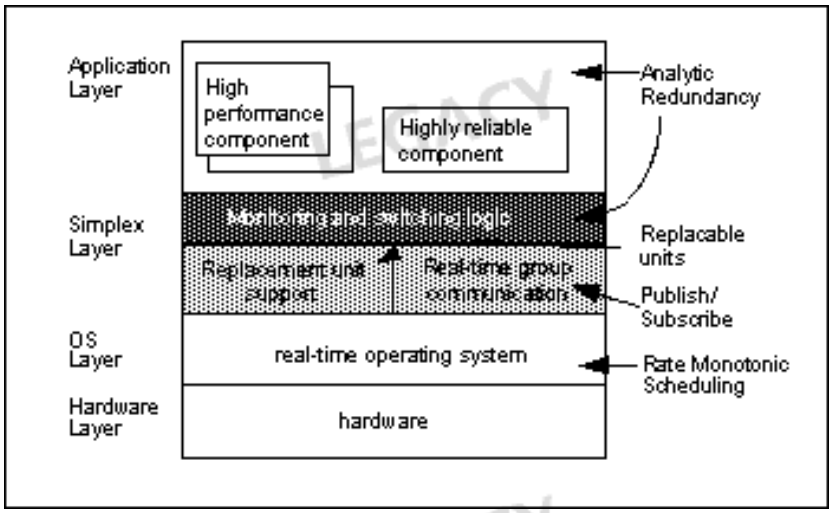


Figure 33: Simplex Technologies and Architecture

Figure 34 is a highly simplified view of the data flow in a system using Simplex. Notice that multiple versions of a component are employed-a *Highly Reliable Component* (HRC) and a *High Performance Component* (HPC). The HRC might be legacy software designed to control the device. It has known performance characteristics and presumably, due to long use, is relatively bug free. If we suppose that the HPC is a new version of the software with improved performance characteristics, but possibly also containing bugs since it has not yet been used extensively, the following scenario takes place.

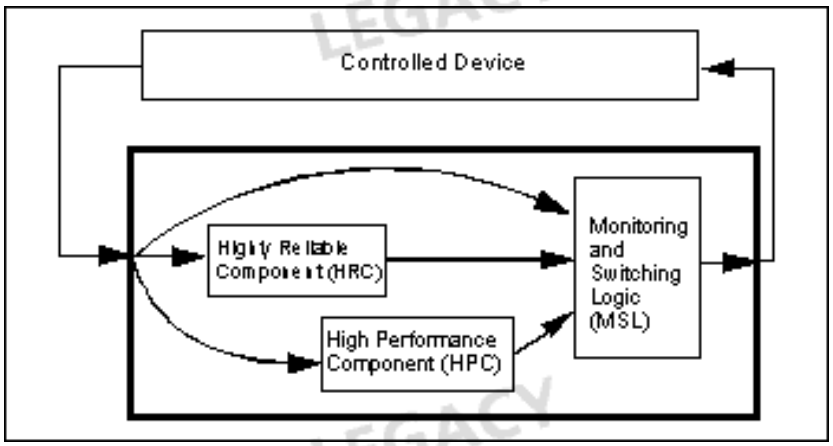


Figure 34: Simplex: Simplified Data Flow

The device under control is sampled at a regular interval. The data is processed by both HRC and HPC. Instead of controlling the device directly, a *simple leadership protocol* is used. Under this protocol, both modules send their results to the *Monitoring and Switching Logic* (MSL), which also uses inputs obtained from the device under control to decide which output to pass back to the device. As long as HPC is behaving properly, it is the leader and its output will be transmitted to the device. Should MSL decide that HPC is not behaving correctly, it makes the HRC the leader and uses its output instead. Thus the device will perform no worse than it did before the upgrade to HPC occurred. This solves the upgrade paradox even in the presence of multiple alternatives because at any instant only the output of one of the alternatives is used. Not

shown, for reasons of complexity, is the module that would actually remove a failed HPC from the system and allow it to be replaced with a corrected version for another try.

Usage Considerations

Simplex is most suitable for systems that have high availability and reliability requirements. It seems especially suitable for systems such as control systems (real-time or process) whose behavior can be modeled and monitored.

Because Simplex is relatively immature, pilot studies will be needed to determine its suitability for any intended application. This would involve developing a rapid prototype, using Simplex, of a simplified instance of the intended application.

Maturity

The safe, online upgrade of both software and hardware, including COTS components, using Simplex has been successfully demonstrated in the laboratory. Simplex is being transitioned into practice via several pilot studies:

- *Silicon Wafer Manufacturing*. The objective was to demonstrate the use of Simplex as the basis for the control architecture in manufacturing process-control software. This was a joint effort between the Software Engineering Institute and the Department of Electrical and Computer Engineering at Carnegie Mellon, guided by engineers from SEMATECH.
- *NSSN (new attack submarine program).* This study involved a US Navy program whose goal is the development, demonstration, and transition of a COTS-based fault-tolerant submarine control system that can be upgraded inexpensively and dependably.
- *INSERT (INcremental Software Evolution for Real-Time Systems)*. This project was funded by the Air Force/DARPA EDCS (evolutionary design of complex software) program, whose goal is to evaluate the possible use of Simplex in the context of onboard avionics systems. Work is proceeding with Lockheed-Martin Tactical Aircraft Systems to investigate the application of this technology to the automated maneuvering capability of the F-16 fighter.

Costs and Limitations

Simplex is designed to support the evolution of mission-critical systems that have high availability or reliability requirements. Its suitability for management information systems (e.g., MIS) applications that do not have such requirements has yet to be determined. Its usefulness in C4I systems is currently being investigated.

Although Simplex has been designed to reduce the life-cycle cost of systems, data on its impact on system life-cycle cost is not available at this time. Much of Simplex is built upon COTS components such as a POSIX compliant real-time operating system running on modern hardware. This tends to reduce costs relative to custom designs.

When using Simplex, engineering costs are increased by the need to analyze and create the analytically redundant modules. Additionally, there is some overhead involved in the operation of the monitoring and switching logic. Finally, the need to run multiple copies of an application (i.e., the HRC and HPC simultaneously) requires additional resources-at the very least additional memory and CPU cycles. These factors tend to have an upward effect on costs-compensated for by the increased reliability and flexibility which Simplex provides.

A perhaps more important consideration is the savings that Simplex provides by reducing the required testing and downtime when installing an upgraded component. The expectation is that the use of Simplex will provide a significant savings in total life-cycle cost.

Complementary Technologies

Software and hardware reliability modeling and analysis allow users to estimate the impact of Simplex on system reliability. System life-cycle cost estimation techniques will allow users to estimate the cost impact.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of Technology	Simplex Architecture
Application category	Reapply Software Life Cycle (AP.1.9.3) Reengineering (AP.1.9.5) Software Architecture (AP.2.1) Restart/Recovery (AP.2.10)
Quality measures category	Availability/Robustness (QM.2.1.1) Reliability (QM.2.1.2) Safety (QM.2.1.3) Real-time Responsiveness/Latency (QM.2.2.2) Maintainability (QM.3.1)
Computing reviews category	Fault-tolerance (D.4.5) Real-time and embedded systems (D.4.7) Network communication (D.4.4)

References and Information Sources

[Altman 97] Altman, Neal. *The Simplex Architecture* [online]. Available WWW <URL: http://www.sei.cmu.edu/simplex/simplex_architecture.html> (May 6, 1997).

[Sha 96] Sha, L.; Rajkumar, R.; & Gagliardi, M. "Evolving Dependable Real Time Systems," 335-346. *Proceedings of the 1996 IEEE Aerospace Applications Conference*. Aspen, CO, February 3-10, 1996. New York, NY: IEEE Computer Society Press, 1996.

[Rajkumar 95] Rajkumar, R.; Gagliardi, M.; & Sha, L. "The Real-Time Publisher/Subscriber Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation," 66-75. *The First IEEE Real-Time Technology and Applications Symposium*. Chicago, IL, May 15-17, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

Current Author/Maintainer

Charles B. Weinstock, SEI
Lui R. Sha, SEI

External Reviewers

John Lehoczky, Professor, Statistics Department, CMU

Modifications

29 Oct 97 changes include:

· Updated list of pilot studies.

· Provided additional detail on constituent technologies.

· Added application architecture diagram.

· Improved data flow diagram and enhanced the explanation.

· Added additional information on anticipated costs (where these are generally understood.)

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/simplex_body.html
Last Modified: 24 July 2008

Related Topics

Restart/Recovery (AP.2.10)

- Simplex Architecture

Related Topics

Safety (QM.2.1.3)

- Simplex Architecture

References and Information Sources

[Fagan 76] Fagan, M. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal 15*, 3 (1976): 182-211.

Notes

[1] Capability Maturity Model and CMM are service marks of Carnegie Mellon University.

References and Information Sources

[Ebenau 94] Ebenau, Robert G. & Strauss, Susan H. *Software Inspection Process*. New York, NY: McGraw-Hill, 1994.

References and Information Sources

[O'Neill 01a] O'Neill, Don. *Peer Reviews*. Encyclopedia of Software Engineering. New York, New York: Wiley Publishing, Inc., to appear 2001.

References and Information Sources

[O'Neill 88] O'Neill, Don & Ingram, Albert L. "Software Inspections Tutorial," 92-120. *Software Engineering Institute Technical Review 1988*. Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute, 1988.

References and Information Sources

[O'Neill 92] O'Neill, Don. "Software Inspections: More Than a Hunt for Errors." Crosstalk, Journal Of Defense Software Engineering 30 (January 1992): 8-10.

References and Information Sources

[Linger 79] Linger, R.C.; Mills, H.D.; & Witt, B.I. Structured Programming: Theory and Practice. Reading, MA: Addison-Wesley, 1979.

References and Information Sources

[Freedman 90] Freedman, D.P. & Weinberg, G.M. *Handbook of Walkthroughs, Inspections, and Technical Reviews*. New York, NY: Dorset House, 1990.

References and Information Sources

[O'Neill 01c] O'Neill, Don. Return on Investment Tool [online]. Available WWW <URL: http://members.aol.com/ONeillDon/nsqe-roi.html> (2001).

References and Information Sources

[Basili/Boehm 01] Basili, Vic, & Barry Boehm. "Software Defect Reduction Top 10 List." *Computer 34*,1, (January 2001): 135-137.

References and Information Sources

[O'Neill 89] O'Neill, Don. *Software Inspections Course and Lab*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1989.

References and Information Sources

[**O'Neill 95,96,00**] O'Neill, Don. "National Software Quality Experiment: Results 1992-1999." Software Technology Conference, Salt Lake City, 1995, 1996, and 2000.

References and Information Sources

[O'Neill 01b] O'Neill, Don. Software Inspection Measurements and Derived Metrics Tool [online]. Available WWW <URL: http://members.aol.com/ONeillDon/nsqe-assessment.html> (2001).

References and Information Sources

[Humphrey 89] Humphrey, Watts S. Managing the Software Process. Reading, MA: Addison-Wesley, 1989.

References and Information Sources

[Sundaram 96] Sundaram, Aurobindo. *An Introduction to Intrusion Detection* [online]. Available WWW <URL: http://www.acm.org/crossroads/xrds2-4/xrds2-4.html> (1996).

References and Information Sources

[Teng 90] Teng, Henry S.; Chen, Kaihu; & Lu, Stephen C. "Security Audit Trail Analysis Using Inductively Generated Predictive Rules," 24-29. *Sixth Conference on Artificial Intelligence Applications*. Santa Barbara, CA, May 5-9, 1990. Los Alamitos, CA: IEEE Computer Society Press, 1990.

References and Information Sources

[Lunt 93] Lunt, Teresa F. "A Survey of Intrusion Detection Techniques." *Computers and Security 12*, 4 (June 1993): 405-418.

References and Information Sources

[Smaha 88] Smaha, Stephen E. "Haystack: An Intrusion Detection System," 37-44. *Proceedings of the Fourth Aerospace Computer Security Applications Conference*. Orlando, Florida, December 12-16, 1988. Washington, DC: IEEE Computer Society Press, 1989.

Notes

[1] See http://www.research.att.com for more details.

References and Information Sources

[Florac 99] William A. Florac and Anita D. Carleton, Measuring the Software Process: Statistical Process Control for Software Process Improvement, Addison &endash;Wesley, 1999.

References and Information Sources

[Wheeler 92] Donald J. Wheeler and David S. Chambers, Understanding Statistical Process Control, Second Edition, SPC Press, Knoxville, TN, 1992.

References and Information Sources

[Paige 93] Paige, Emmett. *Selection of Migration Systems* ASD (C3I) Memorandum. Washington, DC: Department of Defense, November 12, 1993.

References and Information Sources

[TAFIM 94] U.S. Department Of Defense. *Technical Architecture Framework For Information Management (TAFIM)* Volumes 1-8, Version 2.0. Reston, VA: DISA Center for Architecture, 1994. Also available [online] WWW <URL: http://www-library.itsi.disa.mil/tafim/tafim.html> (1996).

TAFIM Reference Model

Software Technology Roadmap

Status

Advanced

Note

We recommend Reference Models, Architectures, Implementations- An Overview as prerequisite reading for this technology.

Purpose and Origin

The Technical Architectural Framework for Information Management (TAFIM) reference model was developed by the Defense Information Systems Agency (DISA) to guide the evolution of Department of Defense (DoD) systems, including sustaining base, strategic, and tactical systems, as well as interfaces to weapon systems. Application of the TAFIM reference model is required on most DoD systems [Paige 93]. TAFIM is a set of services, standards, design components, and configurations that are used in design, implementation, and enhancement of information management system architectures. The intent is that the DoD infrastructure will have a common architecture that will, over time, be a fully *flexible* and *interoperable* enterprise. Details on the TAFIM model are available in a seven volume TAFIM document, but are primarily in Volume 3 [TAFIM 94].

Technical Detail

The TAFIM reference model (Figure 27) describes services (functionality) needed within each of the model's components. It contains a set of general principles on how components and component services relate to each other. This model is designed to enhance transition from legacy applications to a distributed environment. TAFIM addresses the following six software components:

1. *Application software*. Application software consists of mission area applications and support applications. Mission area applications may be custom-developed software, commercial-off-the-shelf (COTS) products, or Non-developmental items (NDI). Support applications are building blocks for mission area applications. They manage processing for the communication environment and can be shared by multiple mission and support applications. Common COTS support applications include multimedia, communications, business processing, environment management, database utilities, and engineering support (analysis, design, modeling, development, and simulation) capabilities.
2. *Application platform*. Application platform consists of hardware services and software

services, including operating system, real-time monitoring program, and peripheral drivers. Application software must access platform resources by a request across Application Programming Interfaces (APIs) to ensure integrity and consistency. A platform service may be realized by a single process shared by a group of applications, or by a distributed system with portions of an application operating on separate processors. Application platform services include software engineering, user interface, data management, data interchange, graphic, network, and operating system capabilities.

3. *Application platform cross-area services*. Application platform cross-area services are services that have a direct effect on the operation of one or more of the functional areas. Application platform cross-area services include culturally-related application environments, security, system administration and distributed computing capabilities.
4. *External environment*. The external environment supports system and application interoperability and user and data portability. The external environment interface specifies a complete interface between the application platform and underlying external environment. The external environment includes human-computer interaction, information services, and communication capabilities.
5. *TAFIM application program interface (API)*.The API is the interface between an application and a service that resides on a platform. The API specifies how a service is invoked- without specifying its implementation- so that the implementation may be changed without causing a change in the applications that use that API. The API makes the platform transparent to the application. A platform may be a single computer or a network of hosts, clients, and servers where distributed applications are implemented. A service invoked through an API can reside on the same platform as the requesting application, on a different platform, or on a remote platform. APIs are defined for mission and support applications and platform services. APIs are generally required for platform services such as compilers, window management, data dictionaries, database management systems, communication protocols, and system management utilities.
6. *TAFIM external environment interface*. The TAFIM external environment interface (which could be considered and API) is between the application platform and the external environment. This interface allows the exchange of information. It supports system and application software interoperability. User and data portability are directly provided by the external environment interface.

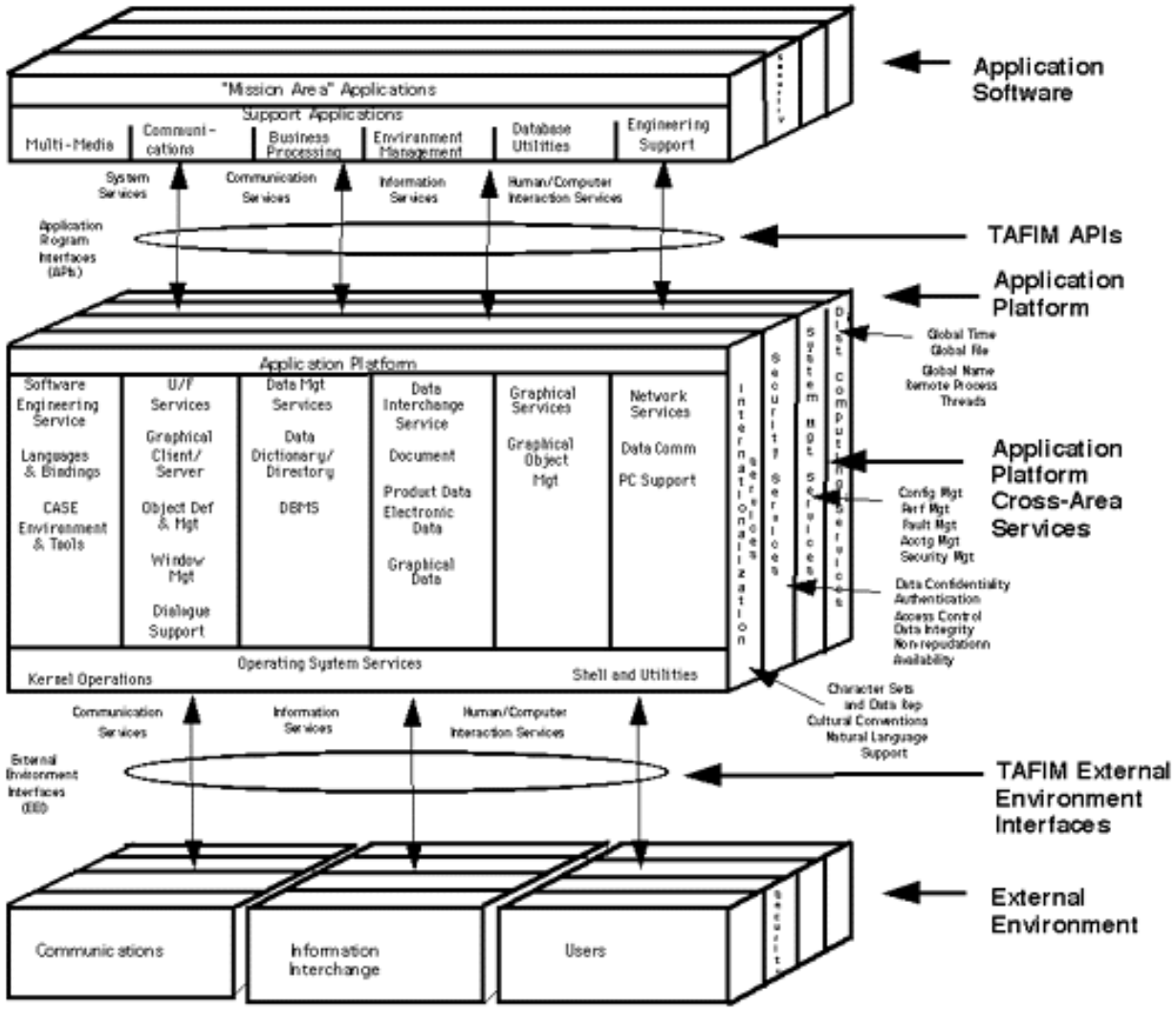


Figure 27: DoD TAFIM Technical Reference Model

Usage Considerations

The TAFIM reference model is applicable to most information systems, including sustaining base, strategic, and tactical systems, as well as interfaces to weapon systems [TAFIM 94]. It is mandatory for use on most DoD programs [Paige 93]. However, systems built using the reference model have been criticized by Rear Adm. John Gauss, the Interoperability Chief at DISA, when speaking on systems in the field in Bosnia: "We have built a bunch of state-of-the-art, open-systems, TAFIM-compliant stove-pipes" [Temin 96]. TAFIM-compliant means that the applicable standards and guidelines are met for the implemented component services. This suggests that even when complying with the TAFIM reference model, problems of interoperability are not necessarily resolved. The Joint Technical Architecture (JTA) provides a set of standards and guidelines for C4I systems, specifically in the area of interoperability, that supersedes TAFIM Volume 7 [JTA 96].

There are TAFIM-compliant software products available for use when implementing a TAFIM-based architecture in areas such as support applications, communication services, business process services, environment management, and engineering services. Additional products exist or are being developed in areas such as user interface, data management, data

interchange, graphics, operating systems, internationalization, security system management, and distributed computing.

Maturity

The latest version of TAFIM, Version 2.0, was published in 1994. DoD organizations and contractors have been applying this set of guidelines to current and future information systems. The Defense Information Infrastructure Common Operating Environment is an implementation of TAFIM. This COE is currently being used by the Global Command and Control System (GCCS) and the Global Combat Support System (GCSS). The Air Force Theater Battle Management Core System (TBMCS) is also required to comply with the TAFIM and use the COE. It may take several years, after multiple new TAFIM-compliant systems are in the field, to determine the effectiveness of the reference model with respect to achieving a common, flexible, and interoperable DoD infrastructure.

Costs and Limitations

The TAFIM reference model does not fully specify components and component connections [Clements 96]. It does not dictate the specific components for implementation. (No reference model prescribes implementation solutions.) TAFIM does provide the guidance necessary to improve commonality among DoD information technical architectures.

One contractor has found that there is no cost difference in using the TAFIM reference model (as compared to any other reference model) when designing and implementing a software architecture. This is based on the fact that application of a reference model is part of the standard design and implementation practice.

Dependencies

The TAFIM reference model is dependent on the evolution of component and service standards that apply specifically to software; it may be affected by computer platforms and network hardware as well.

Alternatives

Under conditions where the TAFIM reference model is not required, an alternative model would be the Reference Model for Frameworks of Software Engineering Environments (known as the ECMA model [ECMA 93]) that is promoted in Europe and used commercially and worldwide. Commercially-available Hewlett-Packard products use this model [HP 96]. Another alternative would be the Common Object Request Broker Architecture (CORBA) if the design called for object-oriented infrastructure .

Complementary Technologies

Open systems (see COTS and Open Systems-An Overview) would be a complementary technology to TAFIM because work done in open system supports the TAFIM goals of achieving interoperable systems.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	TAFIM Reference Model
Application category	Software Architecture Models (AP.2.1.1) Distributed Computing (AP.2.1.2)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1)
Computing reviews category	Distributed Systems (C.2.4) Software Engineering Design (D.2.10)

References and Information Sources

[Clements 96]	Clements, Paul C. & Northrop, Linda M. *Software Architecture: An Executive Overview* (CMU/SEI-96-TR-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
[ECMA 93]	*Reference Model for Frameworks of Software Engineering Environments*, 3rd Edition (NIST Special Publication 500-211/Technical Report ECMA TR/55). Prepared jointly by NIST and the European Computer Manufacturers Association (ECMA). Washington, DC: U.S. Government Printing Office, 1993.
[HP 96]	*Integrated Solutions Catalog for the SoftBench Product Family*. Palo Alto, CA: Hewlett-Packard, 1996.
[JTA 96]	U.S. Department of Defense. *Joint Technical Architecture (JTA)* [online]. Available WWW <URL: http://www-jta.itsi.disa.mil/>(1996).
[Paige 93]	Paige, Emmett. *Selection of Migration Systems* ASD (C3I) Memorandum. Washington, DC: Department of Defense, November 12, 1993.
[TAFIM 94]	U.S. Department Of Defense. *Technical Architecture Framework For Information Management (TAFIM)* Volumes 1-8, Version 2.0. Reston, VA: DISA Center for Architecture, 1994. Also available [online] WWW <URL: http://www-library.itsi.disa.mil/tafim/tafim.html> (1996).
[Temin 96]	Temin, Thomas, ed. "Mishmash at Work (DoD Systems in Bosnia are not Interoperable)." *Government Computer News 15, 7* (April 1996): 28.

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Peter Garrabrant, GTE
Tricia Oberndorf, SEI

Modifications

10 Jan 97 (original)

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/tafim_body.html
Last Modified: 24 July 2008

References and Information Sources

[Temin 96] Temin, Thomas, ed. "Mishmash at Work (DoD Systems in Bosnia are not Interoperable)." *Government Computer News 15, 7* (April 1996): 28.

References and Information Sources

[JTA 96] U.S. Department of Defense. *Joint Technical Architecture (JTA)* [online]. Available WWW <URL: http://www-jta.itsi.disa.mil/>(1996).

References and Information Sources

[Clements 96] Clements, Paul C. & Northrop, Linda M. *Software Architecture: An Executive Overview* (CMU/SEI-96-TR-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

References and Information Sources

[Humphrey 95] Watts S. Humphrey, A Discipline for Software Engineering, ISBN 0-201-54610-8, Addison-Wesley Publishing Company, Reading, MA, 1995.

References and Information Sources

[Ferguson 97]
Ferguson, P.; Humphrey, W. S.; Khajenoori, S.; Macke, S.; and Matvya, A. "Introducing the Personal Software Process: Three Industry Case Studies." IEEE Computer 30 (May 1997): 24-31.

References and Information Sources

[Humphrey 00] Humphrey, Watts. *The Team Software Process*^{SM} *(TSP*^{SM}*)* (CMU/SEI-2000-TR-023). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2000.

References and Information Sources

[Eckerson 95] Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." *Open Information Systems 10,* 1 (January 1995): 3(20).

Three Tier Software Architectures

Software Technology Roadmap

Status

Complete

Note

We recommend Client/Server Software Architectures as prerequisite reading for this technology description.

Purpose and Origin

The three tier software architecture (a.k.a. three layer architectures) emerged in the 1990s to overcome the limitations of the two tier architecture (see Two Tier Software Architectures). The third tier (middle tier server) is between the user interface (client) and the data management (server) components. This middle tier provides process management where business logic and rules are executed and can accommodate hundreds of users (as compared to only 100 users with the two tier architecture) by providing functions such as queuing, application execution, and database staging. The three tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased *performance*, *flexibility*, *maintainability*, *reusability*, and *scalability*, while hiding the complexity of distributed processing from the user. For detailed information on three tier architectures see Schussel and Eckerson. Schussel provides a graphical history of the evolution of client/server architectures [Schussel 96, Eckerson 95].

The three tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. These characteristics have made three layer architectures a popular choice for Internet applications and net-centric information systems.

Technical Detail

A three tier distributed client/server architecture (as shown in Figure 28) includes

a user system interface top tier where user services (such as session, text input, dialog, and display management) reside.

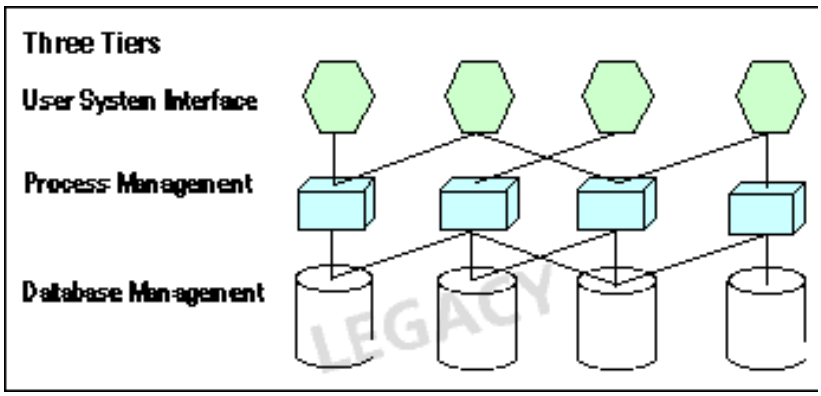


Figure 28: Three tier distributed client/server architecture depiction [Louis 95]

The third tier provides database management functionality and is dedicated to data and file services that can be optimized without using any proprietary database management system languages. The data management component ensures that the data is consistent throughout the distributed environment through the use of features such as data locking, consistency, and replication. It should be noted that connectivity between tiers can be dynamically changed depending upon the user's request for data and services.

The middle tier provides process management services (such as process development, process enactment, process monitoring, and process resourcing) that are shared by multiple applications.

The middle tier server (also referred to as the application server) improves performance, flexibility, maintainability, reusability, and scalability by centralizing process logic. Centralized process logic makes administration and change management easier by localizing system functionality so that changes must only be written once and placed on the middle tier server to be available throughout the systems. With other architectural designs, a change to a function (service) would need to be written into every application [Eckerson 95].

In addition, the middle process management tier controls transactions and asynchronous queuing to ensure reliable completion of transactions [Schussel 96]. The middle tier manages distributed database integrity by the two phase commit process (see Database Two Phase Commit). It provides access to resources based on names instead of locations, and thereby improves scalability and flexibility as system components are added or moved [Edelstein 95].

Sometimes, the middle tier is divided in two or more unit with different functions, in these cases the architecture is often referred as multi layer. This is the case, for example, of some Internet applications. These applications typically have light clients written in HTML and application servers written in C++ or Java, the gap between these two layers is too big to link them together. Instead, there is an intermediate layer (web server) implemented in a scripting language. This

layer receives requests from the Internet clients and generates html using the services provided by the business layer. This additional layer provides further isolation between the application layout and the application logic.

It should be noted that recently, mainframes have been combined as servers in distributed architectures to provide massive storage and improve security (see Distributed/Collaborative Enterprise Architectures).

Usage Considerations

Three tier architectures are used in commercial and military distributed client/server environments in which shared resources, such as heterogeneous databases and processing rules, are required [Edelstein 95]. The three tier architecture will support hundreds of users, making it more scalable than the two tier architecture (see Two Tier Software Architectures) [Schussel 96].

Three tier architectures facilitate software development because each tier can be built and executed on a separate platform, thus making it easier to organize the implementation. Also, three tier architectures readily allow different tiers to be developed in different languages, such as a graphical user interface language or light internet clients (HTML, applets) for the top tier; C, C++, SmallTalk, Basic, Ada 83, or Ada 95 for the middle tier; and SQL for much of the database tier [Edelstein 95].

Migrating a legacy system to a three tier architecture can be done in a manner that is low-risk and cost-effective. This is done by maintaining the old database and process management rules so that the old and new systems will run side by side until each application and data element or object is moved to the new design. This migration might require rebuilding legacy applications with new sets of tools and purchasing additional server platforms and service tools, such as transaction monitors (see Transaction Processing Monitor Technology) and Message-Oriented Middleware. The benefit is that three tier architectures hide the complexity of deploying and supporting underlying services and network communications.

Maturity

Three tier architectures have been used successfully since the early 1990s on thousands of systems of various types throughout the Department of Defense (DoD) and in commercial industry, where distributed information computing in a heterogeneous environment is required. An Air Force system that is evolving from a legacy architecture to a three tier architecture is Theater Battle Management Core System (TBMCS). Multi tier architectures have been widely and successfully applied in some of the biggest Internet servers.

Costs and Limitations

Building three tier architectures is complex work. Programming tools that support the design and deployment of three tier architectures do not yet provide all of the desired services needed to support a distributed computing environment.

A potential problem in designing three tier architectures is that separation of user interface logic, process management logic, and data logic is not always obvious. Some process management logic may appear on all three tiers. The placement of a particular function on a tier should be based on criteria such as the following [Edelstein 95]:

- ease of development and testing
- ease of administration
- scalability of servers
- performance (including both processing and network load)

Dependencies

Database management systems must conform to X/Open systems standards and XA Transaction protocols to ensure distributed database integrity when implementing a heterogeneous database two phase commit.

Alternatives

Two tier client server architectures (see Two Tier Software Architectures) are appropriate alternatives to the three tier architectures under the following circumstances:

- when the number of users is expect to be less than 100
- for non-real-time information processing in non-complex systems that requires minimal operator intervention

Distributed/collaborative enterprise computing (see Distributed/Collaborative Enterprise Architectures) is seen as a viable alternative, particularly if object-oriented technology on an enterprise-wide scale is desired. An enterprise-wide design is comprised of numerous smaller systems or subsystems.

Although three tier architecture has proven sound, the supporting products implementing the architecture are not as mature as other competing technologies. Transaction Monitors (TM) are a valid alternative when reliability and scalability requirements can not be fulfilled with existing multi layer

technology. Although TMs don't support modern development paradigms like Object Orientation (OO) they are still quite useful when massive scalability and robustness is needed.

Complementary Technologies

Complementary technologies to three tier architectures are Object-Oriented Design (to implement decomposable applications), three tier client/server architecture tools, and Database Two Phase Commit processing.

For communication between potentially distributed layers some middleware is needed. This middleware can be a Remote Procedure Call (RPC) mechanism or a Message-Oriented Middleware (MOM), depending on whether synchronous or asynchronous communication is preferred.

The middle tier encapsulates business logic. Some of this logic is application specific but a significant percentage is organization or even domain wide. Domain Engineering and Domain Analysis can be used to capture this inter-application commonality and create a set of assets that can be effectively reused in different application.

It should be noted that recently, mainframes have been combined as servers in distributed architectures to provide massive storage and improve security (see Distributed/Collaborative Enterprise Architectures).

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Three Tier Software Architectures
Application category	Client/Server (AP.2.1.2.1)
Quality measures category	Maintainability (QM.3.1) Scalability (QM.4.3) Reusability (QM.4.4) Reliability (QM.2.1.2)
Computing reviews category	Distributed Systems (C.2.4) Software Engineering Design (D.2.10)

References and Information Sources

[Dickman 95] Dickman, A. "Two-Tier Versus Three-Tier Apps." *Informationweek* 553 (November 13, 1995): 74-80.

[Eckerson 95] Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." *Open Information Systems 10*, 1 (January 1995): 3(20).

[Edelstein 95] Edelstein, Herb. "Unraveling Client Server Architectures." *DBMS 7*, 5 (May 1994): 34(7).

[Gallaugher 96] Gallaugher, J. & Ramanathan, S. "Choosing a Client/Server Architecture. A Comparison of Two-Tier and Three-Tier Systems." *Information Systems Management Magazine 13*, 2 (Spring 1996): 7-13.

[Louis 95] *Louis* [online]. Available WWW <URL: http://www.softis.is> (1995).

[Newell 95] Newell, D.; Jones, O.; & Machura, M. "Interoperable Object Models for Large Scale Distributed Systems," 30-31. *Proceedings. International Seminar on Client/Server Computing*. La Hulpe, Belgium, October 30-31, 1995. London, England: IEE, 1995.

[Schussel 96] Schussel, George. *Client/Server Past, Present, and Future* [online]. Formerly Available WWW <URL: http://news.dci.com/geos/dbsejava.htm> (1995).

Current Author/Maintainer

Darleen Sadoski, GTE
Santiago Comella-Dorda, SEI

External Reviewers

Paul Clements, SEI
Frank Rogers, GTE

Modifications

16 Feb 2000: Inclusion of multi-layer architectures and net-centric systems.

10 Jan 1997 (original)

URL: http://www.sei.cmu.edu/str/descriptions/threetier_body.html
Last Modified: 24 July 2008

References and Information Sources

[Louis 95] *Louis* [online]. Available WWW <URL: http://www.softis.is> (1995).

References and Information Sources

[Edelstein 95] Edelstein, Herb. "Unraveling Client Server Architectures." *DBMS 7*, 5 (May 1994): 34 (7).

About The STR

Software Technology Roadmap

- Background
- Target Audiences
- Sponsors & Contributors

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/about/index.html
Last Modified: 24 July 2008

Taxonomies

Software Technology Roadmap

- About the Taxonomies
- View the Application Taxonomy
- View the Quality Measures Taxonomy
- Search the Taxonomies

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/taxonomies/index.html
Last Modified: 24 July 2008

Glossary & Indexes

Software Technology Roadmap

- Glossary
- Keyword Index

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/indexes/index.html
Last Modified: 24 July 2008

References and Information Sources

[Dickman 95] Dickman, A. "Two-Tier Versus Three-Tier Apps." *Informationweek* 553 (November 13, 1995): 74-80.

References and Information Sources

[Hudson 94] Hudson, D. & Johnson, J. *Client-Server Goes Business Critical*. Dennis, MA: The Standish Group International, 1994.

Transaction Processing Monitor Technology

Software Technology Roadmap

Status

Advanced

Note

We recommend Client/Server Software Architectures *as prerequisite reading for this technology description.*

Purpose and Origin

Transaction processing (TP) monitor technology provides the distributed client/server environment the capacity to *efficiently* and *reliably* develop, run, and manage transaction applications.

TP monitor technology controls transaction applications and performs business logic/rules computations and database updates. TP monitor technology emerged 25 years ago when Atlantic Power and Light created an online support environment to share concurrently applications services and information resources with the batch and time sharing operating systems environment. TP monitor technology is used in data management, network access, security systems, delivery order processing, airline reservations, and customer service. Use of TP monitor technology is a cost-effective alternative to upgrading database management systems or platform resources to provide this same functionality. Dickman and Hudson provide more details on TP monitor technology [Dickman 95, Hudson 94].

Technical Detail

TP monitor technology is software that is also referred to as Middleware. It can provide application services to thousands of clients in a distributed client/server environment. TP monitor technology does this by multiplexing client transaction requests (by type) onto a controlled number of processing routines that support particular services. These events are depicted in Figure 37.

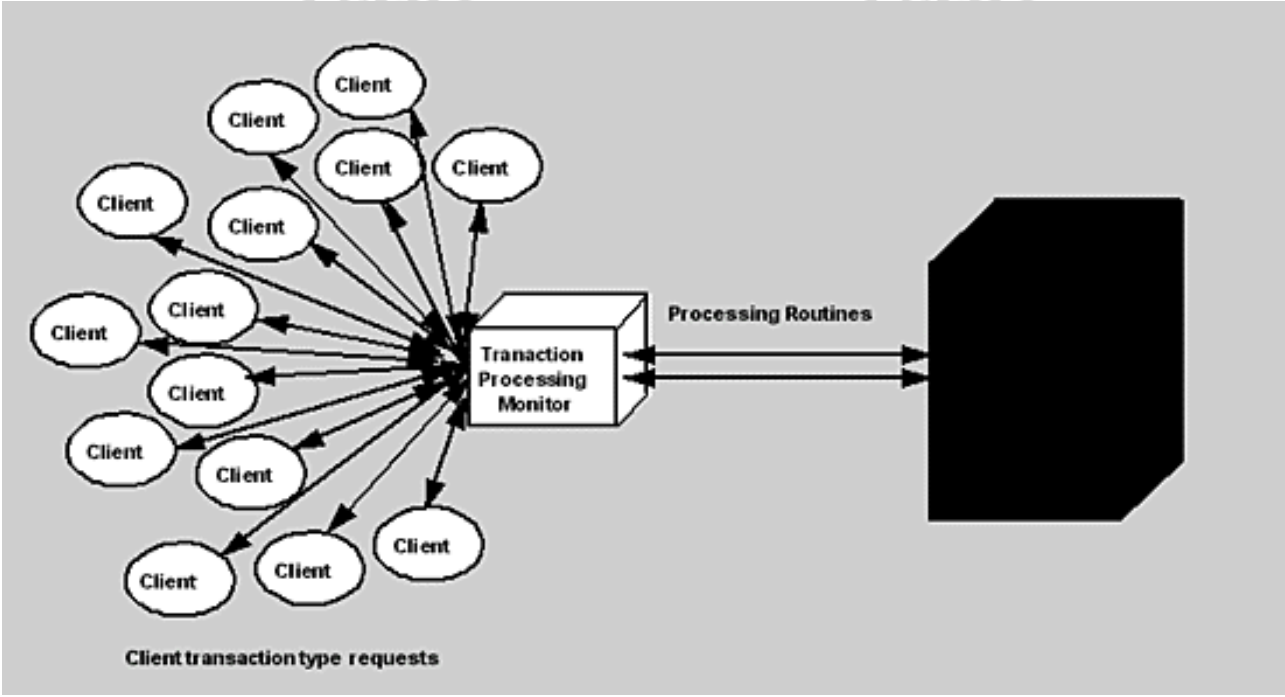


Figure 37: Transaction Processing Monitor Technology

Clients are bound, serviced, and released using stateless servers that minimize overhead. The database sees only the controlled set of processing routines as clients [Dickman 95, Hudson 94].

TP monitor technology maps numerous client requests through application services routines to improve system performance. The TP monitor technology (located as a server) can also take the application transitions logic from the client. This reduces the number of upgrades required by these client platforms. In addition, TP monitor technology includes numerous management features, such as restarting failed processes, dynamic load balancing, and enforcing consistency of distributed data. TP monitor technology is easily scalable by adding more servers to meet growing numbers of users [Dickman 95, Hudson 94].

TP monitor technology is independent of the database architecture. It supports flexible and robust business modeling and encourages modular, *reusable* procedures. TP monitor designs allow Application Programming Interfaces (APIs) to support components such as heterogeneous client libraries, databases and resource managers, and peer-level application systems. TP monitor technology supports architecture *flexibility because* each component in a distributed system is comprised of products that are designed to meet specific functionality, such as graphical user interface builders and database engines [Dickman 95, Hudson 94].

Usage Considerations

Within distributed client/server systems, each client that is supported adds overhead to system resources (such as memory). Responsiveness is improved and system resource overhead is reduced by using TP monitor technology to multiplex many clients onto a much smaller set of application service routines. TP monitor technology provides a highly active system that includes services for delivery order processing, terminal and forms management, data management, network access, authorization, and security.

TP monitor technology supports a number of program-to-program communication models, such as store-and-forward, asynchronous, Remote Procedure Call (RPC), and conversational. This improves interactions among application components. TP monitor technology provides the ability to construct complex business applications from modular, well-defined functional components. Because this technology is well-known and well-defined it should reduce program risk and associated costs [Dickman 95, Hudson 94].

Maturity

TP monitor technology has been used successfully in the field for 25 years. TP monitor technology is used for delivery order processing, hotel and airline reservations, electronic fund transfers, security trading, and manufacturing resource planning and control. It improves batch and time-sharing application effectiveness by creating online support to share application services and information resources [Dickman 95, Hudson 94].

Costs and Limitations

TP monitor technology makes database processing cost-effective for online applications. Spending relatively little money on TP monitor technology can result in significant savings compared to the resources required to improve database or platform resources to provide the same functionality [Dickman 95].

A limitation to TP technology is that the implementation code is usually written in a lower-level language (such as COBOL), and is not yet widely available in the popular visual toolsets [Schussel 96].

Alternatives

A variation of TP monitor technology is session based technology. In the TP monitor technology, transactions from the client are treated as messages. In the session based technology, a single server provides both database and transaction services. In session based technology, the server must be aware of clients in advance to maintain each client's processing thread. The session server must constantly send messages to the client (even when work is not being done in the client) to ensure that the client is still alive. Session based architectures are not as scalable because of the adverse effect on network performance as the number of clients grow.

Another alternative to TP monitor technology is remote data access (RDA). The RDA centers the application in a client computer, communicating with back-end database servers. Clients can be network-intensive, but scalability is limited.

A third alternative to TP monitor technology is the database server approach, which provides functions (usually specific to the database) and is architecturally locked to the specific database system [Dickman 95, Hudson 94].

Complementary Technologies

Complementary technologies include mainframe client/server software architectures (see Mainframe Server Software Architectures) and Three Tier Software Architectures; in both cases

the TP monitor technology could server as the middle tier.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Transaction Processing Monitor Technology
Application category	Client/Server (AP.2.1.2.1) Client/Server Communication (AP.2.2.1)
Quality measures category	Efficiency/ Resource Utilization (QM.2.2) Reusability (QM.4.4) Maintainability (QM.3.1)
Computing reviews category	Distributed Systems (C.2.4)

References and Information Sources

[Dickman 95]	Dickman, A. "Two-Tier Versus Three-Tier Apps." *Informationweek* 553 (November 13, 1995): 74-80.
[Hudson 94]	Hudson, D. & Johnson, J. *Client-Server Goes Business Critical*. Dennis, MA: The Standish Group International, 1994.
[Schussel 96]	Schussel, George. *Client/Server Past, Present, and Future* [online]. Available WWW <URL: http://www.dciexpo.com/geos/> (1995).
[TP 96]	*TP Lite vs. TP Heavy* [online]. Available WWW <URL: http://www.byte.com/art/9504/sec11/art4.htm> (1996).

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

David Altieri, GTE

Modifications

10 Jan 97 (original)

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/descriptions/tpmt_body.html
Last Modified: 24 July 2008

References and Information Sources

[Schussel 96] Schussel, George. *Client/Server Past, Present, and Future* [online]. Originally available WWW <URL: http://www.dciexpo.com/geos/> (1995).

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

Trusted Operating Systems

Software Technology Roadmap

Status

Advanced

Note

We recommend Computer System Security--An Overview as prerequisite reading for this technology description.

Purpose and Origin

Trusted operating systems provide the basic security mechanisms and services that allow a computer system to protect, distinguish, and separate classified data. Trusted operating systems have been developed since the early 1980s and began to receive National Security Agency (NSA) evaluation in 1984.

Technical Detail

Trusted operating systems lower the security risk of implementing a system that processes classified data. Trusted operating systems implement security policies and accountability mechanisms in an operating system package. A security policy is the rules and practices that determine how sensitive information is managed, protected, and distributed [Abrams 95]. Accountability mechanisms are the means of identifying and tracing who has had access to what data on the system so they can be held accountable for their actions.

Trusted operating systems are evaluated by the NSA National Computer Security Center (NCSC) against a series of six requirements-level classes listed in the table below. C1 systems have basic capabilities. A1 systems provide the most capability. The higher the rating level is, the wider the range of classified data is that may be processed.

Table 10 below shows the NCSC Evaluation Criteria Classes.

Table 10: NCSC Evaluation Criteria Classes

Class	Title	Number of Approved Operating Systems in this Class [TPEP 96]
A1	Verified Design	0
B3	Security Domains	1
B2	Structured Protection	1
B1	Labeled Security Protection	7
C2	Controlled Access Protection	5
C1	Discretionary Security Protection	No Longer Evaluated

A low level (C1 and C2) system provides limited discretionary access controls and identification and authentication mechanisms. Discretionary access controls identify who can have access to system data based on the need to know. Mandatory access controls identify who or what process can have access to data based on the requester having formal clearance for the security level of the data. A low-level system is used when the system only needs to be protected against human error and it is unlikely that a malicious user can gain access to the system.

A higher level (B2, B3, and A1) system provides complete mandatory and discretionary access control, thorough security identification of data devices, rigid control of transfer of data and access to devices, and complete auditing of access to the system and data. These higher level systems are used when the system must be protected against a malicious user's abuse of authority, direct probing, and human error [Abrams 95].

The portion of the trusted operating system that grants requesters access to data and records the action is frequently called the reference monitor because it refers to an authorization database to determine if access should be granted. Higher level trusted operating systems are used in MLS hosts and compartmented mode workstations (see Computer System Security- an Overview for overview information).

Usage Considerations

Trusted operating systems must be used to implement multi-level security systems and to build security guards that allow systems of different security levels to be connected to exchange data. Use of a trusted operating system may

be the only way that a system can be networked with other high security systems. Trusted operating systems may be required if a C4I system processes intelligence data and provides data to war fighters. Department of Defense (DoD) security regulations define what evaluation criteria must be satisfied for a multi-level system based on the lowest and highest classification of the data in a system and the clearance level of the users of the system. Using an NCSC-evaluated system reduces accreditation cost and risk. The security officer identified as the Designated Approving Authority (DAA) for secure computer systems has the responsibility and authority to review and approve the systems to process classified information. The DAA will require analysis and tests of the system to assure that it will operate securely. The DAA can accept the NCSC evaluation of a system rather than generating the data. For a B3 or A1 system, that can represent a savings of 1 to 2 years in schedule and the operating system will provide a proven set of functions.

Maturity

This technology has been implemented by several vendors for commercial-off-the-shelf (COTS) use in secure systems. As of September 1996, the NCSC Evaluated Product List indicated that fourteen operating systems have been evaluated as level C2, B1,B2, and B3 systems in the last three years [TPEP 96]. The number of operating systems evaluated by class (excluding evaluations of updated versions of operating systems) is included in the table. Use of one of the approved trusted operating systems can result in substantial cost and schedule reductions for a system development effort and provide assurance that the system can be operated securely.

Costs and Limitations

The heavy access control and accounting associated with high security systems can affect system performance; as such, higher performance processors, I/O, and interfaces may be required. Trusted operating systems have unique interfaces and operating controls that require special security knowledge to use and operate. Frequently COTS products that operate satisfactorily with a standard operating system must be replaced or augmented to operate with a trusted operating system.

Dependencies

Trusted operating systems at B2 and above enable the development of system interoperability for systems at different security levels and allow applications to perform data fusion. They are dependent on a trusted computing base that provides secure data paths and protected memory.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Trusted Operating Systems
Application category	Trusted Operating Systems (AP.2.4.1)
Quality measures category	Security (QM.2.1.5)
Computing reviews category	Operating System Security and Protection (D.4.6) Computer-Communications Network Security Protection (C.2.0)

References and Information Sources

[Abrams 95] Abrams, Marshall D.; Jajodia, Sushil; & Podell, Harold J. *Information Security An Integrated Collection of Essays.* Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Russel 91] Russel, Deborah & Gangemi, G.T. Sr. *Computer Security Basics.* Sebastopol, CA: O'Reilly & Associates, Inc., 1991.

[TPEP 96] *Trusted Product Evaluation Program Evaluated Product List* [online]. Available WWW <URL: http://www.radium.ncsc.mil/tpep/index.html> (1996).

[White 96] White, Gregory B.; Fisch, Eric A.; & Pooch, Udo W. *Computer System and Network Security.* Boca Raton, FL: CRC Press, 1996.

Current Author/Maintainer

Tom Mills, Lockheed Martin

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/trusted_body.html
Last Modified: 24 July 2008

Related Topics

Trusted Operating Systems (AP.2.4.1)

- Trusted Operating Systems

Two Tier Software Architectures

Software Technology Roadmap

Status

Complete

Note

We recommend Client/Server Software Architectures, as prerequisite reading for this technology description.

Purpose and Origin

Two tier software architectures were developed in the 1980s from the file server software architecture design. The two tier architecture is intended to improve *usability* by supporting a forms-based, user-friendly interface. The two tier architecture improves *scalability* by accommodating up to 100 users (file server architectures only accommodate a dozen users), and improves *flexibility* by allowing data to be shared, usually within a homogeneous environment [Schussel 96]. The two tier architecture requires minimal operator intervention, and is frequently used in non-complex, non-time critical information processing systems. Detailed readings on two tier architectures can be found in Schussel and Edelstein [Schussel 96, Edelstein 94].

Technical Detail

Two tier architectures consist of three components distributed in two layers: client (requester of services) and server (provider of services). The three components are

1. User System Interface (such as session, text input, dialog, and display management services)
2. Processing Management (such as process development, process enactment, process monitoring, and process resource services)
3. Database Management (such as data and file services)

The two tier design allocates the user system interface exclusively to the client. It places database management on the server and splits the processing management between client and server, creating two layers. Figure 38 depicts the two tier software architecture.

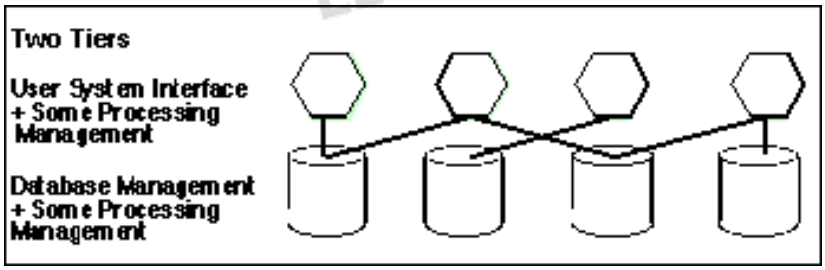


Figure 38: Two Tier Client Server Architecture Design [Louis 95]

In general, the user system interface client invokes services from the database management server. In many two tier designs, most of the application portion of processing is in the client environment. The database management server usually provides the portion of the processing related to accessing data (often implemented in store procedures). Clients commonly communicate with the server through SQL statements or a call-level interface. It should be noted that connectivity between tiers can be dynamically changed depending upon the user's request for data and services.

As compared to the file server software architecture (that also supports distributed systems), the two tier architecture improves flexibility and scalability by allocating the two tiers over the computer network. The two tier improves usability (compared to the file sever software architecture) because it makes it easier to provide a customized user system interface.

It is possible for a server to function as a client to a different server- in a hierarchical client/server architecture. This is known as a chained two tier architecture design.

Usage Considerations

Two tier software architectures are used extensively in non-time critical information processing where management and operations of the system are not complex. This design is used frequently in decision support systems where the transaction load is light. Two tier software architectures require minimal operator intervention. The two tier architecture works well in relatively homogeneous environments with processing rules (business rules) that do not change very often and when workgroup size is expected to be fewer than 100 users, such as in small businesses.

Maturity

Two tier client/server architectures have been built and fielded since the middle to late 1980s. The design is well known and used throughout industry. Two tier architecture development was enhanced by fourth generation languages.

Costs and Limitations

Scalability. The two tier design will scale-up to service 100 users on a network. It appears that beyond this number of users, the performance capacity is exceeded. This is because the client and server exchange "keep alive" messages continuously, even when no work is being done, thereby saturating the network [Schussel 96].

Implementing business logic in stored procedures can limit scalability because as more application logic is moved to the database management server, the need for processing power grows. Each client uses the server to execute some part of its application code, and this will ultimately reduce the number of users that can be accommodated.

Interoperability. The two tier architecture limits interoperability by using stored procedures to implement complex processing logic (such as managing distributed database integrity) because stored procedures are normally implemented using a commercial database management system's proprietary language. This means that to change or interoperate with more than one type of database management system, applications may need to be rewritten. Moreover, database management system's proprietary languages are generally not as capable as standard programming languages in that they do not provide a robust programming environment with testing and debugging, version control, and library management capabilities.

System administration and configuration. Two tier architectures can be difficult to administer and maintain because when applications reside on the client, every upgrade must be delivered, installed, and tested on each client. The typical lack of uniformity in the client configurations and lack of control over subsequent configuration changes increase administrative workload.

Batch jobs. The two tiered architecture is not effective running batch programs. The client is typically tied up until the batch job finishes, even if the job executes on the server; thus, the batch job and client users are negatively affected [Edelstein 94].

Dependencies

Developing a two tier client/server architecture following an object-oriented methodology would be dependent on the CORBA standards for design implementation. See Common Object Request Broker Architecture.

Alternatives

Possible alternatives for two tier client server architectures are

- the three-tier architecture (see Three Tier Software Architectures) if there is a requirement to accommodate greater than 100 users
- distributed/collaborative architectures (see Distributed/Collaborative Enterprise Architectures) if there is a requirement to design on an enterprise-wide scale. An enterprise-wide design is comprised of

numerous smaller systems or subsystems.

When preparing a two tier architecture for possible migration to an alternative three tier architecture, the following five steps will make the transition less costly and of lower risk [Dickman 95]:

1. Eliminate application diversity by ensuring a common, cross-hardware library and development tools.
2. Develop smaller, more comparable service elements, and allow access through clearly-defined interfaces.
3. Use an Interface Definition Language (IDL) to model service interfaces and build applications using header files generated when compiled.
4. Place service elements into separate directories or files in the source code.
5. Increase flexibility in distributed functionality by inserting service elements into Dynamic Linked Libraries (DLLs) so that they do not need to be complied into programs.

Complementary Technologies

Complementary technologies for two tier architectures are CASE (computer-aided software engineering) tools because they facilitate two tier architecture development, and open systems (see COTS and Open Systems-An Overview) because they facilitate developing architectures that improve scalability and flexibility.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Two Tier Software Architectures
Application category	Client/Server (AP.2.1.2.1)
Quality measures category	Usability (QM.2.3) Maintainability (QM.3.1) Scalability (QM.4.3)
Computing reviews category	Distributed Systems (C.2.4) Software Engineering Design (D.2.10)

References and Information Sources

[Dickman 95]	Dickman, A. "Two-Tier Versus Three-Tier Apps." *Informationweek* 553 (November 13, 1995): 74-80.
[Edelstein 94]	Edelstein, Herb. "Unraveling Client/Server Architecture." *DBMS 7*, 5 (May 1994): 34(7).
[Gallaugher 96]	Gallaugher, J. & Ramanathan, S. "Choosing a Client/Server Architecture. A Comparison of Two-Tier and Three-Tier Systems." *Information Systems Management Magazine 13*, 2 (Spring 1996): 7-13.
[Louis 95]	*Louis* [online]. Available WWW <URL: http://www.softis.is> (1995).
[Newell 95]	Newell, D.; Jones, O.; & Machura, M. "Interoperable Object Models for Large Scale Distributed Systems," 30-31. *Proceedings. International Seminar on Client/Server Computing*. La Hulpe, Belgium, October 30-31, 1995. London, England: IEE, 1995.
[Schussel 96]	Schussel, George. *Client/Server Past, Present, and Future* [online]. Available WWW <URL: http://www.dciexpo.com/geos/> (1995).

Current Author/Maintainer

Darleen Sadoski, GTE

External Reviewers

Paul Clements, SEI
Frank Rogers, GTE

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/twotier_body.html
Last Modified: 24 July 2008

References and Information Sources

[Dickman 95] Dickman, A. "Two-Tier Versus Three-Tier Apps." *Informationweek* 553 (November 13, 1995): 74-80.

References and Information Sources

[Denning 90] Denning, Peter J. *Computers Under Attack Intruders, Worms and Viruses*. New York, NY: ACM Press, 1990.

References and Information Sources

[Slade 96b] Slade, Robert. *Reviewing Anti-virus Products* [online]. Available WWW <URL: http://www.bocklabs.wisc.edu/~janda/sladerev.html> (1996).

References and Information Sources

[Slade 96a] Slade, Robert. *Quick Reference Antiviral Review Chart* [online]. Available WWW <URL: http://csrc.ncsl.nist.gov/virus/quickref.rvw> (1996).

Related Topics

Denial of Service (Accessibility) (QM.2.1.4.1.3)

- Virus Detection

References and Information Sources

[Peercy 81] Peercy, David E. "A Software Maintainability Evaluation Methodology." *Transactions on Software Engineering 7*, 7 (July 1981): 343-351.

References and Information Sources

[Bennett 93] Bennett, Brad & Satterthwaite, Paul. "A Maintainability Measure of Embedded Software," 560-565. *Proceedings of the IEEE 1993 National Aerospace and Electronics Conference*. Dayton, OH, May 24-28, 1993. New York, NY: IEEE, 1993.

References and Information Sources

[Oman 92a] Oman, P. & Hagemeister, J. *Construction and Validation of Polynomials for Predicting Software Maintainability* (92-01TR). Moscow, ID: Software Engineering Test Lab, University of Idaho, 1992.

References and Information Sources

[Oman 94] Oman, P. & Hagemeister, J. "Constructing and Testing of Polynomials Predicting Software Maintainability." *Journal of Systems and Software 24*, 3 (March 1994): 251-266.

References and Information Sources

[Oman 91] Oman, P. *HP-MAS: A Tool for Software Maintainability, Software Engineering* (#91-08-TR). Moscow, ID: Test Laboratory, University of Idaho, 1991.

References and Information Sources

[Pearse 95] Pearse, Troy & Oman, Paul. "Maintainability Measurements on Industrial Source Code Maintenance Activities," 295-303. *Proceedings. of the International Conference on Software Maintenance*. Opio, France, October 17-20, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

Maintainability Index Technique for Measuring Program Maintainability

Software Technology Roadmap

Status

Complete

Purpose and Origin

Quantitative measurement of an operational system's *maintainability* is desirable both as an instantaneous measure and as a predictor of maintainability over time. Efforts to measure and track maintainability are intended to help reduce or reverse a system's tendency toward "code entropy" or degraded integrity, and to indicate when it becomes cheaper and/or less risky to rewrite the code than to change it. *Software Maintainability Metrics Models in Practice* is the latest report from an ongoing, multi-year joint effort (involving the Software Engineering Test Laboratory of the University of Idaho, the Idaho National Engineering Laboratory, Hewlett-Packard, and other companies) to quantify maintainability via a Maintainability Index (MI) [Welker 95]. Measurement and use of the MI is a process technology, facilitated by simple tools, that in implementation becomes part of the overall development or maintenance process. These efforts also indicate that MI measurement applied during software development can help reduce lifecycle costs. The developer can track and control the MI of code as it is developed, and then supply the measurement as part of code delivery to aid in the transition to maintenance.

Other studies to define code maintainability in various environments have been done [Peercy 81, Bennett 93], but the set of reports leading to the MI measurement technique offered by Welker [Welker 95] describes a method that appears to be very applicable to today's Department of Defense (DoD) systems.

Technical Detail

The literature of at least the last ten years shows that there have been several efforts to characterize and quantify software maintainability; Maintenance of Operational Systems--An Overview provides a broad overview of software maintenance issues. In this specific technology, a program's maintainability is calculated using a combination of widely-used and commonly-available measures to form a Maintainability Index (MI). The basic MI of a set of programs is a polynomial of the following form (all are based on average-per-code-module

measurement):

171 - 5.2 * ln(aveV) - 0.23 * aveV(g') - 16.2 * ln (aveLOC) + 50 * sin (sqrt(2.4 * perCM))

The coefficients are derived from actual usage (see Usage Considerations). The terms are defined as follows:

aveV = average Halstead Volume V per module (see Halstead Complexity Measures)

aveV(g') = average extended cyclomatic complexity per module (see Cyclomatic Complexity)

aveLOC = the average count of lines of code (LOC) per module; and, optionally

perCM = average percent of lines of comments per module

Oman develops the MI equation forms and their rationale [Oman 92a]; the Oman study indicates that the above metrics are good and sufficient predictors of maintainability. Oman builds further on this work using a modification of the MI and describing how it was calibrated for a specific large suite of industrial-use operational code [Oman 94]. Oman describes a prototype tool that was developed specifically to support capture and use of maintainability measures for Pascal and C [Oman 91]. The aggregate strength of this work and the underlying simplicity of the concept make the MI technique potentially very useful for operational Department of Defense (DoD) systems.

Usage Considerations

Calibration of the equations. The coefficients shown in the equation are the result of calibration using data from numerous software systems being maintained by Hewlett-Packard. Detailed descriptions of how the MI equation was calibrated and used appear in Coleman, Pearse, and Welker [Coleman 94, Coleman, 95, Pearse 95, Welker 95]. The authors claim that follow-on efforts show that this form of the MI equation generally fits other industrial-sized software systems [Oman 94 and Welker 95], and the breadth of the work tends to support this claim. It is advisable to test the coefficients for proper fit with each major system to which the MI is applied.

Effects from comments in code. The user must analyze comment content and quality in the specific system to decide whether the comment term perCM is useful.

Ways of using MI

1. The system can be checked periodically for maintainability, which is also a way of calibrating the equations.
2. It can be integrated into a development effort to screen code quality as it is being built and modified; this could yield potentially significant life cycle cost savings.
3. It can be used to drive maintenance activities by evaluating modules either selectively or globally to find high-risk code.
4. MI can be used to compare or evaluate systems: Comparing the MIs of a known-quality system and a third-party system can provide key information in a make-or-buy decision.

Example of usage. Welker relates how a module containing a routine with some "very ugly" code was assessed as unmaintainable, when expressed in terms of the MI (note that just quantifying the problem is a step forward) [Welker 95]. The module was first redesigned, and then functionally enhanced. The measured results are shown in Table 7:

Table 7: Measured Results

Measure	Initial Code		Restructured Code		After Enhancement	
Code Unit	Routine	Module	Routine	Module	Routine	Module
MI (larger MI = more maintainable)	6.47	33.55	39.93	70.13	37.62	69.60
Halstead Effort[1]	2,216,499	2,233,072	182,216	480,261	201,429	499,474
Extended Cyclomatic Complexity[2]	45	49	18	64	21	67
Lines of Code	622	663	196	732	212	748

[1] Halstead Effort, rather than Halstead Volume, was used in this case study. See Halstead Complexity Measures for more information on both these measures. Generally, the lower a program's measure of effort, the simpler a change to the program will be (because Halstead measures are weighted toward measuring computational complexity, not all programs will behave this way).

[2] Note that a low Cyclomatic Complexity is generally indicative of a lower risk, hence more maintainable, program. In this case, restructuring increased the module complexity slightly (from 49 to 64), but reduced the "ugly" routine's

complexity significantly. In both, the subsequent enhancement drove the complexity slightly higher.

If the enhancement had been made without first doing the restructuring, these figures indicate the change would have been much more risky.

Coleman, Pearse, and Welker provide detailed descriptions of how MI was calibrated and used at Hewlett-Packard [Coleman 94, Coleman 95, Pearse 95, Welker 95].

Maturity

Oman tested the MI approach by using production operational code containing around 50 KLOC to determine the metric parameters, and by checking the results against subjective data gathered using the 1989 AFOTEC maintainability evaluation questionnaire [AFOTEC 89, Oman 94]. Other production code of about half that size was used to check the results, with apparent consistency.

Welker applied the results to analyses of a US Air Force (USAF) system, the Improved Many-On-Many (IMOM) electronic combat modeling system. The original IMOM (in FORTRAN) was translated to C and the C version was later reengineered into Ada. The maintainability of both newer versions was measured over time using the MI approach [Welker 95]. Results were as follows:

- The *reengineered* version's MI was more than twice as high as the original code (larger MI = more maintainable), and declined only slightly over time (note that the original code was not measured over time for maintainability, so change in its MI could not be measured).
- The *translated* baseline's MI was *not* significantly different from the original. This is of special interest to those considering translation, because one of the primary objectives of translation is to reduce future maintenance costs. There was also evidence that the MI of translated code deteriorates more quickly than reengineered code.

Costs and Limitations

Calculating the MI is generally simple and straightforward, given that several commercially-available programming environments contain utilities to count code lines, comment lines, and even Cyclomatic Complexity. Other than the tool described in Oman [Oman 91], tools to calculate Halstead Complexity Measures are less common because the measure is not used as widely. However, once conventions for the counting have been established, it is generally not difficult to write language-specific code scanners to count the Halstead components (operators and operands) and calculate the E and V measures. In relating that removal of unused code in a single module did not affect the MI, Pearse highlights the fact that MI is a system measurement; its parameters are average values [Pearse 95]. However, measuring the MI of individual modules is useful

because changes in either structural or computational complexity are reflected in a module's MI. A product/process measurement program not already gathering the metrics used in MI could find them useful additions. Those metrics already being gathered may be useful in constructing a custom MI for the system. However, it would be advisable to consult the references for their findings on the effectiveness of metrics, other than Halstead E and V and cyclomatic complexity, in determining maintainability.

Dependencies

The MI method depends on the use of Cyclomatic Complexity and Halstead Complexity Measures. To realize the full benefit of MI, the maintenance environment must allow the rewriting of a module when it becomes measurably unmaintainable. The point of measuring the MI is to identify risk; when unacceptably risky code is identified, it should be rewritten.

Alternatives

The process described by Sittenauer is designed to assist in deciding whether or not to reengineer a system [Sittenauer 92]. There are also many research and analytic efforts that deal with maintainability as a function of program structure, design, and content, but none was found that was as clearly appropriate as MI to current DoD systems in the lifecycle phases described in Maintenance of Operational Systems--An Overview.

Complementary Technologies

The test in Sittenauer is meant to verify generally the condition of a system, and would be useful as a periodic check of a software system and to compare to the MI [Sittenauer 92].

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Maintainability Index Technique for Measuring Program Maintainability
Application category	Debugger (AP.1.4.2.4) Test (AP.1.4.3) Unit Testing (AP.1.4.3.4) Component Testing (AP.1.4.3.5) Reapply Software Life Cycle (AP.1.9.3) Reengineering (AP.1.9.5)

Quality measures category	Maintainability (QM.3.1) Testability (QM.1.4.1) Understandability (QM.3.2)
Computing reviews category	Software Engineering Distribution and Maintenance (D.2.7) Software Engineering Metrics (D.2.8) Complexity Classes (F.1.3) Tradeoffs Among Complexity Measures (F.2.3)

References and Information Sources

[AFOTEC 89] *Software Maintainability Evaluation Guide* 800-2, Volume 3. Kirtland AFB, NM: HQ Air Force Operational Test and Evaluation Center (AFOTEC), 1989.

[Ash 94] Ash, Dan, et al. "Using Software Maintainability Models to Track Code Health," 154-160. *Proceedings of the International Conference on Software Maintenance*. Victoria, BC, Canada, September 19-23, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.

[Bennett 93] Bennett, Brad & Satterthwaite, Paul. "A Maintainability Measure of Embedded Software," 560-565. *Proceedings of the IEEE 1993 National Aerospace and Electronics Conference*. Dayton, OH, May 24-28, 1993. New York, NY: IEEE, 1993.

[Coleman 94] Coleman, Don, et al. "Using Metrics to Evaluate Software System Maintainability." *Computer 27,* 8 (August 1994): 44-49.

[Coleman 95] Coleman, Don; Lowther, Bruce; & Oman, Paul. "The Application of Software Maintainability Models in Industrial Software Systems." *Journal of Systems Software 29,* 1 (April 1995): 3-16.

[Oman 91] Oman, P. *HP-MAS: A Tool for Software Maintainability, Software Engineering* (#91-08-TR). Moscow, ID: Test Laboratory, University of Idaho, 1991.

[Oman 92a] Oman, P. & Hagemeister, J. *Construction and Validation of Polynomials for Predicting Software Maintainability* (92-01TR). Moscow, ID: Software Engineering Test Lab, University of Idaho, 1992.

[Oman 92b] Oman, P. & Hagemeister, J. "Metrics for Assessing a Software System's Maintainability," 337-344. *Conference on Software Maintenance 1992*. Orlando, FL, November 9-12, 1992. Los Alamitos, CA: IEEE Computer Society Press, 1992.

[Oman 94] Oman, P. & Hagemeister, J. "Constructing and Testing of Polynomials Predicting Software Maintainability." *Journal of Systems and Software 24*, 3 (March 1994): 251-266.

[Pearse 95] Pearse, Troy & Oman, Paul. "Maintainability Measurements on Industrial Source Code Maintenance Activities," 295-303. *Proceedings. of the International Conference on Software Maintenance*. Opio, France, October 17-20, 1995. Los Alamitos, CA: IEEE Computer Society Press, 1995.

[Peercy 81] Peercy, David E. "A Software Maintainability Evaluation Methodology." *Transactions on Software Engineering 7*, 7 (July 1981): 343-351.

[Sittenauer 92] Sittenauer, Chris & Olsem, Mike. "Time to Reengineer?" *Crosstalk, Journal of Defense Software Engineering* 32 (March 1992): 7-10.

[Welker 95] Welker, Kurt D. & Oman, Paul W. "Software Maintainability Metrics Models in Practice." *Crosstalk, Journal of Defense Software Engineering 8*, 11 (November/December 1995): 19-23.

[Zhuo 93] Zhuo, Fang, et al. "Constructing and Testing Software Maintainability Assessment Models," 61-70. *Proceedings of the First International Software Metrics Symposium*. Baltimore, MD, May 21-22, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

Current Author/Maintainer

Edmond VanDoren, Kaman Sciences, Colorado Springs

External Reviewers

Paul W. Oman, Ph.D., Computer Science Department, University of Idaho, Moscow, ID
Kurt Welker, Lockheed Martin, Idaho Falls, ID

Modifications

10 Jan 97 (original)
12 Mar 02 Correction of Maintainability Index (MI) formula

URL: http://www.sei.cmu.edu/str/descriptions/mitmpm_body.html
Last Modified: 24 July 2008

References and Information Sources

[AFOTEC 89] *Software Maintainability Evaluation Guide* 800-2, Volume 3. Kirtland AFB, NM: HQ Air Force Operational Test and Evaluation Center (AFOTEC), 1989.

Application Taxonomy

Software Technology Roadmap

Readers will use the application taxonomy if they are looking for software technologies that address a particular use, such as design or testing. The technology descriptions have been classified into this taxonomy according to how they are used in systems. Specifically, the application taxonomy divides software technologies into two major categories:

1. Used to support operational systems
2. Used in operational systems

Under the category "Used to Support Operational Systems" (AP.1), by referencing ANSI/IEEE Std 1002-1987 [IEEE 87], we provide the standard life cycle phases plus two major activities that cross all of the phases. IEEE Std 1074-1991 [IEEE 91] helped provide a breakdown of the activities that occur in each life cycle phase. Support in this context means any technology used to develop and maintain an operational system within the life-cycle framework.

The category "Used in Operational Systems" (AP.2) simply provides a breakdown of categories of technologies that are used and operate in operational systems.

▶ View the Application Taxonomy

URL: http://www.sei.cmu.edu/str/taxonomies/ap_tax_body.html
Last Modified: 24 July 2008

References

This frame provides full citations for references.

Quality Measures Taxonomy

Software Technology Roadmap

Readers will use the quality measures taxonomy if they are looking for software technologies that affect particular quality measures or attributes of a software component or system. The technology descriptions have been categorized into this taxonomy by the particular quality measure(s) that they directly influence. Software quality can be defined as the degree to which software possesses a desired combination of attributes (e.g., reliability, interoperability) [IEEE 90]. Software technologies are typically developed to affect certain quality measures.

We developed a reasonably exhaustive an non-overlapping set of measures by which the quality of software is judged. With the help of work done by Boehm, Barbacci, Deutsch and Willis, and Evans and Marciniak, we established a hierarchical relationship among our list of quality measures to create the taxonomy [Boehm 78, Barbacci 95, Deutsch 88, Evans 87]. The following table explains the categories of quality measures and the areas they address:

Quality Measure	Area Addressed
Need Satisfaction (QM.1)	How well does the system meet the user's needs and requirements?
Performance (QM.2)	How well does the system function?
Maintenance (QM.3)	How easily can the system be repaired or changed?
Adaptive (QM.4)	How easily can the system evolve or migrate?
Organizational (QM.5)	none specifically, usually indirect

Categories 1 - 4 are all considered to be direct measures, i.e., quality attributes that can be directly impacted by software technologies. The measures listed in category 5 are measures that generally can not be affected directly by software technologies, but have an indirect relationship. Many factors influence these measures, such as management, politics, bureaucracy, employee skill-level, and work environment. For example, software alone can not improve productivity. A software technology that improves a direct measure such as understandability may indirectly improve productivity. Therefore, most technology descriptions will *not* be categorized into category 5. An example of a technology the reader may find in this category is a technology that was specifically developed to measure or estimate costs of productivity associated with software.

▶ View the Quality Measures Taxonomy

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/taxonomies/qm_tax_body.html
Last Modified: 24 July 2008

References

This frame provides full citations for references.

Related Topics

Parallel Computing (AP.2.1.3)

Parallel Processing Software Architecture

Related Topics

User Interfaces (AP.2.3.1)

Window Managers

Related Topics

Target Operating Systems (AP.2.5)

POSIX
Real-Time Operating Systems

Related Topics

Agents (AP.2.8)

Mediating

Related Topics

Database Utilities (AP.1.4.2.2)

SQL

Related Topics

Create Test Data (AP.1.4.3.1)

Test Data Generation by Chaining

Related Topics

Test Tools (AP.1.4.3.3)

Automatic Test Case Generation
Redundant Test Case Elimination
Statistical Test Plan Generation and Coverage Analysis Techniques
Test and Analysis Tool Generation

Related Topics

Fidelity (QM.2.4)

- Hybrid Automata

Glossary

Software Technology Roadmap

A-H I-P Q-Z

Abstractness
: the degree to which a system or component performs only the necessary functions relevant to a particular purpose.

Acceptance testing
: formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system [IEEE 90].

Accessibility

1. (Denial of Service) the degree to which the software system protects system functions or service from being denied to the user
2. (Reusability) the degree to which a software system or component facilitates the selective use of its components [Boehm 78].

Accuracy
: a quantitative measure of the magnitude of error [IEEE 90].

Acquisition cycle time
: the period of time that starts when a system is conceived and ends when the product meets its initial operational capability.

Adaptability
: the ease with which software satisfies differing system constraints and user needs [Evans 87].

Adaptive maintenance
: software maintenance performed to make a computer program usable in a changed environment [IEEE 90].

Adaptive measures
: a category of quality measures that address how easily a system can evolve or migrate.

Agent
: a piece of software which acts to accomplish tasks on behalf of its user [McGill 96].

Anonymity

the degree to which a software system or component allows for or supports anonymous transactions.

ANSI

American National Standards Institute. This organization is responsible for approving U.S. standards in many areas, including computers and communications. Standards approved by this organization are often called ANSI standards (e.g., ANSI C is the version of the C language approved by ANSI). ANSI is a member of ISO. *See also:* International Organization for Standardization.

Application program interface

a formalized set of software calls and routines that can be referenced by an application program in order to access supporting system or network services [ITS 96].

Architectural design

the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system [IEEE 90].

Artificial intelligence

a subfield within computer science concerned with developing technology to enable computers to solve problems (or assist humans in solving problems) using explicit representations of knowledge and reasoning methods employing that knowledge [DoD 91].

Auditable

the degree to which a software system records information concerning transactions performed against the system.

Availability

the degree to which a system or component is operational and accessible when required for use [IEEE 90].

Capacity

a measure of the amount of work a system can perform [Barbacci 95].

Code

the transforming of logic and data from design specifications (design descriptions) into a programming language [IEEE 90].

Commonality

the degree to which standards are used to achieve interoperability.

Communication software

software concerned with the representation, transfer, interpretation, and processing of data among computer systems or networks. The meaning assigned to the data must be preserved during these operations.

Compactness

the degree to which a system or component makes efficient use of its data storage space- occupies a small volume.

Compatibility
: the ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment [IEEE 90].

Completeness
: the degree to which all the parts of a software system or component are present and each of its parts is fully specified and developed [Boehm 78].

Complexity
1. (Apparent) the degree to which a system or component has a design or implementation that is difficult to understand and verify [IEEE 90].
2. (Inherent) the degree of complication of a system or system component, determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, and the types of data structures [Evans 87].

Component testing
: testing of individual hardware or software components or groups of related components [IEEE 90].

Concept phase
: the initial phase of a software development project, in which the user needs are described and evaluated through documentation (for example, statement of needs, advance planning report, project initiation memo, feasibility studies, system definition, documentation, regulations, procedures, or policies relevant to the project) [IEEE 90].

Conciseness
: the degree to which a software system or component has no excessive information present.

Confidentiality
: the nonoccurrence of the unauthorized disclosure of information [Barbacci 95].

Consistency
: the degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component [IEEE 90].

Corrective maintenance
: maintenance performed to correct faults in hardware or software [IEEE 90].

Correctness
: the degree to which a system or component is free from faults in its specification, design, and implementation [IEEE 90].

Cost estimation
: the process of estimating the "costs" associated with software development projects, to include the effort, time, and labor required.

Cost of maintenance
: the overall cost of maintaining a computer system to include the costs associated with personnel, training, maintenance control, hardware and software maintenance, and requirements growth.

Cost of operation
: the overall cost of operating a computer system to include the costs associated with personnel, training, and system operations.

Cost of ownership
: the overall cost of a computer system to an organization to include the costs associated with operating and maintaining the system, and the lifetime of operational use of the system.

Data management security
: the protection of data from unauthorized (accidental or intentional) modification, destruction, or disclosure [ITS 96].

Data management
: the function that provides access to data, performs or monitors the storage of data, and controls input/output operations [McDaniel 94].

Data recording
: to register all or selected activities of a computer system. Can include both external and internal activity.

Data reduction
: any technique used to transform data from raw data into a more useful form of data. For example, grouping, summing, or averaging related data [IEEE 90].

Database administration
: the responsibility for the definition, operation, protection, performance, and recovery of a database [IEEE 90].

Database design
: the process of developing a database that will meet a user's requirements. The activity includes three separate but dependent steps: conceptual database design, logical database design, and physical database design [IEEE 91].

Database
1. a collection of logically related data stored together in one or more computerized files. Note: Each data item is identified by one or more keys [IEEE 90].
2. an electronic repository of information accessible via a query language interface [DoD 91].

Denial of service
: the degree to which a software system or component prevents the interference or disruption of system services to the user.

Dependability

that property of a computer system such that reliance can justifiably be placed on the service it delivers [Barbacci 95].

Design phase
the period of time in the software life cycle during which the designs for architecture, software components, interfaces, and data are created, documented, and verified to satisfy requirements [IEEE 90].

Detailed design
the process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented [IEEE 90].

Distributed computing
a computer system in which several interconnected computers share the computing tasks assigned to the system [IEEE 90].

Domain analysis
the activity that determines the common requirements within a domain for the purpose of identifying reuse opportunities among the systems in the domain. It builds a domain architectural model representing the commonalities and differences in requirements within the domain (problem space) [ARC 96].

Domain design
the activity that takes the results of domain analysis to identify and generalize solutions for those common requirements in the form of a Domain-Specific Software Architecture (DSSA). It focuses on the problem space, not just on a particular system's requirements, to design a solution (solution space) [ARC 96].

Domain engineering
the process of analysis, specification and implementation of software assets in a domain which are used in the development of multiple software products [SEI 96]. The three main activities of domain engineering are: domain analysis, domain design, and domain implementation [ARC 96].

Domain implementation
the activity that realizes the reuse opportunities identified during domain analysis and design in the form of common requirements and design solutions, respectively. It facilitates the integration of those reusable assets into a particular application [ARC 96].

Effectiveness
the degree to which a system's features and capabilities meet the user's needs.

Efficiency
the degree to which a system or component performs its designated functions with minimum consumption of resources (CPU, Memory, I/O, Peripherals, Networks) [IEEE 90].

Error handling
: the function of a computer system or component that identifies and responds to user or system errors to maintain normal or at the very least degraded operations.

Error proneness
: the degree to which a system may allow the user to intentionally or unintentionally introduce errors into or misuse the system.

Error tolerance
: the ability of a system or component to continue normal operation despite the presence of erroneous inputs [IEEE 90].

Evolvability
: the ease with which a system or component can be modified to take advantage of new software or hardware technologies.

Expandability
: see Extendability [IEEE 90].

Extendability
: the ease with which a system or component can be modified to increase its storage or functional capacity [IEEE 90].

Fail safe
: pertaining to a system or component that automatically places itself in a safe operating mode in the event of a failure [IEEE 90].

Fail soft
: pertaining to a system or component that continues to provide partial operational capability in the event of certain failures [IEEE 90].

Fault tolerance
: the ability of a system or component to continue normal operation despite the presence of hardware or software faults [IEEE 90].

Fault
: an incorrect step, process, or data definition in a computer program [IEEE 90].

Fidelity
: the degree of similarity between a model and the system properties being modeled [IEEE 90].

Flexibility
: the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed [IEEE 90].

Functional scope
: the range or scope to which a system component is capable of being applied.

Functional testing

testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. Synonym: black-box testing [IEEE 90].

Generality

the degree to which a system or component performs a broad range of functions [IEEE 90].

Graphics

methods and techniques for converting data to or from graphic display via computers [McDaniel 94].

Hardware maintenance

the cost associated with the process of retaining a hardware system or component in, or restoring it to, a state in which it can perform its required functions.

Human Computer Interaction

a subfield within computer science concerned with the design, evaluation, and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them [Toronto 95].

Human engineering

the extent to which a software product fulfills its purpose without wasting user's time and energy or degrading their morale [Boehm 78].

Implementation phase

the period of time in the software life cycle during which a software product is created from design documentation and debugged [IEEE 90].

Incompleteness

the degree to which all the parts of a software system or component are not present and each of its parts is not fully specified or developed.

Information Security

the concepts, techniques, technical measures, and administrative measures used to protect information assets from deliberate or inadvertent unauthorized acquisition, damage, disclosure, manipulation, modification, loss, or use [McDaniel 94].

Installation and checkout phase

the period of time in the software life cycle during which a software product is integrated into its operational environment and tested in this environment to ensure it performs as required [IEEE 90].

Integration testing

testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them [IEEE 90].

Integrity

the degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data [IEEE 90].

Interface testing

testing conducted to evaluate whether systems or components pass data and control correctly to one another [IEEE 90].

Interfaces design

the activity concerned with the interfaces of the software system contained in the software requirements and software interface requirements documentation. Consolidates the interface descriptions into a single interface description of the software system [IEEE 91].

Interoperability

the ability of two or more systems or components to exchange information and to use the information that has been exchanged [IEEE 90].

ISO

International Organization for Standardization. A voluntary, non-treaty organization founded in 1946 which is responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of the 89 member countries, including ANSI for the U.S.

Latency

the length of time it takes to respond to an event [Barbacci 95].

Lifetime of operational capability

the total period of time in a system's life that it is operational and meeting the user's needs.

Maintainability

the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment [IEEE 90].

Maintenance control

the cost of planning and scheduling hardware preventive maintenance, and software maintenance and upgrades, managing the hardware and software baselines, and providing response for hardware corrective maintenance.

Maintenance measures

a category of quality measures that address how easily a system can be repaired or changed.

Maintenance personnel

the number of personnel needed to maintain all aspects of a computer system, including the support personnel and facilities needed to support that activity.

Managed device

any type of node residing on a network, such as a computer, printer or routers that contain a management agent.

Managed object

a characteristic of a managed device that can be monitored, modified or controlled.

Management agent
: software that resides in a managed device that allows the device to be monitored and/or controlled by a network management application.

Manufacturing phase
: the period of time in the software life cycle during which the basic version of a software product is adapted to a specified set of operational environments and is distributed to a customer base [IEEE 90].

Model
: an approximation, representation, or idealization of selected aspects of the structure, behavior, operation, or other characteristics of a real-world process, concept, or system. Note: Models may have other models as components [IEEE 90].

Modifiability
: the degree to which a system or component facilitates the incorporation of changes, once the nature of the desired change has been determined [Boehm 78].

Necessity of characteristics
: the degree to which all of the necessary features and capabilities are present in the software system.

Need satisfaction measures
: a category of quality measures that address how well a system meets the user's needs and requirements.

Network management
: the execution of the set of functions required for controlling, planning, allocating, deploying, coordinating, and monitoring the resources of a computer network [ITS 96].

Network management application
: application that provides the ability to monitor and control the network.

Network management information
: information that is exchanged between the network management station (s) and the management agents that allows the monitoring and control of a managed device.

Network management protocol
: protocol used by the network management station(s) and the management agent to exchange management information.

Network management station
: system that hosts the network management application.

Openness
: the degree to which a system or component complies with standards.

Operability
: the ease of operating the software [Deutsch 88].

Operational testing
: testing conducted to evaluate a system or component in its operational environment [IEEE 90].

Operations and maintenance phase
: the period of time in the software life cycle during which a software product is employed in its operational environment, monitored for satisfactory performance, and modified as necessary to correct problems or to respond to changing requirements [IEEE 90].

Operations personnel
: the number of personnel needed to operate all aspects of a computer system, including the support personnel and facilities needed to support that activity.

Operations system
: the cost of environmentals, communication, licenses, expendables, and documentation maintenance for an operational system.

Organizational measures
: a category of quality measures that address how costly a system is to operate and maintain.

Parallel computing
: a computer system in which interconnected processors perform concurrent or simultaneous execution of two or more processes [McDaniel 94].

Perfective maintenance
: software maintenance performed to improve the performance, maintainability, or other attributes of a computer program [IEEE 90].

Performance measures
: a category of quality measures that address how well a system functions.

Performance testing
: testing conducted to evaluate the compliance of a system or component with specified performance requirements [IEEE 90].

Portability
: the ease with which a system or component can be transferred from one hardware or software environment to another [IEEE 90].

Productivity
: the quality or state of being productive [Webster 87].

Protocol
: a set of conventions that govern the interaction of processes, devices, and other components within a system [IEEE 90].

Provably correct
: the ability to mathematically verify the correctness of a system or component.

Qualification phase
: the period of time in the software life cycle during which it is determined whether a system or component is suitable for operational use.

Qualification testing
: testing conducted to determine whether a system or component is suitable for operational use [IEEE 90].

Quality measure
: a software feature or characteristic used to assess the quality of a system or component.

Readability
: the degree to which a system's functions and those of its component statements can be easily discerned by reading the associated source code.

Real-time responsiveness
: the ability of a system or component to respond to an inquiry or demand within a prescribed time frame.

Recovery
: the restoration of a system, program, database, or other system resource to a prior state following a failure or externally caused disaster; for example, the restoration of a database to a point at which processing can be resumed following a system failure [IEEE 90].

Reengineering
: rebuilding a software system or component to suit some new purpose; for example to work on a different platform, to switch to another language, to make it more maintainable.

Regression testing
: selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements [IEEE 90].

Reliability
: the ability of a system or component to perform its required functions under stated conditions for a specified period of time [IEEE 90].

Requirements engineering
: involves all life-cycle activities devoted to identification of user requirements, analysis of the requirements to derive additional requirements, documentation of the requirements as a specification, and validation of the documented requirements against user needs, as well as processes that support these activities [DoD 91].

Requirements growth
: the rate at which the requirements change for an operational system. The rate can be positive or negative.

Requirements phase
: the period of time in the software life cycle during which the requirements

for a software product are defined and documented [IEEE 90].

Requirements tracing
: describing and following the life of a requirement in both forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases) [Gotel 95].

Resource utilization
: the percentage of time a resource (CPU, Memory, I/O, Peripheral, Network) is busy [Barbacci 95].

Responsiveness
: the degree to which a software system or component has incorporated the user's requirements.

Restart
: to cause a computer program to resume execution after a failure, using status and results recorded at a checkpoint [IEEE 90].

Retirement phase
: the period of time in the software life cycle during which support for a software product is terminated [IEEE 90].

Reusability
: the degree to which a software module or other work product can be used in more than one computing program or software system [IEEE 90].

Reverse engineering
: the process of analyzing a system's code, documentation, and behavior to identify its current components and their dependencies to extract and create system abstractions and design information. The subject system is not altered; however, additional knowledge about the system is produced.

Robustness
: the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions [IEEE 90].

Safety
: a measure of the absence of unsafe software conditions. The absence of catastrophic consequences to the environment [Barbacci 95].

Scalability
: the ease with which a system or component can be modified to fit the problem area.

Security
: the ability of a system to manage, protect, and distribute sensitive information.

Select or develop algorithms
: the activity concerned with selecting or developing a procedural representation of the functions in the software requirements documentation for each software component and data structure. The

algorithms shall completely satisfy the applicable functional and/or mathematical specifications [IEEE 91].

Self-descriptiveness
the degree to which a system or component contains enough information to explain its objectives and properties [IEEE 90].

Simplicity
the degree to which a system or component has a design and implementation that is straightforward and easy to understand [IEEE 90].

Software architecture
the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time [Clements 96].

Software change cycle time
the period of time that starts when a new system requirement is identified and ends when the requirement has been incorporated into the system and delivered for operational use.

Software life cycle
the period of time that begins when a software product is conceived and ends when the software is no longer available for use. The life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and sometimes, retirement phase. These phases may overlap or be performed iteratively, depending on the software development approach used [IEEE 90].

Software maintenance
the cost associated with modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment.

Software migration and evolution
see Adaptive maintenance.

Software upgrade and technology insertion
see Perfective maintenance.

Speed
the rate at which a software system or component performs its functions.

Statistical testing
employing statistical science to evaluate a system or component. Used to demonstrate a system's fitness for use, to predict the reliability of a system in an operational environment, to efficiently allocate testing resources, to predict the amount of testing required after a system change, to qualify components for reuse, and to identify when enough testing has been accomplished [Poore 96].

Structural testing
testing that takes into account the internal mechanism of a system or

component. Types include branch testing, path testing, statement testing. Synonym: white-box testing [IEEE 90].

Structuredness
: the degree to which a system or component possesses a definite pattern of organization of its interdependent parts [Boehm 78].

Sufficiency of characteristics
: the degree to which the features and capabilities of a software system adequately meet the user's needs.

Survivability
: the degree to which essential functions are still available even though some part of the system is down [Deutsch 88].

System allocation
: mapping the required functions to software and hardware. This activity is the bridge between concept exploration and the definition of software requirements [IEEE 91].

System analysis and optimization
: a systematic investigation of a real or planned system to determine the information requirements and processes of the system and how these relate to each other and to any other system, and to make improvements to the system where possible.

System security
: a system function that restricts the use of objects to certain users [McDaniel 94].

System testing
: testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements [IEEE 90].

Taxonomy
: a scheme that partitions a body of knowledge and defines the relationships among the pieces. It is used for classifying and understanding the body of knowledge [IEEE 90].

Test drivers
: software modules used to invoke a module(s) under test and, often, provide test inputs, control and monitor execution, and report test results [IEEE 90].

Test phase
: the period of time in the software life cycle during which the components of a software product are evaluated and integrated, and the software product is evaluated to determine whether or not requirements have been satisfied [IEEE 90].

Test tools
: computer programs used in the testing of a system, a component of the system, or its documentation. Examples include monitor, test case generator, timing analyzer [IEEE 90].

Test
: an activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component [IEEE 90].

Testability
: the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met [IEEE 90]. Note: Not only is testability a measurement for software, it can also apply to the testing scheme.

Testing
: the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component [IEEE 90].

Throughput
: the amount of work that can be performed by a computer system or component in a given period of time [IEEE 90].

Traceability
: the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another [IEEE 90].

Training
: Provisions to learn how to develop, maintain, or use the software system.

Trouble report analysis
: the methodical investigation of a reported operational system deficiency to determine what, if any, corrective action needs to be taken.

Trustworthiness
: the degree to which a system or component avoids compromising, corrupting, or delaying sensitive information.

Understandability
: the degree to which the purpose of the system or component is clear to the evaluator [Boehm 78].

Unit testing
: testing of individual hardware or software units or groups of related units [IEEE 90].

Upgradeability
: see Evolvability.

Usability
: the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component [IEEE 90].

User interface

an interface that enables information to be passed between a human user and hardware or software components of a computer system [IEEE 90].

Verifiability

the relative effort to verify the specified software operation and performance [Evans 87].

Vulnerability

the degree to which a software system or component is open to unauthorized access, change, or disclosure of information and is susceptible to interference or disruption of system services.

References and Information Sources

[IEEE 90] Institute of Electrical and Electronics Engineers. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.* New York, NY: 1990.

References and Information Sources

[Boehm 78] Boehm, Barry W.; Brown, John R.; Kaspar, Hans; Lipow, Myron; MacLeod, Gordon J. & Merritt, Michael J. *Characteristics of Software Quality*. New York, NY: North-Holland Publishing Company, 1978.

References and Information Sources

[Evans 87] Evans, Michael W. & Marciniak, John. *Software Quality Assurance and Management.* New York, NY: John Wiley & Sons, Inc., 1987.

References and Information Sources

[McGill 96] *The Software Agents Mailing List FAQ* [online]. Available WWW <URL: http://www.ee.mcgill.ca/~belmarc/agent_faq.html> (1996).

References and Information Sources

[ITS 96] *Letter-by-Letter Listing* [online]. Available WWW <URL: http://www.its.bldrdoc.gov/fs-1037/dir-001/_0064.htm> (1996).

References and Information Sources

[DoD 91] Department of Defense. *Software Technology Strategy*. DRAFT: December, 1991.

References and Information Sources

[Barbacci 95] Barbacci, Mario; Klein, Mark H.; Longstaff, Thomas H. & Weinstock, Charles B. *Quality Attributes* (CMU/SEI-95-TR-021). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995.

References and Information Sources

[McDaniel 94] McDaniel, George, ed. *IBM Dictionary of Computing.* New York, NY: McGraw-Hill, Inc., 1994.

References and Information Sources

[IEEE 91] IEEE Std 1074-1991. *IEEE Standard for Developing Life Cycle Processes.* New York, NY: Institute of Electrical and Electronics Engineers, 1991.

References and Information Sources

[ARC 96] Laforme, Deborah & Stropky, Maria E. *An Automated Mechanism for Effectively Applying Domain Engineering in Reuse Activities* [online]. Available WWW <URL: http://arc_www.belvoir.army.mil/htmldocs/arc/da_papers/applying_domain_engineering.htm> (1996).

References and Information Sources

[SEI 96] *What is Model Based Software Engineering (MBSE)?* [online]. Available WWW <URL: http://www.sei.cmu.edu/mbse/> (1996).

Glossary Term

Extendability

the ease with which a system or component can be modified to increase its storage or functional capacity [IEEE 90].

References and Information Sources

[Toronto 95] comp.human-factors faq WWW page [online]. Available WWW <URL: http://www.dgp.toronto.edu/people/ematias/faq/G/G-1.html> (1995).

References and Information Sources

[Deutsch 88] Deutsch, Michael S. & Willis, Ronald R. *Software Quality Engineering: A Total Technical and Management Approach.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

References and Information Sources

[Webster 87] *Webster's Ninth New Collegiate Dictionary*. Springfield, MA: Merriam-Webster Inc., 1987.

References and Information Sources

[Gotel 95] Gotel, Orlena. *Contribution Structures for Requirements Traceability*. London, England: Imperial College, Department of Computing, 1995.

Glossary Term

Adaptive maintenance

software maintenance performed to make a computer program usable in a changed environment [IEEE 90].

Glossary Term

Perfective maintenance

software maintenance performed to improve the performance, maintainability, or other attributes of a computer program [IEEE 90].

References and Information Sources

[Poore 96] Poore, Jesse. *Re: Definition for Statistical Testing* [email to Gary Haines], [online]. Available email: ghaines@spacecom.af.mil (October 2, 1996).

Keyword Index

Software Technology Roadmap

The Keyword Index is structured as in any typical document; a few nuances of our index include the following:

- If the keyword is defined in the glossary, it is linked to that glossary definition.
- If the keyword is the name of a technology, it appears in bold type and is linked to that technology description.
- If the keyword is a category in one of the taxonomies, it is followed by that category's index label in parenthesis. AP and QM labels are linked to a list of technology descriptions included in the category.

Each keyword is followed by a list of the technology descriptions that reference it. After selecting one of the descriptions, use your browser's "find" capabilities to locate instances of the keyword.

A| B| C| D| E| F| G| H| I| J| K| L| M| N| O| P| Q| R| S| T| U| V| W| X| Y| Z

A

abstraction
    Object-Oriented Analysis

abstractness (QM.4.4.1.x)

acceptance testing (AP.1.8.2.2)

accessibility (QM.2.1.4.1.3.x), (QM.4.4.1.x)

accountability (QM.2.1.4.2)

accuracy (QM.2.1.2.1)
    Database Two Phase Commit

acquisition cycle time

Active Group
    Component Object Model (COM), DCOM, and Related Capabilities

ActiveX
    Component Object Model (COM), DCOM, and Related Capabilities

Ada 83
    Ada 95

Ada 95

Ada 83
Common Object Request Broker Architecture
Rate Monotonic Analysis

adaptability (QM.3.1.x)

adaptive-maintenance (AP.1.9.3.2)

adaptive-measures (QM.4)

ADL. see architecture description languages

agents (AP.2.8)

Algorithm Formalization

American National Standards Institute
Ada 83

anonymity (QM.2.1.4.1.2.x)

ANSI. see American National Standards Institute

aperiodic task/process
Rate Monotonic Analysis

API. see application program interfaces

application engineering
Domain Analysis and Domain Engineering

application program interfaces (AP.2.7)
Application Program Interface
COTS and Open Systems--An Overview
Java
Message-Oriented Middleware
Middleware
private
Defense Information Infrastructure Common Operating Environment
public
Defense Information Infrastructure Common Operating Environment

application server
Three Tier Software Architectures

applications
event-driven
Message-Oriented Middleware

architectural design (AP.1.3.1)

architecture
Architecture Description Languages
Cleanroom Software Engineering
description languages
Architecture Description Languages
Module Interconnection Languages
modeling
Feature-Oriented Domain Analysis
Reference Models, Architectures, Implementations--An Overview

Argument-Based Design Rationale Capture Methods for Requirements

Tracing
artificial intelligence
asynchronous
processing
Distributed Computing Environment
auditable (QM.2.1.4.2.1)
Authenticode
Component Object Model (COM), DCOM, and Related Capabilities
automatic programming
availability (QM.2.1.1)
Intrusion Detection
Software Inspections
Statistical-Based Intrusion Detection

B

backfiring
Function Point Analysis
Bang measure
Function Point Analysis
binary large objects
Object-Oriented Database
black-box testing (AP.1.4.3.4.x)
BLOBS. see binary large objects
Bowles metrics
Cyclomatic Complexity
box structure method
Cleanroom Software Engineering
browsers
Computer System Security--An Overview

C

C
Ada 83
Common Object Request Broker Architecture
Distributed Computing Environment
C++
Ada 83
Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures
C4I
Computer System Security--An Overview
Capability Maturity Model

Cleanroom Software Engineering
Personal Software Process for Module-Level Development
Software Inspections

capacity (QM.2.2.1)

cell
in distributed computing
Distributed Computing Environment

Cleanroom Software Engineering
Object-Oriented Analysis
Object-Oriented Design

client
Two Tier Software Architectures

client/server (AP.2.1.2.1)
communication (AP.2.2.1)
Distributed Computing Environment
Mainframe Server Software Architectures
Message-Oriented Middleware
Object Request Broker
Remote Procedure Call
Software Architectures

CMIP. see Common Management Information Protocol

CMM. see Capability Maturity Model

Coad-Yourdan
Object-Oriented Analysis

COCOMO. see constructive cost model

code (AP.1.4.2)
analyzers (AP.1.4.3.4.x)
complexity
Halstead Complexity Measures
entropy
Maintenance of Operational Systems--An Overview
generator
Graphical User Interface Builders

COE. see Common Operating Environment

commercial-off-the-shelf
Application Programming Interface
Component-Based Software Development/COTS Integration
COTS and Open Systems--An Overview
integration
Application Programming Interface

commit phase
Database Two Phase Commit

Common Management Information Protocol
Simple Network Management Protocol

Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures

Distributed Computing Environment
Middleware
Object Request Broker
compliance
Object Request Broker
implementations
Object Request Broker

Common Operating Environment
TAFIM Reference Model
architecture
Defense Information Infrastructure Common Operating Environment
compliance levels
Defense Information Infrastructure Common Operating Environment
component segments
Defense Information Infrastructure Common Operating Environment
Information Server
Defense Information Infrastructure Common Operating Environment
Software Repository System
Defense Information Infrastructure Common Operating Environment

commonality (QM.4.1.2.x)
Domain Engineering and Domain Analysis
communication software (AP.2.2)
compactness (QM.2.2.x)
compartmented mode workstations
Computer System Security--An Overview
Trusted Operating Systems
compatibility (QM.4.1.1)
Graphic Tools for Legacy Database Migration
compiler (AP.1.4.2.3)
Ada 83
Ada 95
completeness (QM.1.3.1)
Requirements Tracing
complexity (QM.3.2.1)
Halstead Complexity Measures
Message-Oriented Middleware
Remote Procedure Call
analysis
Maintenance of Operational Systems--An Overview
apparent (QM.3.2.1.x)
inherent (QM.3.2.1.x)
compliance (standalone)

Common Object Request Broker Architecture
Defense Information Infrastructure Common Operating Environment

component
adaptation
Component-Based Software Development/COTS Integration
assembly
Component-Based Software Development/COTS Integration
selection and evaluation
Component-Based Software Development/COTS Integration
testing (AP.1.4.3.5)

Component Object Model
Distributed Computing Environment
Middleware
Object Request Broker

Component-Based Software Development/COTS Intergration

component-based software engineering
Component-Based Software Development/COTS Intergration

computational complexity
Halstead Complexity Measures

Computer System Security--An Overview

concept phase (AP.1.1)

conciseness (QM.3.2.4.x)

concurrent engineering
Cleanroom Software Engineering

confidentiality (QM.2.1.4.1.2)
Intrusion Detection

conformance
COTS and Open Systems--An Overview

connected graph
Cyclomatic Complexity

connectivity software
Middleware

consistency (QM.1.3.2)
Algorithm Formalization
Requirements Tracing

constructive cost model
Function Point Analysis

context analysis
Feature-Oriented Domain Analysis

CORBA. see Common Object Request Broker Architecture

corrective maintenance (AP.1.9.3.1)

correctness (QM.1.3)
Cleanroom Software Engineering

cost estimation (AP.1.3.7)
Function Point Analysis

cost of maintenance (QM.5.1.2)

cost of operation (QM.5.1.1)
cost of ownership (QM.5.1)
COTS and Open Systems--An Overview
COTS. see commercial-off-the-shelf
cycle time
Cleanroom Software Engineering
Cyclomatic Complexity

D

data
analyzers (AP.1.4.3.4.x)
complexity
Cyclomatic Complexity
exchange
Object Request Broker
integrity
Database Two Phase Commit
Public Key Digital Signatures
management (AP.2.6.1)
management security (AP.2.4.2)
mining
Mainframe Server Software Architectures
recording (AP.2.9)
reduction (AP.2.9)
sharing
Application Programming Interface
visualization
Graphic Tools for Legacy Database Migration
warehouses
Mainframe Server Software Architectures
Data Encryption Standard
Simple Network Management Protocol
databases (AP.2.6)
Graphic Tools for Legacy Database Migration
administration (AP.1.9.1)
design (AP.1.3.2)
management
Three Tier Software Architecture
Two Tier Software Architectures
management system
Multi-Level Secure Database Management Schemes
server
Transaction Processing Monitor Technology
two phase commit
Database Two Phase Commit

utilities (AP.1.4.2.2)

DBMS. see database management system

debugger (AP.1.4.2.4.x)

decision support systems

Two Tier Software Architectures

defect

detection

Software Inspections

leakage

Software Inspections

management

Personal Software Process for Module-Level Development

prevention

Cleanroom Software Engineering

Defense Information Infrastructure Common Operating Environment. see Common Operating Environment

Defense Information Systems Agency

COTS and Open Systems--An Overview

TAFIM Reference Model

denial of service (QM.2.1.4.1.3)

Department of Defense systems

evolution of

TAFIM Reference Model

dependability (QM.2.1)

Rate Monotonic Analysis

DES. see Data Encryption Standard

design

Cleanroom Software Engineering

architectural (AP.1.3.1)

complexity

Cyclomatic Complexity

database (AP.1.3.2)

decision

history

Feature-Based Design Rationale Capture Method for Requirements Tracing

decisions

Feature-Based Design Rationale Capture Method for Requirements Tracing

detailed (AP.1.3.5)

interface (AP.1.3.3)

phase (AP.1.3)

rationale

Argument-Based Design Rationale Capture Methods for Requirements Tracing

Feature-Based Design Rationale Capture Method for Requirements Tracing

Requirements Tracing
capture
Argument-Based Design Rationale Capture Methods for Requirements Tracing
history
Argument-Based Design Rationale Capture Methods for Requirements Tracing
detailed design (AP.1.3.5)
development phase
Cleanroom Software Engineering
Maintenance of Operational Systems--An Overview
digital signatures
Computer System Security--An Overview
Public Key Digital Signatures
DII COE. see Defense Information Infrastructure Common Operating Environment
directory services
Distributed Computing Environment
DISA. see Defense Information Systems Agency
diskless support
Distributed Computing Environment
distributed
business models
Distributed/Collaborative Enterprise Architectures
client/server architecture
Three Tier Software Architecture
computing (AP.2.1.2)
database system
Database Two Phase Commit
environment
TAFIM Reference Model
system
Distributed Computing Environment
Object Request Broker
Remote Procedure Call
services
Middleware
Distributed/Collaborative Enterprise Architectures
Client/Server Software Architectures
Distributed Computing Environment
Common Object Request Broker Architecture
Middleware
domain
Domain Engineering and Domain Analysis
analysis
Cleanroom Software Engineering
Domain Engineering and Domain Analysis

Feature-Oriented Domain Analysis
Organization Domain Modeling
design
engineering (AP.1.2.4)
Domain Engineering and Domain Analysis
Organization Domain Modeling
implementation
modeling
Feature-Oriented Domain Analysis
Organization Domain Modeling
Domain Engineering and Domain Analysis
dynamic binding
Object-Oriented Programming Languages

E

early operational phase
Maintenance of Operational Systems--An Overview
effectiveness (QM.1.1)
efficiency (QM.2.2)
Algorithm Formalization
Transaction Processing Monitor Technology
electronic encryption key distribution cryptography
Computer System Security--An Overview
end-to-end encryption
Computer System Security--An Overview
engineering function points
Function Point Analysis
Halstead Complexity Measures
entropy
Maintenance of Operational Systems--An Overview
error
handling (AP.2.11)
proneness (QM.2.3.1)
tolerance (QM.2.1.1.x)
essential complexity
Cyclomatic Complexity
estimating
Personal Software Process for Module-Level Development
event-driven applications
Message-Oriented Middleware
evolution/replacement phase
Maintenance of Operational Systems--An Overview
evolvability (QM.3.1.x)
expandability (QM.3.1.x)

extendability (QM.3.1.x)

F

fail safe (QM.2.1.1.x)
fail soft (QM.2.1.1.x)
FARS. see Federal Acquisition Regulations
FASA. see Federal Acquisition Streamlining Acts
fault
fault tolerance (QM.2.1.1.x)
feature analysis
    Feature-Oriented Domain Analysis
feature points
    Function Point Analysis
Feature-Based Design Rationale Capture Method for Requirements Tracing
Feature-Oriented Domain Analysis
Federal Acquisition Regulations
    COTS and Open Systems--An Overview
Federal Acquisition Streamlining Acts
    COTS and Open Systems--An Overview
fidelity (QM.2.4)
file systems
    Distributed Computing Environment
    support for
        Remote Procedure Call
file transfer
    Application Programming Interface
firewalls
    Computer System Security--An Overview
    Firewalls and Proxies
    proxies, and
        Firewalls and Proxies
fixed priority
    Rate Monotonic Analysis
flexibility (QM.3.1.x)
    Ada 83
    Ada 95
    Client/Server Software Architectures
    Component-Based Software Development/COTS Integration
    Distributed/Collaborative Enterprise Architectures
    Distributed Computing Environment
    Mainframe Server Software Architectures
    Message-Oriented Middleware
    Remote Procedure Call
    TAFIM Reference Model

Three Tier Software Architecture
Transaction Processing Monitor Technology
Two Tier Software Architectures

FODA. see Feature-Oriented Domain Analysis

FORTEZZA
Computer System Security--An Overview

function call
Remote Procedure Call

Function Point Analysis

function points
early and easy
Function Point Analysis

functional scope (QM.4.4.1)

functional size measurement
Function Point Analysis

functional testing (AP.1.4.3.4.x)

functionality analysis
Maintenance of Operational Systems--An Overview

fundamental distributed services
Distributed Computing Environment

Futurebus+
Rate Monotonic Analysis

G

GCCS. see Global Command and Control System

GCSS. see Global Combat Support System

generality (QM.4.4.1.x)

Global Combat Support System
Defense Information Infrastructure Common Operating Environment
TAFIM Reference Model

Global Command and Control System
Defense Information Infrastructure Common Operating Environent
TAFIM Reference Model

Graphic Tools for Legacy Database Migration

graphical user interface
Graphical User Interface Builders
builders
Graphical User Interface Builders

graphics (AP.2.3.2)

GUI builders. see graphical user interface builders

H

Halstead complexity measures
- Cyclomatic Complexity
- Function Point Analysis

hardware maintenance

hardware-software co-design (AP.1.3.1.x)

Henry metrics
- Cyclomatic Complexity

heterogeneous databases
- Three Tier Software Architecture

homogeneous environment
- Two Tier Software Architectures

human computer interaction (AP.2.3)

human engineering

I

IDTs. see interface development tools

IFPUG. see International Function Point User Group

implementation phase (AP.1.4)

implementations
- overview of
  - **Reference Models, Architectures, Implementations--An Overview**

incompleteness (QM.1.3.1)

incremental development
- Cleanroom Software Engineering

independence
- Distributed Computing Environment

information
- analysis
  - Feature-Oriented Domain Analysis
- hiding
  - Object-Oriented Programming Languages
- security (AP.2.4)
- warfare
  - Intrusion Detection
  - Rule-Based Intrusion Detection
  - Statistical-Based Intrusion Detection

inheritance
- Object-Oriented Programming Languages

installation and checkout phase (AP.1.8)

integration testing (AP.1.5.3.2)

integrity (QM.2.1.4.1.1)
- Distributed Computing Environment

Intrusion Detection

interface

definition language

Common Object Request Broker Architecture

Distributed Computing Environment

design (AP.1.3.3)

development tools

Graphical User Interface Builders

specification

COTS and Open Systems--An Overview

standards

COTS and Open Systems--An Overview

testing (AP.1.5.3.3)

International Function Point User Group

Function Point Analysis

International Standards Organization

Ada 83

Function Point Analysis

standards

Distributed Computing Environment

internet

Firewalls and Proxies

Object Request Broker

standards

Distributed Computing Environment

interoperability (QM.4.1)

Ada 83

Ada 95

Application Programming Interface

Client/Server Software Architectures

Defense Information Infrastructure Common Operating Environment

Distributed Computing Environment

Message-Oriented Middleware

Middleware

Remote Procedure Call

TAFIM Reference Model

intranet

Computer System Security--An Overview

Firewalls and Proxies

Object Request Broker

Intrusion Detection

Computer System Security--An Overview

model-based

Rule-Based Intrusion Detection

rule-based

Rule-Based Intrusion Detection

statistical-based

Statistical-Based Intrusion Detection

ISO see International Standards Organization

J

Jacobson
Object-Oriented Analysis
Java
Ada 95
Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures
Object Request Broker
Joint Technical Architecture
Defense Information Infrastructure Common Operating Environment
JTA. see Joint Technical Architecture

K

Kafura metrics
Cyclomatic Complexity
kernel COE
Defense Information Infrastructure Common Operating Environment

L

latency (QM.2.2.2)
legacy systems
COTS and Open Systems--An Overview
Distributed Computing Environment
Domain Engineering and Domain Analysis
lifetime of operational capability
Ligier metrics
Cyclomatic Complexity
lines of code
Function Point Analysis
metrics
Halstead Complexity Measures
LOC. see lines of code

M

MAC. see message authentication code

Mainframe Server Software Architectures

maintainability (QM.3.1)
- Ada 83
- Ada 95
- Cyclomatic Complexity
- Domain Engineering and Domain Analysis
- Feature-Oriented Domain Analysis
- Graphic Tools for Legacy Database Migration
- Maintainability Index Technique for Measuring Program Maintainability
- Module Interconnection Languages
- Object-Oriented Analysis
- Object-Oriented Database
- Object-Oriented Design
- Object-Oriented Programming Languages
- Organization Domain Modeling
- Software Inspections
- Three Tier Software Architecture

Maintainability Index Technique for Measuring Program Maintainability

maintenance
- adaptive (AP.1.9.3.2)
- control
- corrective (AP.1.9.3.1)
- costs
  - Maintenance of Operational Systems--An Overview
- documentation
  - Maintenance of Operational Systems--An Overview
- measures
- metric
  - Halstead Complexity Measures
- perfective (AP.1.9.3.3)
- personnel

Maintenance of Operational Systems--An Overview

managed device

managed object

managed objects
- Simple Network Management Protocol

management agent

management information base
- Simple Network Management Protocol

manufacturing phase (AP.1.7)

marshalling
- Component Object Model (COM), DCOM, and Related Capabilities

mature operational phase
- Maintenance of Operational Systems--An Overview

mature systems
- Maintenance of Operational Systems--An Overview

McCabe's complexity
Cyclomatic Complexity
message authentication code
Public Key Digital Signatures
message delivery
Application Programming Interface
message digest function
Public Key Digital Signatures
Message-Oriented Middleware
Middleware
Remote Procedure Call
metrics
Cyclomatic Complexity
Halstead
Cyclomatic Complexity
Function Point Analysis
Henry
Cyclomatic Complexity
McCabe
Cyclomatic Complexity
Troy
Cyclomatic Complexity
Zweben
Cyclomatic Complexity
MIB. see management information base
middle tier server
Three Tier Software Architecture
Middleware
Application Programming Interface
Message-Oriented Middleware
Object Request Broker
Transaction Processing Monitor Technology
minimal operator intervention
Two Tier Software Architecture
MISSI. see Multilevel Information Systems Security Initiative
MLS Host
Computer System Security--An Overview
MLS Operating System
Computer System Security--An Overview
MLS. see multi-level security
models (AP.2.1.1)
modifiability (QM.3.1.x)
Application Programming Interface
Cleanroom Software Engineering
Module Interconnection Languages
module-level development
Personal Software Process for Module-Level Development

MOM. see message-oriented middleware
moniker
Component Object Model (COM), DCOM, and Related Capabilities
Morris Worm
Virus Detection
Motif
Graphical User Interface Builders
Multilevel Information Systems Security Initiative
Computer System Security--An Overview
multi-level secure
database management schemes
Computer System Security--An Overview
Multi-Level Secure Database Management Schemes
guard
Computer System Security--An Overview
one way guard with random acknowledgement
Multi-Level Secure One Way Guard with Random Acknowledgment
systems
Computer System Security--An Overview
multi-level security
Multi-Level Secure Database Management Schemes
Trusted Operating System
multiplexing client transaction requests
Transaction Processing Monitor Technology

N

NDI. see non-developmental items
necessity of characteristics (QM.1.1.1)
need satisfaction measures
network
Application Programming Interface
Distributed Computing Environment
architecture
Middleware
hardware
Distributed Computing Environment
management (AP.2.2.2)
manager
Message-Oriented Middleware
overhead
Distributed Computing Environment
performance of
Remote Procedure Call
protocols

interface to
Remote Procedure Call
security
Firewalls and Proxies
network management
network management application
network management information
network management protocol
network management station
Network Management--An Overview
non-developmental items
COTS and Open Systems--An Overview
Nonrepudiation in Network Communications

O

object activation
Object Request Broker
Object Linking and Embedding
Component Object Model (COM), DCOM, and Related Capabilities
Object Management Architecture
Common Object Request Broker Architecture
Object Management Group
Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures
object model
Object-Oriented Analysis
Object-Oriented Database
object orientation
Object Request Broker
object-oriented
Cleanroom Software Engineering
Distributed Computing Environment
Remote Procedure Call
analysis
Object-Oriented Analysis
database
Object-Oriented Database
design
Object-Oriented Design
programming
Ada 83
Ada 95
programming language
Object-Oriented Programming Languages
systems

Message-Oriented Middleware

Object Request Broker

Client/Server Software Architectures

Common Object Request Broker Architecture

Distributed/Collaborative Enterprise Architectures

Middleware

objects

Object-Oriented Analysis

Object Request Broker

ODM. see organization domain modeling

one way guards

Computer System Security--An Overview

OOA. see object-oriented analysis

OOD. see object-oriented design

OODB. see object-oriented database

OOPL. see object-oriented programming languages

Open Group

Component Object Model (COM), DCOM, and Related Capabilities

open systems

Application Programming Interface

COTS and Open Systems--An Overview

Mainframe Server Software Architectures

cost

COTS and Open Systems--An Overview

COTS, and

COTS and Open Systems--An Overview

interconnect standards

Distributed Computing Environment

openness (QM.4.1.2)

operability (QM.2.3.2)

operational analysis

Feature-Oriented Domain Analysis

operational testing (AP.1.8.2.1)

operations

personnel

system

operations and maintenance phase (AP.1.9)

opportunistic reuse

Domain Engineering and Domain Analysis

ORB. see object request broker

Organization Domain Modeling

organizational measures

overview of reference models, architectures, implementations

Reference Models, Architectures, Implementations--An Overview

P

parallel computing (AP.2.1.3)
payload
Virus Detection
peer reviews
Software Inspections
perfective maintenance (AP.1.9.3.3)
performance
Graphic Tools for Legacy Database Migration
Rate Monotonic Analysis
Three Tier Software Architecture
measures
testing (AP.1.5.3.5)
periodic task/process
Rate Monotonic Analysis
persistent
data
Object-Oriented Database
objects
Object-Oriented Database
Personal Software Process
Personal Software Process for Module-Level Development
for module-level development
Personal Software Process for Module-Level Development
piecewise reengineering
Maintenance of Operational Systems--An Overview
pilot project
Cleanroom Software Engineering
plug-and-play
COTS and Open Systems--An Overview
polymorphism
Object-Oriented Programming Languages
portability (QM.4.2)
Ada 83
Ada 95
Defense Information Infrastructure Common Operating Environment
Distributed Computing Environment
Graphic Tools for Legacy Database Migration
POSIX
Rate Monotonic Analysis
pre-delivery phase
Maintenance of Operational Systems--An Overview
prepare phase

Database Two Phase Commit
priority inheritance
Rate Monotonic Analysis
priority inversion
Rate Monotonic Analysis
process management services
Three Tier Software Architecture
processing management
Two Tier Software Architectures
product line
Component-Based Software Development/COTS Integration
productivity (QM.5.2)
Function Point Analysis
Object-Oriented Analysis
rates
Function Point Analysis
profiles
Statistical-Based Intrusion Detection
programming language (AP.1.4.2.1)
proprietary interfaces
COTS and Open Systems--An Overview
protocols (AP.2.2.3)
COTS and Open Systems--An Overview
support of
Message-Oriented Middleware
provably correct (QM.1.3.4)
proxies
Computer System Security--An Overview
Firewalls and Proxies
PSP. see Personal Software Process
public key cryptography
Computer System Security--An Overview
Public Key Digital Signatures
Public Key Digital Signatures

Q

qualification phase (AP.1.6)
qualification testing (AP.1.6.1)
quality
Cleanroom Software Engineering
quality measures

Component-Based Software Development/COTS Integration
Personal Software Process for Module-Level Development

queuing theory
Rate Monotonic Analysis

R

Rate Monotonic Analysis

rate monotonic scheduling
Rate Monotonic Analysis

rationale capture
Argument-Based Design Rationale Capture Methods for Requirements Tracing
Feature-Based Design Rationale Capture Method for Requirements Tracing

RBID. see Rule-Based Intrusion Detection

RDA. see remote data access

readability (QM.3.2.4)

real-time
Rate Monotonic Analysis
responsiveness
responsiveness (QM.2.2.2)
systems
COTS and Open Systems--An Overview
Rate Monotonic Analysis

recovery (AP.2.10)

reengineering (AP.1.9.5)
Cyclomatic Complexity
Graphic Tools for Legacy Database Migration
Graphical User Interface Builders
Maintenance of Operational Systems--An Overview

reference models
overview of
Reference Models, Architectures, Implementations--An Overview

regression testing (AP.1.5.3.4)

reliability (QM.2.1.2)
Ada 83
Ada 95
Cleanroom Software Engineering
Distributed/Collaborative Enterprise Architectures
Software Inspections

Transaction Processing Monitor Technology

remote data access
Transaction Processing Monitor Technology

remote method invocation
Object Request Broker

Remote Procedure Call
Application Programming Interface
Distributed Computing Environment
Message-Oriented Middleware
Middleware
Transaction Processing Monitor Technology

requirements
cross referencing
Requirements Tracing
engineering (AP.1.2.2)
growth (QM.5.1.2.6)
phase (AP.1.2)
tracing (AP.1.2.3)
Maintenance of Operational Systems--An Overview
Requirements Tracing

requirements-to-code (AP.1.2.3.1)

resource utilization (QM.2.2)

responsiveness (QM.1.2)

restart (AP.2.10)

restructuring
Maintenance of Operational Systems--An Overview

retirement phase (AP.1.10)

retrievability (QM.4.4.2)

reusability (QM.4.4)
Ada 83
Ada 95
Architecture Description Languages
Defense Information Infrastructure Common Operating Environment
Domain Engineering and Domain Analysis
Feature-Based Design Rationale Capture Method for Requirements Tracing
Feature-Oriented Domain Analysis
Mainframe Server Software Architectures
Object-Oriented Analysis
Object-Oriented Design
Organization Domain Modeling
Three Tier Software Architecture
Transaction Processing Monitor Technology

reuse
Component-Based Software Development/COTS Integration
Module Interconnection Languages
reverse-engineering (AP.1.9.4)
Maintenance of Operational Systems--An Overview
design recovery
Maintenance of Operational Systems--An Overview
REVIC. see revised intermediate COCOMO
revised intermediate COCOMO
Function Point Analysis
risk analysis
Cyclomatic Complexity
RMA. see rate monotonic analysis
RMI. see remote method invocation
robustness (QM.2.1.1)
RPC. see remote procedure call
Rule-Based Intrusion Detection
Rumbaugh
Object-Oriented Analysis
runtime environment
Defense Information Infrastructure Common Operating Environment

S

safety (QM.2.1.3)
scalability (QM.4.3)
Distributed/Collaborative Enterprise Architectures
Distributed Computing Environment
Mainframe Server Software Architectures
Three Tier Software Architecture
Two Tier Software Architectures
schedulability analysis
Rate Monotonic Analysis
scheduling
Rate Monotonic Analysis
security (QM.2.1.5)
Distributed Computing Environment
Firewalls and Proxies
Intrusion Detection
Multi-Level Secure Database Management Schemes
Public Key Digital Signatures
Trusted Operating Systems

security services
Distributed Computing Environment
segments
Defense Information Infrastructure Common Operating Environment
select or develop algorithms
self descriptiveness (QM.3.2.4.x)
server
Two Tier Software Architecture
session based technology
Transaction Processing Monitor Technology
sharing services
Distributed Computing Environment
Shlaer-Mellor
Object-Oriented Analysis
Simple Network Management Protocol
secure SNMP
Simple Network Management Protocol
SNMPv1
Simple Network Management Protocol
SNMPv2
Simple Network Management Protocol
Simplex Architecture
simplicity (QM.3.2.2)
Smalltalk
Common Object Request Broker Architecture
Distributed/Collaborative Enterprise Architectures
SNMP. see Simple Network Management Protocol
software
architecture (AP.2.1)
change cycle time
complexity
Maintenance of Operational Systems--An Overview
engineering
Personal Software Process for Module-Level Development
engineering tools
Graphical User Interface Builders
entropy
Maintenance of Operational Systems--An Overview
generation
Algorithm Formalization
inspections
Software Inspections
life cycle
Cleanroom Software Engineering

maintainability
Maintenance of Operational Systems--An Overview
maintenance (QM.5.1.2.5)
metrics
Cyclomatic Complexity
migration and evolution (AP.1.9.3.2)
process improvement
Personal Software Process for Module-Level Development
productivity
Function Point Analysis
synthesis
Algorithm Formalization
upgrade and technology insertion (AP.1.9.3.3)
Software Technology for Adaptable Reliable Systems
Cleanroom Software Engineering
Organization Domain Modeling
speed (QM.2.2.x)
SQL. see standard query language
standard query language
Application Programming Interface
STARS. see Software Technology for Adaptable Reliable Systems
static metrics
Cyclomatic Complexity
statistical quality control
Cleanroom Software Engineering
statistical testing (AP.1.5.3.5.x)
Cleanroom Software Engineering
Statistical-Based Intrusion Detection
structural complexity
Cyclomatic Complexity
structural testing (AP.1.4.3.4.x)
structuredness (QM.3.2.3)
sufficiency of characteristics (QM.1.1.2)
support requirements
Function Point Analysis
survivability (QM.2.1.4.1.4)
synchronous mechanism
Message-Oriented Middleware
Remote Procedure Call
synchronous processing
Distributed Computing Environment
system
administrators

Distributed Computing Environment
allocation (AP.1.2.1)
analysis and optimization (AP.1.3.6)
Cleanroom Software Engineering
change costs
Function Point Analysis
evolution
Component-Based Software Development/COTS Integration
integration
Component-Based Software Development/COTS Integration
lifecycle
Maintenance of Operational Systems--An Overview
migration
Graphic Tools for Legacy Database Migration
security (AP.2.4.3)
testing (AP.1.5.3.1)
system engineering
Domain Engineering and Domain Analysis
systematic reuse
Domain Engineering and Domain Analysis
Organization Domain Modeling

T

TAFIM
Defense Information Infrastructure Common Operating Environment
Application Program Interface
TAFIM Reference Model
External Environment Interface
TAFIM Reference Model
reference model
TAFIM Reference Model
tasks
Rate Monotonic Analysis
taxonomy
test (AP.1.4.3)
drivers (AP.1.4.3.2, AP.1.5.1)
generation
Maintenance of Operational Systems--An Overview
optimization
Maintenance of Operational Systems--An Overview
phase (AP.1.5)
planning
Cyclomatic Complexity

tools (AP.1.4.3.3, AP.1.5.2)

testability (QM.1.4.1)

testing (AP.1.5.3)

acceptance (AP.1.8.2.2)

black-box (AP.1.4.3.4.x)

component (AP.1.4.3.5)

functional (AP.1.4.3.4.x)

integration (AP.1.5.3.2)

interface (AP.1.5.3.3)

operational (AP.1.8.2.1)

performance (AP.1.5.3.5)

qualification (AP.1.6.1)

regression (AP.1.5.3.4)

statistical (AP.1.5.3.5.x)

Cleanroom Software Engineering

structural (AP.1.4.3.4.x)

system (AP.1.5.3.1)

unit (AP.1.4.3.4)

white-box (AP.1.4.3.4.x)

threads

Distributed Computing Environment

Rate Monotonic Analysis

services

Distributed Computing Environment

three tier

architecture

Mainframe Server Software Architectures

Three Tier Software Architecture

software architectures

Three Tier Software Architecture

client/server

Message-Oriented Middleware

with application server

Client/Server Software Architectures

with message server

Client/Server Software Architectures

with ORB architecture

Client/Server Software Architectures

throughput (QM.2.2.3)

Intrusion Detection

time services

Distributed Computing Environment

TP Heavy

Client/Server Software Architectures

TP Lite
Client/Server Software Architectures
TP monitor. see transaction processing monitor technology.
traceability (QM.1.3.3)
Requirements Tracing
training (QM.5.1.1.2), QM.5.1.2.2)
transaction applications
Transaction Processing Monitor Technology
Transaction Processing Monitor Technology
Client/Server Software Architectures
Middleware
translation
Maintainability Index Technique for Measuring Program Maintainability
Maintenance of Operational Systems--An Overview
restructuring/modularizing
Maintenance of Operational Systems--An Overview
transport software
Distributed Computing Environment
trouble report analysis (AP.1.9.2)
Troy metrics
Cyclomatic Complexity
Trusted Operating Systems (AP.2.4.1)
Multi-Level Secure Database Management Schemes
trustworthiness (QM.2.1.4)
Public Key Digital Signatures
two life cycle model
Domain Engineering and Domain Analysis
two phase commit technology
Database Two Phase Commit
two tier
architecture
Mainframe Server Software Architectures
software architectures
Two Tier Software Architectures

U

UDP. see user datagram protocol
UIL. see user interface language
UIMS. see user interface management system
understandability (QM.3.2)
Architecture Description Languages

Cleanroom Software Engineering
Domain Engineering and Domain Analysis
Graphic Tools for Legacy Database Migration
Module Interconnection Languages
Organization Domain Modeling

unit testing (AP.1.4.3.4)

UNIX
Mainframe Server Software Architectures

unmarshalling
Component Object Model (COM), DCOM, and Related Capabilities

upgradeability (QM.3.1.x)

usability (QM.2.3)
Client/Server Software Architectures
Domain Engineering and Domain Analysis
Graphical User Interface Builders
Two Tier Software Architectures

user datagram protocol
Simple Network Management Protocol

user interfaces (AP.2.3.1)
development tools
Graphical User Interface Builders
language
Graphical User Interface Builders
management system
Graphical User Interface Builders

user services
Three Tier Software Architecture

user system interface
Two Tier Software Architectures

user friendly interface
Two Tier Software Architectures

V

validation suite
Ada
Ada 83
Ada 95

variability
Domain Engineering and Domain Analysis

vendor-driven upgrades
Component-Based Software Development/COTS Integration

verifiability (QM.1.4)

Cleanroom Software Engineering

VHDL. see VHSIC Hardware Description Language

VHSIC Hardware Description Language

Architecture Description Languages

virus

Virus Detection

Virus Detection

Computer System Security--An Overview

visualization tool

Graphic Tools for Legacy Database Migration

vulnerability (QM.2.1.4.1)

W

walkthroughs

Software Inspections

white-box testing (AP.1.4.3.4.x)

widgets

Graphical User Interface Builders

workstation compliance level three

Defense Information Infrastructure Common Operating Environment

World Wide Web

Firewalls and Proxies

X

Y

Z

Zweben metrics

Cyclomatic Complexity

URL: http://www.sei.cmu.edu/str/indexes/keywords/index_body.html
Last Modified: 24 July 2008

Glossary Term

Abstractness

the degree to which a system or component performs only the necessary functions relevant to a particular purpose.

Glossary Term

Acceptance testing

formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system [IEEE 90].

Glossary Term

Accessibility

1. (Denial of Service) the degree to which the software system protects system functions or service from being denied to the user
2. (Reusability) the degree to which a software system or component facilitates the selective use of its components [Boehm 78].

Glossary Term

Accountability

the ability of a system to keep track of who or what accessed and/or made changes to the system.

Glossary Term

Acquisition cycle time

the period of time that starts when a system is conceived and ends when the product meets its initial operational capability.

Glossary Term

Adaptive measures

a category of quality measures that address how easily a system can evolve or migrate.

Glossary Term

Agent

a piece of software which acts to accomplish tasks on behalf of its user [McGill 96].

Glossary Term

ANSI

American National Standards Institute. This organization is responsible for approving U.S. standards in many areas, including computers and communications. Standards approved by this organization are often called ANSI standards (e.g., ANSI C is the version of the C language approved by ANSI). ANSI is a member of ISO. *See also:* International Organization for Standardization.

Glossary Term

Anonymity

the degree to which a software system or component allows for or supports anonymous transactions.

Glossary Term

Application program interface

a formalized set of software calls and routines that can be referenced by an application program in order to access supporting system or network services [ITS 96].

Glossary Term

Architectural design

the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system [IEEE 90].

Glossary Term

Artificial intelligence

a subfield within computer science concerned with developing technology to enable computers to solve problems (or assist humans in solving problems) using explicit representations of knowledge and reasoning methods employing that knowledge [DoD 91].

Glossary Term

Auditable

the degree to which a software system records information concerning transactions performed against the system.

Glossary Term

Capacity

a measure of the amount of work a system can perform [Barbacci 95].

Glossary Term

Code

the transforming of logic and data from design specifications (design descriptions) into a programming language [IEEE 90].

Glossary Term

Commonality

the degree to which standards are used to achieve interoperability.

Glossary Term

Communication software

software concerned with the representation, transfer, interpretation, and processing of data among computer systems or networks. The meaning assigned to the data must be preserved during these operations.

Glossary Term

Compactness

the degree to which a system or component makes efficient use of its data storage space-occupies a small volume.

Glossary Term

Component testing

testing of individual hardware or software components or groups of related components [IEEE 90].

Glossary Term

Concept phase

the initial phase of a software development project, in which the user needs are described and evaluated through documentation (for example, statement of needs, advance planning report, project initiation memo, feasibility studies, system definition, documentation, regulations, procedures, or policies relevant to the project) [IEEE 90].

Glossary Term

Conciseness

the degree to which a software system or component has no excessive information present.

Glossary Term

Corrective maintenance

maintenance performed to correct faults in hardware or software [IEEE 90].

Glossary Term

Cost of maintenance

the overall cost of maintaining a computer system to include the costs associated with personnel, training, maintenance control, hardware and software maintenance, and requirements growth.

Glossary Term

Cost of operation

the overall cost of operating a computer system to include the costs associated with personnel, training, and system operations.

Glossary Term

Data management

the function that provides access to data, performs or monitors the storage of data, and controls input/output operations [McDaniel 94].

Glossary Term

Data management security

the protection of data from unauthorized (accidental or intentional) modification, destruction, or disclosure [ITS 96].

Glossary Term

Data recording

to register all or selected activities of a computer system. Can include both external and internal activity.

Glossary Term

Data reduction

any technique used to transform data from raw data into a more useful form of data. For example, grouping, summing, or averaging related data [IEEE 90].

Glossary Term

Database

1. a collection of logically related data stored together in one or more computerized files. Note: Each data item is identified by one or more keys [IEEE 90].
2. an electronic repository of information accessible via a query language interface [DoD 91].

Glossary Term

Database administration

the responsibility for the definition, operation, protection, performance, and recovery of a database [IEEE 90].

Glossary Term

Database design

the process of developing a database that will meet a user's requirements. The activity includes three separate but dependent steps: conceptual database design, logical database design, and physical database design [IEEE 91].

Glossary Term

Denial of service

the degree to which a software system or component prevents the interference or disruption of system services to the user.

Glossary Term

Design phase

the period of time in the software life cycle during which the designs for architecture, software components, interfaces, and data are created, documented, and verified to satisfy requirements [IEEE 90].

Glossary Term

Detailed design

the process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented [IEEE 90].

Glossary Term

Distributed computing

a computer system in which several interconnected computers share the computing tasks assigned to the system [IEEE 90].

Glossary Term

Domain analysis

the activity that determines the common requirements within a domain for the purpose of identifying reuse opportunities among the systems in the domain. It builds a domain architectural model representing the commonalities and differences in requirements within the domain (problem space) [ARC 96].

Glossary Term

Domain design

the activity that takes the results of domain analysis to identify and generalize solutions for those common requirements in the form of a Domain-Specific Software Architecture (DSSA). It focuses on the problem space, not just on a particular system's requirements, to design a solution (solution space) [ARC 96].

Glossary Term

Domain engineering

the process of analysis, specification and implementation of software assets in a domain which are used in the development of multiple software products [SEI 96]. The three main activities of domain engineering are: domain analysis, domain design, and domain implementation [ARC 96].

Glossary Term

Domain implementation

the activity that realizes the reuse opportunities identified during domain analysis and design in the form of common requirements and design solutions, respectively. It facilitates the integration of those reusable assets into a particular application [ARC 96].

Glossary Term

Effectiveness

the degree to which a system's features and capabilities meet the user's needs.

Glossary Term

Error handling

the function of a computer system or component that identifies and responds to user or system errors to maintain normal or at the very least degraded operations.

Glossary Term

Error proneness

the degree to which a system may allow the user to intentionally or unintentionally introduce errors into or misuse the system.

Glossary Term

Error tolerance

the ability of a system or component to continue normal operation despite the presence of erroneous inputs [IEEE 90].

Glossary Term

Expandability

see Extendability [IEEE 90].

Glossary Term

Fail safe

pertaining to a system or component that automatically places itself in a safe operating mode in the event of a failure [IEEE 90].

Glossary Term

Fail soft

pertaining to a system or component that continues to provide partial operational capability in the event of certain failures [IEEE 90].

Glossary Term

Fault

an incorrect step, process, or data definition in a computer program [IEEE 90].

Glossary Term

Fault tolerance

the ability of a system or component to continue normal operation despite the presence of hardware or software faults [IEEE 90].

Glossary Term

Fidelity

the degree of similarity between a model and the system properties being modeled [IEEE 90].

Glossary Term

Functional scope

the range or scope to which a system component is capable of being applied.

Glossary Term

Functional testing

testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. Synonym: black-box testing [IEEE 90].

Glossary Term

Generality

the degree to which a system or component performs a broad range of functions [IEEE 90].

Glossary Term

Graphics

methods and techniques for converting data to or from graphic display via computers [McDaniel 94].

Glossary Term

Hardware maintenance

the cost associated with the process of retaining a hardware system or component in, or restoring it to, a state in which it can perform its required functions.

Glossary Term

Human Computer Interaction

a subfield within computer science concerned with the design, evaluation, and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them [Toronto 95].

Glossary Term

Human engineering

the extent to which a software product fulfills its purpose without wasting user's time and energy or degrading their morale [Boehm 78].

Glossary Term

Implementation phase

the period of time in the software life cycle during which a software product is created from design documentation and debugged [IEEE 90].

Glossary Term

Incompleteness

the degree to which all the parts of a software system or component are not present and each of its parts is not fully specified or developed.

Glossary Term

Information Security

the concepts, techniques, technical measures, and administrative measures used to protect information assets from deliberate or inadvertent unauthorized acquisition, damage, disclosure, manipulation, modification, loss, or use [McDaniel 94].

Glossary Term

Installation and checkout phase

the period of time in the software life cycle during which a software product is integrated into its operational environment and tested in this environment to ensure it performs as required [IEEE 90].

Glossary Term

Integration testing

testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them [IEEE 90].

Glossary Term

Interfaces design

the activity concerned with the interfaces of the software system contained in the software requirements and software interface requirements documentation. Consolidates the interface descriptions into a single interface description of the software system [IEEE 91].

Glossary Term

Interface testing

testing conducted to evaluate whether systems or components pass data and control correctly to one another [IEEE 90].

Glossary Term

ISO

International Organization for Standardization. A voluntary, non-treaty organization founded in 1946 which is responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of the 89 member countries, including ANSI for the U.S.

Glossary Term

Latency

the length of time it takes to respond to an event [Barbacci 95].

Glossary Term

Lifetime of operational capability

the total period of time in a system's life that it is operational and meeting the user's needs.

Glossary Term

Maintenance control

the cost of planning and scheduling hardware preventive maintenance, and software maintenance and upgrades, managing the hardware and software baselines, and providing response for hardware corrective maintenance.

Glossary Term

Maintenance measures

a category of quality measures that address how easily a system can be repaired or changed.

Glossary Term

Maintenance personnel

the number of personnel needed to maintain all aspects of a computer system, including the support personnel and facilities needed to support that activity.

Glossary Term

Managed device

any type of node residing on a network, such as a computer, printer or routers that contain a management agent.

Glossary Term

Managed object

a characteristic of a managed device that can be monitored, modified or controlled.

Glossary Term

Management agent

software that resides in a managed device that allows the device to be monitored and/or controlled by a network management application.

Glossary Term

Manufacturing phase

the period of time in the software life cycle during which the basic version of a software product is adapted to a specified set of operational environments and is distributed to a customer base [IEEE 90].

Glossary Term

Model

an approximation, representation, or idealization of selected aspects of the structure, behavior, operation, or other characteristics of a real-world process, concept, or system. Note: Models may have other models as components [IEEE 90].

Glossary Term

Necessity of characteristics

the degree to which all of the necessary features and capabilities are present in the software system.

Glossary Term

Need satisfaction measures

a category of quality measures that address how well a system meets the user's needs and requirements.

Glossary Term

Network management

the execution of the set of functions required for controlling, planning, allocating, deploying, coordinating, and monitoring the resources of a computer network [ITS 96].

Glossary Term

Network management application
application that provides the ability to monitor and control the network.

Glossary Term

Network management information

information that is exchanged between the network management station(s) and the management agents that allows the monitoring and control of a managed device.

Glossary Term

Network management protocol

protocol used by the network management station(s) and the management agent to exchange management information.

Glossary Term

Network management station
: system that hosts the network management application.

Glossary Term

Openness

the degree to which a system or component complies with standards.

Glossary Term

Operability

the ease of operating the software [Deutsch 88].

Glossary Term

Operational testing

testing conducted to evaluate a system or component in its operational environment [IEEE 90].

Glossary Term

Operations personnel

the number of personnel needed to operate all aspects of a computer system, including the support personnel and facilities needed to support that activity.

Glossary Term

Operations system

the cost of environmentals, communication, licenses, expendables, and documentation maintenance for an operational system.

Glossary Term

Operations and maintenance phase

the period of time in the software life cycle during which a software product is employed in its operational environment, monitored for satisfactory performance, and modified as necessary to correct problems or to respond to changing requirements [IEEE 90].

Glossary Term

Organizational measures

a category of quality measures that address how costly a system is to operate and maintain.

Glossary Term

Parallel computing

a computer system in which interconnected processors perform concurrent or simultaneous execution of two or more processes [McDaniel 94].

Glossary Term

Performance measures

a category of quality measures that address how well a system functions.

Glossary Term

Performance testing

testing conducted to evaluate the compliance of a system or component with specified performance requirements [IEEE 90].

Glossary Term

Protocol

a set of conventions that govern the interaction of processes, devices, and other components within a system [IEEE 90].

Glossary Term

Provably correct

the ability to mathematically verify the correctness of a system or component.

Glossary Term

Qualification phase

the period of time in the software life cycle during which it is determined whether a system or component is suitable for operational use.

Glossary Term

Qualification testing

testing conducted to determine whether a system or component is suitable for operational use [IEEE 90].

Glossary Term

Quality measure

a software feature or characteristic used to assess the quality of a system or component.

Glossary Term

Readability

the degree to which a system's functions and those of its component statements can be easily discerned by reading the associated source code.

Glossary Term

Responsiveness

the degree to which a software system or component has incorporated the user's requirements.

Glossary Term

Recovery

the restoration of a system, program, database, or other system resource to a prior state following a failure or externally caused disaster; for example, the restoration of a database to a point at which processing can be resumed following a system failure [IEEE 90].

Glossary Term

Reengineering

rebuilding a software system or component to suit some new purpose; for example to work on a different platform, to switch to another language, to make it more maintainable.

Glossary Term

Regression testing

selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements [IEEE 90].

Glossary Term

Requirements engineering

involves all life-cycle activities devoted to identification of user requirements, analysis of the requirements to derive additional requirements, documentation of the requirements as a specification, and validation of the documented requirements against user needs, as well as processes that support these activities [DoD 91].

Glossary Term

Requirements growth

the rate at which the requirements change for an operational system. The rate can be positive or negative.

Glossary Term

Requirements phase

the period of time in the software life cycle during which the requirements for a software product are defined and documented [IEEE 90].

Glossary Term

Requirements tracing

describing and following the life of a requirement in both forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases) [Gotel 95].

Glossary Term

Resource utilization

the percentage of time a resource (CPU, Memory, I/O, Peripheral, Network) is busy [Barbacci 95].

Glossary Term

Restart

to cause a computer program to resume execution after a failure, using status and results recorded at a checkpoint [IEEE 90].

Glossary Term

Retirement phase

the period of time in the software life cycle during which support for a software product is terminated [IEEE 90].

Glossary Term

Reverse engineering

the process of analyzing a system's code, documentation, and behavior to identify its current components and their dependencies to extract and create system abstractions and design information. The subject system is not altered; however, additional knowledge about the system is produced.

Glossary Term

Robustness

the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions [IEEE 90].

Glossary Term

Safety

a measure of the absence of unsafe software conditions. The absence of catastrophic consequences to the environment [Barbacci 95].

Glossary Term

Select or develop algorithms

the activity concerned with selecting or developing a procedural representation of the functions in the software requirements documentation for each software component and data structure. The algorithms shall completely satisfy the applicable functional and/or mathematical specifications [IEEE 91].

Glossary Term

Self-descriptiveness

the degree to which a system or component contains enough information to explain its objectives and properties [IEEE 90].

Glossary Term

Simplicity

the degree to which a system or component has a design and implementation that is straightforward and easy to understand [IEEE 90].

Glossary Term

Software architecture

the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time [Clements 96].

Glossary Term

Software change cycle time

the period of time that starts when a new system requirement is identified and ends when the requirement has been incorporated into the system and delivered for operational use.

Glossary Term

Software life cycle

the period of time that begins when a software product is conceived and ends when the software is no longer available for use. The life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and sometimes, retirement phase. These phases may overlap or be performed iteratively, depending on the software development approach used [IEEE 90].

Glossary Term

Software maintenance

the cost associated with modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment.

Glossary Term

Software migration and evolution
see Adaptive maintenance.

Glossary Term

Software upgrade and technology insertion
see Perfective maintenance.

Glossary Term

Speed

the rate at which a software system or component performs its functions.

Glossary Term

Statistical testing

employing statistical science to evaluate a system or component. Used to demonstrate a system's fitness for use, to predict the reliability of a system in an operational environment, to efficiently allocate testing resources, to predict the amount of testing required after a system change, to qualify components for reuse, and to identify when enough testing has been accomplished [Poore 96].

Glossary Term

Structuredness

the degree to which a system or component possesses a definite pattern of organization of its interdependent parts [Boehm 78].

Glossary Term

Sufficiency of characteristics

the degree to which the features and capabilities of a software system adequately meet the user's needs.

Glossary Term

Survivability

the degree to which essential functions are still available even though some part of the system is down [Deutsch 88].

Glossary Term

System allocation

mapping the required functions to software and hardware. This activity is the bridge between concept exploration and the definition of software requirements [IEEE 91].

Glossary Term

System analysis and optimization

a systematic investigation of a real or planned system to determine the information requirements and processes of the system and how these relate to each other and to any other system, and to make improvements to the system where possible.

Glossary Term

System security

a system function that restricts the use of objects to certain users [McDaniel 94].

Glossary Term

System testing

testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements [IEEE 90].

Glossary Term

Taxonomy

a scheme that partitions a body of knowledge and defines the relationships among the pieces. It is used for classifying and understanding the body of knowledge [IEEE 90].

Glossary Term

Test

an activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component [IEEE 90].

Glossary Term

Test drivers

software modules used to invoke a module(s) under test and, often, provide test inputs, control and monitor execution, and report test results [IEEE 90].

Glossary Term

Test phase

the period of time in the software life cycle during which the components of a software product are evaluated and integrated, and the software product is evaluated to determine whether or not requirements have been satisfied [IEEE 90].

Glossary Term

Test tools

computer programs used in the testing of a system, a component of the system, or its documentation. Examples include monitor, test case generator, timing analyzer [IEEE 90].

Glossary Term

Testing

the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component [IEEE 90].

Glossary Term

Unit testing

testing of individual hardware or software units or groups of related units [IEEE 90].

Glossary Term

Training

Provisions to learn how to develop, maintain, or use the software system.

Glossary Term

Trouble report analysis

the methodical investigation of a reported operational system deficiency to determine what, if any, corrective action needs to be taken.

Glossary Term

Upgradeability
see Evolvability.

Glossary Term

User interface

an interface that enables information to be passed between a human user and hardware or software components of a computer system [IEEE 90].

Glossary Term

Verifiability

the relative effort to verify the specified software operation and performance [Evans 87].

Glossary Term

Vulnerability

the degree to which a software system or component is open to unauthorized access, change, or disclosure of information and is susceptible to interference or disruption of system services.

Submit Comments

Software Technology Roadmap

Submitter and Contact Information

Please enter the following information. We need it to process your comments. (Required fields are in boldface.)

Submitter name:

Job Title:

Organization:

Phone:

FAX:

E-mail:

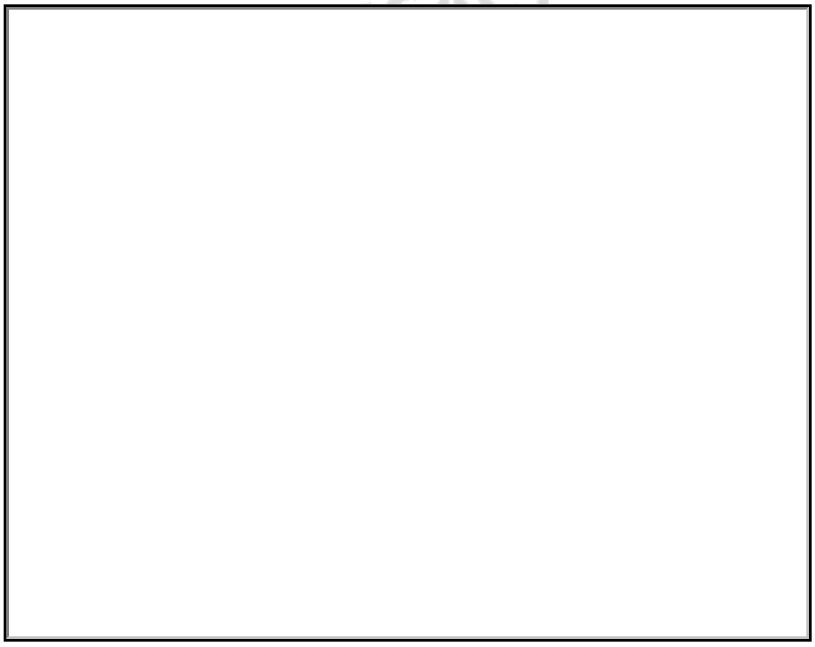
Comments or Additional Information

Name of Technology:

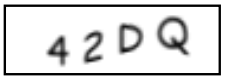
Please enter below any comments or additional information you have about this technology, such as:

- Suggested correction of information in the technology description
- Results (good or bad) of using this technology
- New releases of actual and defacto industry standards

Whenever possible, please provide justifications and/or references to support your comments.



To prevent automated submissions, please enter the text shown on the image as the verification code below:



Submit Comments | Erase Everything

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/feedback/comments-description.html
Last Modified: 24 July 2008

Submit Comments

Software Technology Roadmap

Submitter and Contact Information

Please enter the following information. We need it to process your comments. (Required fields are in boldface.)

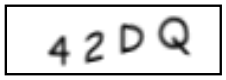
Submitter name:	
Job Title:	
Organization:	
Phone:	
FAX:	
E-mail:	

Your Comments

Please enter below any comments you have about the STR, such as:

- How the STR has helped you to accomplish your job
- In what context you have used the STR (names of projects, programs, efforts, and application domains will be kept confidential)
- What sections you have found useful
- Whether or not you have found the STR easy to use
- How the STR could be improved

To prevent automated submissions, please enter the text shown on the image as the verification code below:



Submit Comments | Erase Everything

URL: http://www.sei.cmu.edu/str/feedback/comments-overall.html
Last Modified: 24 July 2008

Submitting a New Technology Description

Software Technology Roadmap

Thank you for your interest in becoming an author for the Software Technology Review!

Potential topics are listed below. If you are interested in writing a technology description on any of these or on another topic, please send mail to str@sei.cmu.edu and review Guidelines for Authoring a Technology Description.

Potential Topics for the Software Technology Review

- Agents/Agent-Based Computing
- Algebraic Specification Method
- AI/Expert Systems
- ATM
- Bindings
- Bowles Metrics
- C Programming Language
- C++ Programming Language
- COCOMO Method
- Collaboration Technologies
- Common LISP Object System (CLOS) Programming Language
- Comparative/Taxonomic Modeling
- Computer-Human Interface Technology
- Configuration Management
- Data Complexity
- Data Fusion
- Data Integrity
- Data Mining
- Data Warehousing
- Design Complexity
- Dynamic Languages
- Dynamic Simulation
- Object-Oriented Programming Language*
- Object Pascal Programming Language
- Objective C Programming Language
- OSI
- Parallel Processing Software Architecture
- Peer Reviews
- PERL Programming Language
- POSIX
- Probabilistic Automata
- Program Slicing
- Program Understanding
- Project Support Environment Reference Model (PSERM)
- Public Key Cryptography
- Rationale Capture Overview
- Real-Time Computing
- Real-Time Operating Systems
- Redundant Test Case Elimination
- Regression Testing Techniques
- Relational DBMS
- Remote Data Access (RDA)

- Eiffel Programming Language
- Electronic Encryption Key Distribution
- Encryption
- End-to-End Encryption
- Entity-Relationship Modeling
- Essential Complexity
- Essential Systems Analysis
- European Computer Manufacturers Association Reference Model [ECMA]
- Fault Tolerant Computing
- File Server Software Architecture
- Finite State Automata
- Formal Methods
- Functional Decomposition
- Henry and Kafura Metrics
- HTML Programming Language
- Interface Definition Language
- Joint Technical Architecture (JTA)
- Legacy Systems Migration/ Evolution
- Ligier Metrics
- LISP Programming Language
- Mediators
- Model Checking
- Motif User Interface Language (UIL)
- MSSI (NSA)
- Multimedia
- Network Auditing Techniques
- Network Security Guards
- Network Simulation
- Neural Networks
- Object-Oriented Analysis*
- Object-Oriented Database*
- Object-Oriented Design*
- Representation and Maintenance of Process Knowledge Method
- Resolution-Based Theorem Proving
- Risk Management
- Security (Guards, Compartmented Mode Workstations)
- Session-Based Technology
- Simula Programming Language
- Smalltalk Programming Language
- Software Architecture Overview
- Software Generation Systems
- Software Reliability Modeling and Analysis
- Software Reuse
- Specification Construction Techniques
- SQL
- Statistical Test Plan Generation and Coverage Analysis Techniques
- Stochastic Methods
- Structured Analysis and Design
- Systems Engineering Tools
- TCL Programming Language
- TCP/IP
- Test and Analysis Tool Generation
- Test Case Generation
- Test Data Generation by Chaining
- Testing Technologies Overview
- Troy and Zweben Metric
- Trusted Computing Base
- Virtual Reality
- Visual Programming Techniques
- X.25
- Web-Based Computing/Software Development
- Web Security
- Window Managers
- Wrappers

*Technology description currently exists, but needs to be substantially rewritten/ expanded.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/feedback/new-description.html
Last Modified: 24 July 2008

References and Information Sources

[ANSI 83] ANSI/MIL-STD-1815A-1983. *Reference Manual for the Ada Programming Language.* New York, NY: American National Standards Institute, Inc., 1983.

References and Information Sources

[AdaIC 96] *AdaIC NEWS* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/news/> (1996).

References and Information Sources

[Syiek 95] Syiek, David. "C vs. Ada: Arguing Performance Religion." *ACM Ada Letters 15*, 6 (November/December 1995): 67-9.

References and Information Sources

[Hefley 92] Hefley, W.; Foreman, J.; Engle, C.; & Goodenough, J. *Ada Adoption Handbook: A Program Manager's Guide* Version 2.0 (CMU/SEI-92-TR-29, ADA258937). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.

References and Information Sources

[Engle 96] Engle, Chuck. *Re[2]: Ada 83/Ada 95* [email to Gary Haines], [online]. Available email: ghaines@spacecom.af.mil (August 19, 1996).

References and Information Sources

[Compilers 96] *Ada 83 Validated Compilers List* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/AdaIC/compilers/83val/83vcl.txt> (August 1996).

References and Information Sources

[Holzer 96] Holzer, Robert. "Sea Trials Prompt U.S. Navy to Tout Seawolf Sub's Virtues," *Defense News 11*, 28 (July 15-20, 1996): 12.

References and Information Sources

[Pehrson 96] Pehrson, Ron J. *Software Development for the Boeing 777* [online]. Available WWW <URL: http://www.stsc.hill.af.mil/CrossTalk/1996/jan/Boein777.html> (1996).

References and Information Sources

[ReuseIC 95] *Boeing 777: Flying High with Ada and Reuse* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/ReuseIC/pubs/flyers/boe-reus.shtml> (1995).

References and Information Sources

[Zeigler 95] Zeigler, Stephen F. *Comparing Development Costs of C and Ada* [online]. Available WWW <URL: http://sw-eng.falls-church.va.us/> (1995).

References and Information Sources

[IEEE 87] IEEE Std 1002-1987. *IEEE Standard Taxonomy for Software Engineering Standards.* New York, NY: Institute of Electrical and Electronics Engineers, 1987.

Guidelines for Authoring a Technology Description

The purpose of this document is to provide guidance to those writing technology descriptions for the *Software Technology Roadmap.* It provides background information about the document and guides authors through the development and review processes for technology descriptions.

1 Introduction

The *Software Technology Roadmap* (STR) is a reference document containing the latest information, in the form of technology descriptions, on approximately 63 software technologies. The STR is of interest to anyone building or maintaining systems. For more information about the background and audience of the STR, please see Appendix A.

The purpose of a technology description is to

- identify a technology
- characterize it in terms of the properties of systems and measures of software quality that it affects
- point out tradeoffs, benefits, risks and limitations that may arise in various situations of use

Each technology description also provides reference(s) to literature, indications of the current maturity of the technology, and cross references to related technologies.

Technology descriptions are not meant to be comprehensive--each description should provide the reader with enough knowledge to decide whether to investigate the technology further, to find out where to go for more information,

and to know what questions to ask in gathering more information.

Typically, technology descriptions range in size from six to eight pages, depending on the amount of information available or the maturity of the technology.

For other examples of technology descriptions, please review the hard copy document (CMU/SEI-97-HB-001) or visit the STR Web site. The Web site always has the most recent version of the technology descriptions.

[Top] [Prev] [Next] [Bottom]

URL: http://www.sei.cmu.edu/str/feedback/guide/guide.ch01.html
Last Modified: 24 July 2008

2 Writing a Technology Description

This list provides a quick overview of the process involved in creating a technology description; details are to follow:

1. contact us
2. obtain template
3. write description
4. submit description
5. undergo review cycle
6. enter maintenance phase

If you are interested in submitting a technology description, please send email to str@sei.cmu.edu.

Each technology description follows a structured template; to obtain a copy of the template, send email to str@sei.cmu.edu or refer to /str/descriptions/template/template.html.

A description of each section of the template follows.

Status. Each technology description begins with a status indicator. This status indicator provides an assessment of the overall quality and maturity of the technology description. One of four indicators is assigned by the STR staff: Draft, In Review, Advanced, or Complete. All technology descriptions begin in Draft status. (For a more detailed description of these states, please see Section 2.5.)

Note. Include this section if prerequisite or follow-on reading is recommended. The prerequisites are usually technology descriptions that provide an overview of the general topic area and establish a context for the different technologies in the area.

Purpose and Origin. This section should provide a general description and brief background of the technology. It should include what capability or benefit was anticipated for the technology when originally conceived, and identify common aliases for the technology as well as its originator(s) or key developer(s) (if known).

Technical Detail. This section should answer--succinctly--the question, "What does the technology do?" It should include the salient quality measures (see Taxonomy Categories) that are influenced by the technology in all situations and describe the tradeoffs that are enabled by the technology. It may also provide some insight into why the technology works and what advances are expected. Since the STR is not a "how-to" manual, do not provide any implementation details.

Usage Considerations. This section should provide insight for the use of the technology. Issues to be addressed include

- example applications into which this technology may be incorporated (or should not be incorporated); for instance, "this technology, because of its emphasis on synchronized processing, is particularly suited for real-time applications"
- quality measures that may be influenced by this technology, depending on the particular context in which the application is employed

Maturity. The purpose of this section is to provide an indication as to how well-developed the technology is. (A technology that was developed a year or two ago and is still in the experimental stage--or still being developed at the university research level--will likely be more difficult to adopt than one that has been in use in many systems for a decade.) It is *not* the intent of this document to provide an absolute measure of maturity, but to provide enough information *to allow the reader to make an informed judgment as to the technology's maturity for their application area*. Details that will help in this determination include

- the extent to which the technology has been incorporated into real systems, tools, or commercial products
- the success that developers have had in adopting and using the technology
- notable failures of the technology (if any)

Other information that might appear in this section includes trend information, such as a projection of the technology's long term potential; observations about the rate of maturation; and implications of rapid maturation.

Costs and Limitations. This section should point out limitations and costs of using a particular technology. Some examples of the kinds of costs and limitations associated with a technology are the following: a technology may impose an otherwise unnecessary interface standard; it might require investment in other technologies (see Dependencies below); it might require investment of time or money; or it may directly conflict with security or real-time requirements. Specific items to discuss include

- what is needed to adopt this technology (this could mean training requirements, skill levels needed, programming languages, or specific architectures)?
- how long it takes to incorporate or implement this technology?
- barriers to the use of this technology
- reasons why this technology would not be used

Dependencies. This section should identify other technologies that influence or are influenced by the technology being described. You should only include dependencies for which significant influence in either direction is expected. You should also provide an indication as to why the dependency exists (usually in terms of quality measure or usage consideration). If the dependent technology appears in the document, provide a cross-reference to it. Omit this paragraph if no dependencies are known.

Alternatives. An alternative technology is one that could be used for the same purposes as the technology being described. A technology is an alternative if there is any situation or purpose for which both technologies are viable or likely to be considered candidates. Alternatives may represent a simple choice among technologies that achieve the same solution to a problem, or they may represent completely different approaches to the problem being addressed by the technology.

For each alternative technology, provide a concise description of the situations for which it provides an alternative. Also provide any special considerations that could help in selecting among alternatives. If the alternative technology appears in the document, provide a cross-reference to it.

Alternative technologies are distinct from dependent or complementary technologies, which must be used in combination with the technology being described to achieve the given purpose.

Complementary Technologies. A complementary technology is one that enhances or is enhanced by the technology being described, but for which neither is critical to the development or use of the other (if it were critical, then it would appear in the "Dependencies" section above). Typically, a complementary technology is one that--in combination with this technology--will achieve benefits or capabilities that are not obvious when the technologies are considered separately. For each complementary technology, provide a concise description of the conditions under which it is complementary and the additional benefits that are provided by the combination. If the complementary technology appears in the document, provide a cross-reference to it.

Taxonomy Categories. We have created several taxonomies to categorize technology descriptions. These taxonomies are:

- *Application taxonomy*. This taxonomy refers to how this technology would be employed, either in support of operational systems (perhaps in a particular phase of the life cycle) or in actual operation of systems (for example, to provide system security).
- *Quality measures taxonomy*. This is a list of those quality attributes (e.g.,

reliability or responsiveness) that are influenced in some way by the application of this technology.

- Computing Reviews taxonomy: This taxonomy describes the technical subdiscipline within Computer Science into which the technology falls. The category is based on the *ACM Computing Reviews* Classification System developed in 1991 (and currently undergoing revision). A complete description of the Classification System and its contents can be found in any January issue of *Computing Surveys* or in the annual *ACM Guide to Computing Literature*.

For each of these taxonomies, you should suggest terms under which your technology description can be classified. See our Web site for the full taxonomies: /str/taxonomies/

References and Information Sources. The final section in each technology description provides bibliographic information. You should include sources cited in the technology description, as well as pointers to additional resources that a reader can go to for more information. You should designate one to four key references with an asterisk (*). Key references are those that will best assist a reader in learning more about the technology.

Current Author/Maintainer. The author(s)/maintainer(s) of the current version of the technology description are listed in this section. The only exceptions are Draft technology descriptions, which are published without an author's name.

Internal Team Reviewer(s). Name(s) of those on the project team who reviewed the technology description. (This section will not appear in the published version of the technology description.)

External Reviewer(s). This section contains names of external experts who have reviewed this technology description. If no "External Reviewer(s)" heading is present, then an external review has not occurred. Note: if the preface "candidate" is used, the individual has been suggested as a reviewer, but has not yet agreed to participate.

Keyword Index. You should provide a list keywords under which this technology may be indexed (or indicate those by making a notation in the text). This section of the templates indicates whether keywords were chosen for this technology description. This step is not necessary until a technology description reaches the "In Review" state. (This section will not appear in the published version of the technology description.)

Future. This section includes items to be considered as part of the future evolution of this technology description. (This section will not appear in the published version of the technology description.)

Background/Support. Provides an explanation for some key piece of information in the technology description. (This section will not appear in the published version of the technology description.)

Modifications. This area lists the modification history of the technology

description and includes the names of contributing authors from earlier versions of the description.

Pending. A known item that needs to be addressed in future versions of the description. These are posted (when known) so that the reader can pursue these items on their own if necessary.

We ask that you follow these guidelines when writing and submitting new material to the STR. If you have any questions about these guidelines, send email to str@sei.cmu.edu.

2.3.1 Electronic Format

All technology descriptions should be submitted in ASCII (plain text) on a disk or by email. Note: You may develop the description in a word processor if you prefer, but you should save your technology description in a plain text format before you submit the file.

2.3.2 Graphics

Any graphics used in technology descriptions should be drawn in either PowerPoint or FrameMaker and submitted on a disk or by email.

2.3.3 Cross-References and Minor Formatting

When working in ASCII text, use the following notations to indicate cross-references and formatting conventions.

To denote cross-references (to other technology descriptions, etc.), use the format [xref: <name of technology description, heading, etc.>]. For example

See object-oriented programming languages [xref: object-oriented programming languages] for more information about this topic.

The Cleanroom [xref: Cleanroom] technology description covers this in more detail. To denote bold and italics, use the formats <b text to be bold /b> for bold text, <i text to be italicized /i> for italicized text. For example

This is <b not /b> an object-oriented programming language.

This technology supports <i maintainability /i> of large-scale software systems. To denote terms you would like to have included in the Keyword Index, use the format [index: <term to be indexed>].

See object-oriented programming languages [index: object-oriented programming languages] for more information about this topic.

2.3.4 References and Information Sources

Please follow these guidelines when including references and information sources in your technology descriptions.

- Include only published documents, i.e., do not cite draft documents or those "to be published." If you must reference an unpublished document, use a footnote.
- Include only publicly-available documents.
- To the extent possible, follow the reference guidelines below.

Note: Don't forget to indicate key references with an asterisk.

Citations in text should appear in brackets and contain the last name of the author, editor, or publishing organization by which the document is identified in the reference list followed by a space and the last two digits of the year of publication.

[Brown 89]

[IEEE 90]

The only legitimate function of a citation is to refer the reader to the list at the end of the document. Therefore, a citation should not be a semantic element of the sentence in which it occurs. All sentences should be complete without citations. [wrong] The process is described in [Brown 89].

[right] Brown has described this process [Brown 89].

For consecutive citations, use a single bracket.

[Brown 89, IEEE 90]

For consecutive citations by the same author, use a single bracket but enclose the complete citations:

[Brown 89, Brown 90]

The following examples offer formatting information for several types of references. Please make an effort to include all of the information requested for each reference; we need to have the information before the technology

description can be published.

Citing Books

Template

[<citation>] <last name>, <first name> <middle initial>. <*title*>. <city, state of publication>: <publisher's name>, <date of publication>.

Example

[Yourdon 89] Yourdon, Edward N. *Modern Structured Analysis*. Englewood Cliffs, N.J.: Prentice-Hall, 1989.

Citing Technical and Research Reports

Template

[<citation>] <last name>, <first name> <middle initial>. <*title of report*> (<report number>, <DTIC number when available>). <city and state of publication>: <publisher's name>, <date of publication>.

Example

[Graham 89] Graham, Marc H. *Guidelines for the Use of SAME* (CMU/SEI-89-TR-16, ADA228027). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1989.

Citing Journals

Template

[<citation>] <last name>, <first name> <middle initial>. "<title of article>." <*journal title*> <*volume number*>, <number of issue> (<month or season and year of issue>): <inclusive page numbers>.

Example

[Bohm 66] Bohm, Charles. "Flow Diagrams, Turing Machines, and Languages With Only Two Formation Rules." *Communications of the ACM 8*, 5 (May 1966): 366-371.

Citing WWW sites

Template

[<citation>] <last name>, <first name> <middle initial>. <*title*> [online]. Available <FTP/Telnet/WWW>: <<URL>> <(year of publication)>.

Example

[Rogers 92] Rogers, Robin. *User's Guide to the Beaches of Southern California* [online]. Available WWW <URL: http://www.acme.com:/rogers/docs/beaches/so_cal/ug> (1992).

Citing Meetings and Symposia

Template

[<citation>] <last name>, <first name> <middle initial>. "<title of the article>," <page numbers>. *Proceedings of <name of the meeting or symposium>*. <city and state of meeting or symposium>, <date(s) and year of meeting or symposium>. <city, state of publication>: <publisher's name>, <date of publication>.

Example

[Kaiser 89] Kaiser, G. "Mechanisms," 256-275. *Proceedings of the 5th International Software Process Workshop*. Kennebunkport, Maine, Oct. 10-13, 1989. New York: IEE Computer Society Press, 1990.

2.3.5 Editing

All technology descriptions will be edited for consistency and conformance to the *SEI Style Guide*. Any substantial edits will be subject to the approval of the author and/or the STR review staff. To obtain a copy of the style guide, send mail to str@sei.cmu.edu.

See Appendix B for a pre-submission checklist. By following the checklist, you can make sure your submissions are complete before you send them in.

Email submissions to: str@sei.cmu.edu

Mail submissions to: Lauren Heinz
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-3890

In order to sustain credibility and accuracy, each technology description goes through a review process. Please see Appendix C for a copy of the checklist used by reviewers. (The checklist may help you as you write your technology description.)

The review cycle includes the following steps:

- review by technology cluster leader (if one exists)
- internal team review*
- external review

While in the review cycle, a technology description progresses through the following stages: Draft, In Review, Advanced, and Complete. Each of the four status indicators is explained below:

Draft technology descriptions have the following attributes:

- They need more work.
- They have generally not been reviewed.
- Overall assessment: While technology descriptions labeled "Draft" will contain some useful information, readers should not rely on these descriptions as their only source of information about the topic. Readers should consider these descriptions as starting points for conducting their own research about the technology.

In Review technology descriptions have the following attributes:

- They are thought to be in fair technical shape.
- They have begun an internal review cycle*.
- They may have major issues that must be resolved, or some sections that may require additional text.
- Relevant keywords have been added to the Keyword Index.
- Overall assessment: Readers can get some quality information from these, but because these descriptions have not been completely reviewed, readers should explore some of the references for additional information and consider conducting their own research about the technology.

Advanced technology descriptions have the following attributes:

- They are in good technical shape.
- Internal review has occurred.
- There are minor issues to be worked, but it is generally polished.
- They are subject to additional review by external reviewers.
- Relevant keywords have been added to the Keyword Index.
- Overall assessment: These descriptions are in rather good shape, but because they have not been through external review, readers should exercise some caution.

Complete technology descriptions have the following attributes:

- At least one expert external review has occurred, and issues from that review have been resolved.
- Relevant keywords have been added to the Keyword Index.
- No additional work is necessary at this time.
- Overall assessment: These technology descriptions are believed to be complete and correct. They would be revised in the future based on additional external reviewers, new information, and public feedback.

Please note: Once the technology description has been formatted in FrameMaker and coded in HTML, we will accept submissions as hard-copy markups only, unless the technology description has been sufficiently revised to warrant total replacement.

* Internal review cycle refers to the review process that takes place within the development/editorial team.

[Top] [Prev] [Next] [Bottom]

URL: http://www.sei.cmu.edu/str/feedback/guide/guide.ch02.html
Last Modified: 24 July 2008

Appendix C: Reviewer Checklist

Check Yes or No to indicate whether or not the following criteria have been satisfied. If you check No for an item, please document why in the Reviewer's Notes and Comments to STR Staff section below. The author will use your comments for future revisions.

	Yes	No
There is a clear message of **value** to the reader	____	____
The technology description contains information that is **relevant** to the reader, whether or not the reader is already aware of its relevance	____	____
The technology description **does not** discuss specific **products** or compare products	____	____
The technology description includes **all the prescribed information**	____	____
sufficient level of technical detail	____	____
pointers to literature	____	____
applications employing technology description	____	____
quality measure influenced	____	____
usage contexts and considerations	____	____
maturity indication, including trends	____	____
costs and limitations	____	____
all appropriate cross-references	____	____
The technology description is as **concise** as possible	____	____
irrelevant issues are not discussed	____	____
words and phrases have a purpose	____	____
words and phrases add meaning	____	____

points made have value to the reader	____	____
information is not redundantly provided	____	____
hype is excluded	____	____
Most **new** readers will go away with information they did not previously have	____	____
The technology description assists the reader in **making tradeoffs**	____	____
The reader can tell **whether to investigate** further	____	____
The reader can tell **where to investigate** further	____	____
The reader who wants to further pursue this topic will know **where to look next**	____	____
The technology description **does not advocate**, it only characterizes	____	____
The technology description gives a **balanced** and **unbiased** treatment of the subject matter	____	____
The technology description gives the reader confidence	____	____
in the claims being made	____	____
evidence is provided or indicated	____	____
quality of sources and evidence is indicated	____	____
short examples or illustrations are given	____	____
Limits of the technology are clearly stated	____	____

Reviewer's Notes and Comments to STR Staff:

_____	_____
reviewer's name	review date
_____	_____
reviewer's organization	reviewer's phone number
_____	_____
reviewer's title	reviewer's email address

Submit Reviewer Checklist by email to: str@sei.cmu.eduSubmit Reviewer Checklist by US mail to:

Software Technology Review Administrator
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-3890

[Top] [Prev] [Next] [Bottom]

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/feedback/guide/guide.appc.html
Last Modified: 24 July 2008

I forgot to put in a title

Software Technology Roadmap

[Top] [Prev] [Next] [Bottom]

Appendix A: Background of the STR

The Air Force acquisition community tasked the Software Engineering Institute (SEI) to create a *reference document* that would provide the Air Force with a better understanding of software technologies. This knowledge will allow the Air Force to systematically plan the research and development (R&D) and technology insertion required to meet current and future Air Force needs, from the upgrade and evolution of current systems to the development of new systems.

The document is intended to be a guide to specific software technologies of interest to those building or maintaining systems, especially those in command, control, and/or communications applications. The document has many goals:

-
- to provide common ground by which contractors, commercial companies, researchers, government program offices, and software maintenance organizations may assess technologies
- to serve as *Cliffs Notes* for specific software technologies; to encapsulate a large amount of information so that the reader can rapidly read the basics and make a preliminary decision on whether further research is warranted
- to achieve objectivity, balance, and a quantitative focus, bringing out both shortcomings as well as advantages, and provide insight into areas such as costs, risks, quality, ease of use, security, and alternatives
- to layer information so that readers can find subordinate technology descriptions (where they exist) to learn more about the topic(s) of specific interest, and to provide references to sources of more detailed technical information, to include usage and experience

While the document provides balanced coverage of a wide scope of technologies, there are certain constraints on the content of the document:

-
- *Not prescriptive.* This document is not prescriptive; it does not make recommendations, establish priorities, or dictate a specific path/approach. The reader must make decisions about whether a technology is appropriate for a specific engineering and programmatic context depending on the planned intended use, its maturity, other technologies that will be used, the specific time frame envisioned, and funding constraints.
- *Not a product reference.* This document is not a survey or catalog of products. There are many reasons for this, including the rapid proliferation of products, the need to continually assess product capabilities, questions of perceived endorsement, and the fact that products are almost always a collection of technologies. It is up to the reader to decide which products are appropriate for their context. DataPro and Auerbach would likely be better sources of product-specific information.
- *Not an endorsement.* Inclusion or exclusion of a topic in this document does not constitute an endorsement of any type, or selection as any sort of "best technical practice." Judgements such as these must be made by the *readers* based on their contexts; our goal is to provide the balanced information to enable those judgements.
- *Not a market forecasting tool*. While the technology descriptions may project the effect of a technology and discuss trends, more complete technology market analysis and forecast reports are produced by organizations such as The Yankee Group, Gartner Group, and IDC.
- *Not a focused analysis of specific technical areas.* Various sources such as Ovum, Ltd. and The Standish Group offer reports on a subscription or one-time basis on topics such as workflow, open systems, and software project failure analyses, and may also produce specialized analyses and reporting on a consulting basis.

This document is relevant to many audiences. The audiences and a description of how each audience can use this document are shown in the table below.

User	Job Roles/Tasks	Document Capabilities/ Value

PEO/Executive Pentagon Action Officer	Acquisition oversight, funding advocacy Motivate introduction of new/commercial technologies Policy issues	Overview/introductory info Baseline reference document "Cliff Notes" approach--provides high-level, 6-8 page quick study Tradeoff information
System Program Manager (SPM) and Technical Staff (Includes FFRDCs (MITRE, etc.) and may include government laboratories)	Writes Request for Proposal (RFP) or some form of solicitation based on user requirements Reviews proposals and selects developers Manages development and/or maintenance work	All of previous category, plus: Taxonomies to aid in identifying alternatives Back pointers to high-level, related technologies Criteria and guidance for decision-making Tech transfer/insertion guidelines Selected high-value references to more technical information, to include usage and experience data Generally the sort of analysis and survey information that would not be accomplished under normal project circumstances

Developer (to include research and development (R&D) activity)	Performs advanced development, prototyping, and technology investigation focused on risk reduction and securing competitive advantage Concerned about transition and insertion issues Writes a proposal in response to solicitations Performs engineering development and provides initial operational system	Same as previous category.
Maintainer	Maintains operational system until the end of the life cycle Responds to user requirements for corrections or enhancements Concerned about inserting new technologies and migrating to different approaches	Same as previous category.
User	Communicates operational needs End customer for operational system	Communicates alternatives and risks, and provides perspective of what technology can (reasonably) provide

[Top] [Prev] [Next] [Bottom]

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

URL: http://www.sei.cmu.edu/str/feedback/guide/guide.appa.html
Last Modified: 24 July 2008

Appendix B: Technology Description Checklist

Please review this checklist before you submit a new technology description.

____ text is in ascii

____ images are in PowerPoint or FrameMaker

____ all relevant sections of the template have been addressed

____ cross-references are marked

____ references contain all required information

____ key references have been indicated with an asterisk

____ taxonomy categories have been assigned

____ keyword index terms are marked

Submit files by email to: str@sei.cmu.edu

Submit files by US mail to:

Software Technology Review Administrator
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-3890

URL: http://www.sei.cmu.edu/str/feedback/guide/guide.appb.html
Last Modified: 24 July 2008