IEEE Std 1517-1999

IEEE Standard for Information Technology—Software Life Cycle Processes—Reuse Processes

Sponsor

Software Engineering Standards Committee
of the
IEEE Computer Society

Approved 26 June 1999

IEEE-SA Standards Board

Abstract: A common framework for extending the software life cycle processes of IEEE/EIA Std 12207.0-1996 to include the systematic practice of software reuse is provided. This standard specifies the processes, activities, and tasks to be applied during each phase of the software life cycle to enable a software product to be constructed from reusable assets. It also specifies the processes, activities, and tasks to enable the identification, construction, maintenance, and management of assets supplied.
Keywords: asset, domain engineering, process, software life cycle, software reuse

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Print: ISBN 0-7381-1735-8 SH94751
PDF: ISBN 0-7381-1736-6 SS94751

Introduction

(This introduction is not part of IEEE Std 1517-1999, IEEE Standard for Information Technology—Software Life Cycle Processes—Reuse Processes.)

Implementing software reuse entails more than creating and using libraries of assets. It requires formalizing the practice of reuse by integrating reuse processes and activities into the software life cycle. Unless reuse is explicitly defined in the software life cycle processes, an organization will not be able to repeatedly exploit reuse opportunities in multiple software projects or products. Without this repetition, the improvement to the software life cycle and the software products that result from the practice of reuse will be limited and, perhaps, disappointing.

Systematic reuse is the practice of reuse according to a well-defined, repeatable process. Practicing systematic reuse enables significant software productivity, quality, and cost improvements. The major benefits that systematic reuse can deliver are as follows:

- Increase software productivity;
- Shorten software development time;
- Move personnel, tools, and methods more easily from project to project;
- Reduce software development and maintenance costs;
- Produce higher quality software products;
- Improve software product interoperability;
- Provide a competitive advantage to an organization that practices reuse.

There are a variety of approaches to implement the concept of reuse. What distinguishes systematic reuse is the avoidance of the proliferation of multiple versions of otherwise common elements. For example, if a reuse approach results in multiple instantiations of a common element, each of which may be modified by software developers, then the element is no longer common and can no longer be maintained as a single element apart from its instantiations. In the context of this standard, systematic reuse excludes those approaches.

The potential for reuse is enormous since the majority of each software product can be built with assets, assuming they are available. An asset is any item, such as a design or test plan, that has been designed to be used in multiple contexts, such as multiple software products, multiple implementations of a software product, or multiple software projects.

One major problem encountered by organizations attempting to practice reuse is that reuse is simply missing from their software life cycle processes. The intent of this standard is to address this problem by defining a common framework for reuse processes and by defining how to integrate the practice of reuse into traditional software life cycle processes.

Reuse processes describe how software products are built with assets and how to build and manage these assets. The reuse process framework presented in this standard covers both the life cycle of a software product that is developed from assets and the life cycle of an asset.

IEEE/EIA Std 12207.0-1996 defines software life cycle processes in general. This standard adds to IEEE/EIA Std 12207.0-1996 the life cycle processes and activities that enable the practice of systematic reuse. Thus, the use of this standard requires access to and an understanding of IEEE/EIA Std 12207.0-1996.

For organizations that already employ systematic reuse processes, this standard may be used to determine the conformity of those processes to this standard, and as a basis for improvement of those processes where warranted. When establishing or improving systematic reuse processes, organizations are encouraged to assess the business case. While systematic reuse enables major benefits such as those already described, there are costs, risks, and other considerations that may prevent those benefits from being realized. These include the following:

- *The degree to which reuse benefits are relevant to the organization.* For example, a very small organization, or one that experiences insignificant time and costs to develop and maintain software products, may not be in a position to benefit sufficiently from systematic reuse to justify the required commitments and investments.
- *The availability of suitable tools and assets that are designed for reuse.* The capital costs entailed by new software tools can be significant. The costs of acquiring and/or developing assets may not be justified in relation to the expected benefits.
- *The software maturity of the organization.* While the organization may wish to undertake systematic reuse, its capability maturity may be insufficient to implement the processes in this standard. Or, the organization may lack the means to change its infrastructure to support the processes of systematic reuse while continuing to operate its business as usual. Capability maturity should be objectively assessed in relation to this standard, and missing prerequisites, both as to capability maturity and as to infrastructure, need to be put in place before attempting to undertake systematic reuse.
- *The willingness of the people within the organization to make the necessary changes to the way in which they work.* Many software organizations have cultures that are not conducive to systematic reuse. Changing attitudes and associated non-reuse behaviors can be difficult. Policy changes and capital investments, which require senior management to be firmly committed to the achievement of systematic reuse, may be necessary.

Organizations interested in undertaking systematic reuse are advised to analyze their abilities to adopt this standard. A business case that clearly describes the goals, investments, costs, risks, and benefits, along with a timeline for achieving the transition to systematic reuse, is an excellent way to ensure success.

This standard provides the basis for software practices that enable the incorporation of reuse into the software life cycle processes.

IEEE Std 1517-1999 may be used to

- Acquire, supply, develop, manage, and maintain assets;
- Acquire, supply, develop, operate, and maintain software products that are built in whole or in part with assets;
- Manage and improve the organization’s software life cycle processes with respect to the practice of software reuse;
- Establish software management and engineering environments based on reuse processes;
- Foster improved understanding between customers and vendors and parties involved in the reuse-based life cycle of a software product or system and assets;
- Facilitate the use of assets to develop software products and systems;
- Facilitate the development of assets.

IEEE Std 1517-1999 has been written to work with and integrate into IEEE/EIA Std 12207.0-1996.

Participants

This standard was prepared by the Working Group on Information Technology—Software Life Cycle Processes—Reuse Processes. The work was coordinated with the Reuse Steering Committee (RSC) of the IEEE Computer Society Software Engineering Standards Committee (SESC).

At the time this standard was completed, the Reuse Processes Working Group had the following membership:

Carma McClure, *Chair*

Rose Armstrong
Paul Bassett
Steve Cichinski
Verlynda Dobbs
Patrick Donohoe
Simon Heesh
Kay Hohn
Frank Huy
Rebecca Joos
Tony Kram
Ann Laign
Anil Midha
Nader Nada
Tim Niesen
Juan Roa
Mia Watts
Jim Wilson
Kathyrn P. Yglesias

The following members of the balloting committee voted on this standard:

Jeremy A. Adams
Bakul Banerjee
Edward E. Bartlett
Barbara K. Beauchamp
Leo Beltracchi
Richard E. Biehl
Juris Borzovs
James E. Cardow
Enrico A. Carrara
Lawrence Catchpole
Keith Chan
Keith Chow
Antonio M. Cicu
Paul R. Croll
Geoffrey Darnton
Taz Daughtrey
Bostjan K. Derganc
Verlynda Dobbs
George Duffield
Leo G. Egan
William Eventoff
Richard E. Fairley
John W. Fendrich
Neal Fishman
Jay Forster
John H. Fowler
John Garth Glynn
Julio Gonzalez-Sanz
L. M. Gunther
David A. Gustafson
John Harauz
William Hefley
Debra Herrmann
John W. Horch
George Jackelen
Frank V. Jorgensen
Vladan V. Jovanovic
William S. Junk
Robert J. Kierzyk
Thomas M. Kurihara
John B. Lane
Dieter Look
Stan Magee
Henry A. Malec
Tomoo Matsubara
Jacques Meekel
Denis C. Meredith
Jair Merlo
James Bret Michael
Kathryn J. Moland
James W. Moore
Francisco Navas Plano
Susumu Ohno
Gerald L. Ourada
A. Pedar
Donald J. Pfeiffer
Alex Polack
Jeffrey S. Poulin
Kenneth R. Ptack
Surendra K. Reddy
Donald J. Reifer
Annette D. Reilly
Dennis Rilling
Andrew P. Sage
Helmut Sandmayr
Frederico Sousa Santos
Robert J. Schaaf
Norman Schneidewind
David J. Schultz
Carl A. Singer
Mitchell W. Smith
JagSodhi
Alfred R. Sorkowitz
Mark Spillers
Luca Spotorno
Fred J. Strauss
Toru Takeshita
Richard H. Thayer
Booker Thomas
Leonard L. Tripp
Delores Wallace
Scott A. Whitmire
Paul A. Wolfgang
Paul R. Work
Kathryn P. Yglesias
Natalie C. Yopconka
Janusz Zalewski
Geraldine Zimmerman
Peter F. Zoll

Contents

IEEE Standard for Information Technology—Software Life Cycle Processes—Reuse Processes

1. Overview

This standard provides a common framework for extending the software life cycle processes to include the systematic practice of software reuse. It specifies the processes, activities, and tasks to be applied during each phase of a software life cycle to enable a software product to be constructed from assets.

This standard also specifies the processes, activities, and tasks to enable the identification, construction, maintenance, and management of assets.

1.1 Scope

This standard is intended to describe reuse processes and to relate them to the software life cycle processes that are defined in IEEE/EIA Std 12207.0-1996.[1] (See Annex D of this document.) In this sense, this standard supplements IEEE/EIA Std 12207.0-1996. This standard uses process specifications from IEEE/EIA Std 12207.0-1996 to describe reuse processes.

The scope of this standard will be broader than a single project because reuse processes, such as Domain Engineering, transcend the life cycle of any single software product and apply to a set of related software products.

This standard describes reuse processes at the reference level, as defined by the Basili architecture for process abstraction [B1].[2] In the Basili architecture, the responsibility for each process is assigned to a party referred to as the "agent." From this viewpoint, this standard can be viewed as a list of agents and their minimum responsibilities.

Like IEEE/EIA Std 12207.0-1996, this standard uses the term *party* to indicate the individual or organization responsible for executing a process or activity. The party is often designated with a name derived from the name of the process or activity. For example, the party (individual or group) responsible for Domain Engineering may be termed the *domain engineer.*

[1]Information on references can be found in Clause 2.

[2]The numbers in brackets correspond to those of the bibliography in Annex A.

The limitations of this standard are in

- — Describing a high-level framework for reuse processes, but not the details of how to perform the activities and tasks included in the processes;
- — Describing the responsibilities inherent in the various processes, but not the detailed data or control connections among the processes;
- — Specifying software life cycle reuse processes at the reference level, but not prescribing a specific software life cycle process model or software methodology;
- — Viewing reuse processes as an assignment of continuing responsibilities to agents, but not as a series of steps (procedures) to be performed;
- — Specifying provisions for acquiring assets, but not provisions for integration of commercial off-the-shelf (COTS) assets that are not supplied in source code form.

In this standard, as in IEEE/EIA Std 12207.0-1996, there are a number of lists for tasks; none of these is presumed to be exhaustive—they are intended as examples.

1.2 Purpose

The purposes of this standard are to

- — Establish a framework for practicing reuse within the software life cycle;
- — Define the processes, activities, and tasks that are needed to perform and manage the practice of reuse in a single software project, across multiple software projects, and across an organization;
- — Facilitate communication between software acquirers, suppliers, developers, reuse program administrators, asset managers, and domain engineers by providing a common understanding of reuse and by standardizing reuse terminology;
- — Integrate reuse processes, activities, and tasks with the software life cycle processes described in IEEE/EIA Std 12207.0-1996.

1.3 Field of application

This standard applies to the acquisition of software products, including assets; to the acquisition of software services; and to the supply, development, operation, and maintenance of software products, including assets. Off-the-shelf software that is designed for reuse is a special case of assets that is covered by this standard.

The reuse processes specified by this standard are suitable for use on an organization-wide basis.

This standard is written for acquirers of software products—including assets—and services; and for suppliers, developers, operators, maintainers, managers, quality assurance managers, reuse program administrators, asset managers, domain engineers, and users of software products and assets. Acquirers and suppliers may be either internal or external to the organization.

1.4 Conformance

This standard adds activities and tasks to the existing processes of IEEE/EIA Std 12207.0-1996. In addition, this standard specifies processes in addition to those of IEEE/EIA Std 12207.0-1996. Conformance to this standard requires conformance to IEEE/EIA Std 12207.0-1996. (Note that IEEE/EIA Std 12207.0-1996 uses the term *compliance* rather than *conformance;* for the purpose of this standard, the two terms are considered to be synonymous.)

An organization may claim “selective conformance” to a designated process of this standard by implementing, via both plans and performance, all of the requirements specified as mandatory (by the word *shall*) in the activities and tasks of that process. In the case where a process of this standard is specified in IEEE/EIA Std 12207.0-1996, then conformance requires implementation of the corresponding requirements of IEEE/EIA Std 12207.0-1996 as well as the requirements of this standard.

An organization may claim “full conformance” to this standard if it is in selective conformance with every process specified in this standard.

2. References

This standard shall be used in conjunction with the following publications.

The following standards contain provisions, which through reference in this standard, constitute provisions of this standard. At the time of publication, the standard editions indicated were the most recent.

IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.[3]

IEEE/EIA Std 12207.0-1996, IEEE/EIA Standard for Information Technology—Software life cycle processes.

IEEE/EIA Std 12207.1-1997, IEEE/EIA Guide for Information Technology—Software life cycle processes—Life cycle data.

3. Definitions

For the purposes of this standard, the following terms and definitions apply. IEEE Std 100-1996, IEEE/EIA Std 12207.0-1996, and IEEE Std 610.12-1990 should be referenced for terms not defined in this clause.

3.1 application engineering: The process of constructing or refining application systems by reusing assets.

3.2 assemble: The process of constructing from parts one or more identified pieces of software.

3.3 asset: An item, such as design, specifications, source code, documentation, test suites, manual procedures, etc., that has been designed for use in multiple contexts. *Note:* This term has been deliberately redefined from its current IEEE standard definition to more properly convey its meaning in the software reuse context.

3.4 classification: The manner in which the assets are organized for ease of search and extraction within a reuse library [B4].

3.5 construction: The process of writing, assembling, or generating assets.

3.6 domain: A problem space [B7].

3.7 domain analysis: (A) The analysis of systems within a domain to discover commonalities and differences among them. **(B)** The process by which information used in developing software systems is identified, captured, and organized so that it can be reused to create new systems, within a domain. **(C)** The result of the process in (A) and (B) [B7].

[3]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://www.standards.ieee.org/).

3.8 domain architecture: A generic, organizational structure or design for software systems in a domain. The domain architecture contains the designs that are intended to satisfy requirements specified in the domain model. The domain architecture documents design, whereas the domain model documents requirements [B7]. A domain architecture: 1) can be adapted to create designs for software systems within a domain, and 2) provides a framework for configuring assets within individual software systems. *Note:* The term “architecture” has been deliberately refined from its current IEEE standard definition to more properly convey its meaning in the software reuse context.

3.9 domain definition: The process of determining the scope and boundaries of a domain. The result of the process [B7].

3.10 domain engineer: A party that performs domain engineering activities (including domain analysis, domain design, asset construction, and asset maintenance).

3.11 domain engineering: A reuse-based approach to defining the scope (i.e., domain definition), specifying the structure (i.e., domain architecture), and building the assets (e.g., requirements, designs, software code, documentation) for a class of systems, subsystems, or applications. Domain engineering may include the following activities: domain definition, domain analysis, developing the domain architecture, and domain implementation [B7].

3.12 domain expert: An individual who is intimately familiar with the domain and can provide detailed information to the domain engineers. [B7].

3.13 domain model: A product of domain analysis that provides a representation of the requirements of the domain. The domain model identifies and describes the structure of data, flow of information, functions, constraints, and controls within the domain that are included in software systems in the domain. The domain model describes the commonalities and variabilities among requirements for software systems in the domain [B7].

3.14 product line: A collection of systems that are potentially derivable from a single domain architecture.

3.15 reusability: (A) The degree to which an asset can be used in more than one software system, or in building other assets. **(B)** In a reuse library, the characteristics of an asset that make it easy to use in different contexts, software systems, or in building different assets.

3.16 reuse: The use of an asset in the solution of different problems.

3.17 reuse sponsor: A member of the organization’s management who authorizes, approves, promotes, and obtains the funding and other resources for the Reuse Program.

3.18 system: An integrated composite that consists of one or more of the processes, hardware, software, facilities, and people, that provides a capability to satisfy a stated need or objective [IEEE/EIA Std 12207.0-1996].

3.19 systematic reuse: The practice of reuse according to a well-defined, repeatable process.

3.20 template: An asset with parameters or slots that can be used to construct an instantiated asset. *See also:* **construction.**

3.21 usability: A measure of an executable software unit’s or system’s functionality, ease of use, and efficiency. *See also:* **reusability.** *Note:* This term has been deliberately redefined from its current IEEE standard definition to more properly convey its meaning in the software reuse context.

4. Application of this standard

This clause presents the framework of the reuse life cycle processes that can be employed to

— Develop, operate, and maintain systems and software products with assets;
— Develop and maintain assets.

4.1 Organization of this standard

4.1.1 Reuse processes, activities, and tasks

This standard specifies the software life cycle processes, activities, and tasks that are required to practice reuse. They fall into the following four reuse categories:

— Development, operation, and maintenance of software products with assets;
— Development and maintenance of assets;
— Management of the practice of reuse;
— Management of assets.

This standard uses the software life cycle framework that is described in IEEE/EIA Std 12207.0-1996 as the foundation for describing a software life cycle that includes the practice of reuse. Reuse processes, activities, and tasks are integrated into the IEEE/EIA Std 12207.0-1996 software life cycle framework, as depicted in Figure 1.

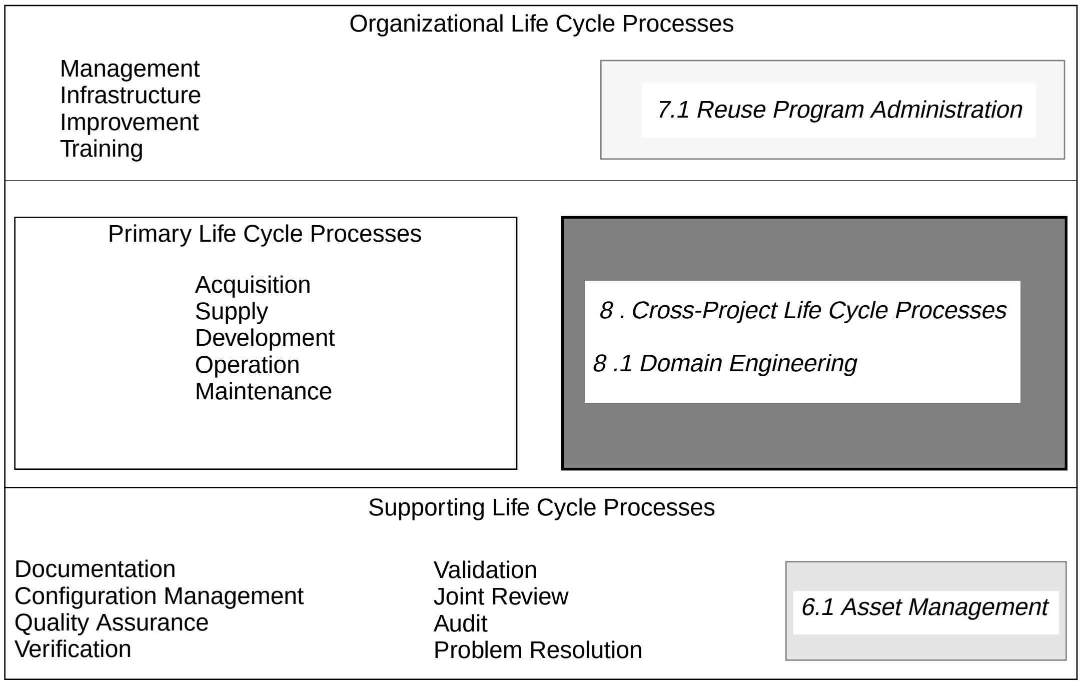

Figure 1—Structure of IEEE Std 1517-1999 within the 12207 framework; and framework for reuse life cycle processes

Each life cycle process discussed in this standard is divided into a set of activities; and the requirements of each activity are specified in a set of tasks. Subclause number *x.x* denotes a process, *x.x.x* denotes an activity, and *x.x.x.x* denotes a task. Correlation of the processes, activities, and tasks in this standard with those in IEEE/EIA Std 12207.0-1996 is based on the name, not the number, associated with a process or activity. Numbers of activities and tasks specified in this standard are not intended to directly map to activities and tasks specified in IEEE/EIA Std 12207.0-1996. When a task defined in this standard adds requirements to an existing task in IEEE/EIA Std 12007.0-1996, the task number of the affected IEEE/EIA Std 12207.0-1996 task is indicted in the text for the corresponding reuse task specified in this standard.

4.1.1.1 Development of software products with assets

Reuse activities and tasks that are related to the development, operation, and maintenance of software products with assets are integrated into the IEEE/EIA Std 12207.0-1996 primary life cycle processes (Clause 5). These primary life cycle processes are employed to acquire, supply, develop, operate, and maintain software products. With the addition of these reuse activities and tasks, the primary life cycle processes described in this standard employ domain models, domain architectures, and assets to develop and maintain software products.

The acquirer, supplier, developer, operator, and maintainer are the parties or agents responsible for performing the primary processes.

4.1.1.2 Development of assets

This standard groups the activities and tasks that fall into the second reuse category (i.e., development and maintenance of assets) into the Domain Engineering Process (Clause 8). This reuse process is specified as a cross-project life cycle process. As the name suggests, this type of process operates across multiple projects. The Domain Engineering Process operates across projects by providing assets that can be used by multiple projects. These assets can be used by the primary life cycle processes to acquire, supply, develop, operate, and maintain software products.

The domain engineer is the party or agent responsible for performing the Domain Engineering Process.

4.1.1.3 Management of the practice of reuse

This standard groups the activities and tasks that fall into the third reuse category (i.e., the management of the practice of reuse) into the Reuse Program Administration Process. This process is specified as an organizational life cycle process. IEEE/EIA Std 12207.0-1996 defines an organizational life cycle process as a process “employed by an organization to establish and implement an underlying structure.” By specifying Reuse Program Administration as an organizational process, reuse can be practiced in a systematic manner within a single project, as well as across an organization. Typically, the Reuse Program Administration Process is employed outside the realm of a specific project or contract in order to manage and coordinate the practice of systematic reuse across multiple projects or contracts in an organization.

This standard adds the Reuse Program Administration Process (Clause 7) to the set of organizational processes defined in IEEE/EIA Std 12207.0-1996. In addition, this standard employs the IEEE/EIA Std 12207.0-1996 organizational processes, such as Management, Infrastructure, Improvement, and Training, to establish an integrated reuse program structure of processes and personnel that can be managed and improved.

The reuse program administrator is the party or agent responsible for performing the Reuse Program Administration Process.

4.1.1.4 Management of assets

This standard groups the activities and tasks that fall into the fourth reuse category (i.e., management of assets) into the Asset Management Process. This process is specified as a supporting process. IEEE/EIA Std 12207.0-1996 defines a supporting process as a process that "supports another process as an integral part with a distinct purpose and contributes to the success and quality of the software project." Asset Management supports other life cycle processes, such as Development, Maintenance, and Domain Engineering, by specifying the activities needed to manage asset certification, classification, storage, retrieval, version control, and change control.

This standard adds the Asset Management Process (Clause 6) to the set of supporting processes defined in IEEE/EIA Std 12207.0-1996. In addition, this standard employs the IEEE/EIA Std 12207.0-1996 supporting processes, such as Documentation, Configuration Management, and Joint Review, to enable the execution of reuse processes such as Domain Engineering and Reuse Program Administration.

The asset manager is the party or agent responsible for performing the Asset Management Process.

5. Integration of reuse into the primary life cycle processes

This clause defines the reuse activities and tasks required to develop, operate, and maintain software products with assets. These reuse activities and tasks are integrated into the following IEEE/EIA Std 12207.0-1996 primary processes as described in this clause:

— Acquisition Process
— Supply Process
— Development Process
— Operation Process
— Maintenance Process

The activities and tasks in each primary process are the responsibility of the party initiating and performing this process. This party assures that the process is in existence and is functional.

5.1 Acquisition Process

The Acquisition Process contains the activities and tasks of the acquirer. The Acquisition Process is employed when a party has a need to acquire a system, software product, or asset. In this situation, the party is referred to as the acquirer.

The purposes of performing the Acquisition Process are as follows:

— Define the need to acquire a system, software product, or asset;
— Prepare and issue a request for proposal (RFP);
— Select a supplier;
— Manage the Acquisition Process.

When employing a reuse-based software life cycle to develop a software product with reusable assets, the acquirer shall perform the Acquisition Process according to the activities and tasks that have been defined in the IEEE/EIA Std 12207.0-1996 Acquisition Process.

The Acquisition Process consists of the following activities:

— Initiation;
— RFP preparation;
— Contract preparation and update;
— Supplier monitoring;
— Acceptance and completion.

In addition, to assure that the Acquisition Process supports the reuse-based life cycle, the acquirer shall perform the following additional reuse-related tasks when performing these IEEE/EIA Std 12207.0-1996 Acquisition Process activities:

— Initiation;
— RFP preparation;
— Acceptance and completion.

5.1.1 Initiation

The following reuse-related tasks are added to this activity:

5.1.1.1 Before acquiring software, the acquirer shall review current, available, and soon-to-be-available assets to determine if what is needed exists or can be assembled to develop the needed software. The acquirer may task the supplier, domain engineer, or asset manager to identify assets for consideration.

5.1.1.2 The acquirer shall require assets to

a) Be reusable;
b) Have interfaces compatible with the domain architecture;
c) Be compatible with the domain model.

5.1.1.3 The acquirer shall consider and document reuse as an option for acquisition as part of the acquirer’s analysis of risk, cost, and benefit criteria. The acquirer shall compare software requirements with asset capabilities and, if proper, evaluate and document costs and benefits of modifying software requirements to enable or increase the reuse of assets. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.1.1.6.)

5.1.1.4 When an off-the-shelf software product or asset is to be acquired, the acquirer should select products or assets that meet the standards and criteria for reusability of the acquirer’s organization. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.1.1.7.)

5.1.1.5 The acquirer shall create and document an acquisition plan, reusing an applicable acquisition plan template, if any exists, to define the resources and procedures to acquire software. The acquirer may use the acquisition plan template included in Annex B of IEEE Std 1062, 1998 Edition [B3]. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.1.1.8.)

5.1.2 RFP preparation

The following reuse-related task is added to this activity:

5.1.2.1 The acquisition documentation shall include any requirements for practicing reuse that the supplier will be required to meet. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.1.2.1.)

5.1.3 Acceptance and completion

The following reuse-related task is added to this activity:

5.1.3.1 If the software product is to be reusable, then acceptance test cases should include reusability tests and domain architecture compatibility tests. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.1.5.1.)

5.2 Supply Process

The Supply Process contains the activities and tasks of the supplier. The Supply Process is employed by a party that needs to prepare a proposal or a contract to provide a system, software product, or asset to an acquirer. In this situation, the supplying party is referred to as the supplier.

The purposes of performing the Supply Process are as follows:

— Prepare a proposal to respond to an RFP from an acquirer;
— Prepare a contract to provide a system, software product, or asset to an acquirer;
— Determine the procedures and resources needed to manage a project to develop and deliver a system, software product, or asset to an acquirer.

When employing a reuse-based software life cycle to develop a software product with assets, the supplier shall perform the Supply Process according to the activities and tasks that have been defined in the IEEE/EIA Std 12207.0-1996 Supply Process. In addition, to assure that the Supply Process supports the reuse-based life cycle, the supplier shall perform the following additional reuse-related tasks when performing this IEEE/EIA Std 12207.0-1996 Supply Process activity:

— Planning.

5.2.1 Planning

The following reuse-related tasks are added to this activity:

5.2.1.1 When defining or selecting a software life cycle model for this supply project, the supplier shall evaluate and document the capabilities of this model to satisfy reuse process requirements. The supplier should explore with the acquirer the possibility of modifying software requirements to enable or increase the reuse of assets and, if proper, evaluate and document the costs and benefits thereof. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.2.4.2.)

5.2.1.2 Once the planning requirements are established, the supplier shall consider and document the option of developing the system, software product, or asset with assets as part of the supplier's analysis of risks and benefits associated with this option. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.2.4.4.)

5.2.1.3 Reuse strategies that the supplier shall consider and document in the project plan include the following:

a) Developing the software product with assets;
b) Developing the software product, or some parts of it, as an asset.

(This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.2.4.5.)

5.2.1.4 The supplier shall gather existing candidate software products and assets that may be used to develop a software product that satisfies the requirements specifications from sources including the following:

a) Domain engineering deliverables;
b) Internal and external suppliers;
c) Organizations' reuse libraries;
d) Other on-going projects in the organization;
e) Off-the-shelf software products and assets.

The supplier shall document this information in the project plans and communicate it using the domain engineering feedback mechanism and the asset management communication mechanism.

5.2.1.5 The supplier should identify and document in the project plans, candidate software products and assets that may be developed within the scope and context of this project. This information should be communicated using the domain engineering feedback mechanism and the asset management communication mechanism.

5.3 Development Process

The Development Process contains the activities and tasks of the developer. In the context of this standard, the process contains the activities for requirements analysis, design, coding, integration, testing, and installation and acceptance related to software products developed with assets. The Development Process based on developing a software product with assets is also referred to as *application engineering.*

When the reuse-based life cycle is deployed to build software products with assets, the assets employed should include the domain architectures and assets that have been built for the domain in which this software product is a member. In addition, the assets used to build this software product may be obtained from other sources including other domain engineering processes, the organization's reuse libraries, other on-going internal projects, and outside suppliers.

The party that performs the Development Process is referred to as the *developer.* The developer shall use the IEEE/EIA Std 12207.0-1996 Supporting Life Cycle Processes and Organizational Processes to establish, support, and manage the Development Process.

The Development Process, based on developing a software product with assets, consists of the following activities:

— Process implementation;
— System requirements analysis;
— System architectural design;
— Software requirements analysis;
— Software architectural design;
— Software detailed design;
— Software coding and testing;
— Software integration;
— Software qualification testing;
— System integration;
— System qualification testing;
— Software installation;
— Software acceptance support.

NOTE—The Development Process may be carried out as an iterative process in which its activities and tasks are performed repeatedly.

The developer shall perform the Development Process according to the activities and tasks that are defined in IEEE/EIA Std 12207.0-1996. In addition, to assure that a reuse-based life cycle is employed to develop the software product, the developer shall perform the additional reuse-related tasks associated with the Development Process activities described in 5.3.1 through 5.3.12.

5.3.1 Process implementation

The following reuse-related tasks are added to this activity:

5.3.1.1 The developer shall evaluate and document the capabilities of the model to satisfy reuse requirements when selecting or defining a life cycle process model appropriate to the scope, magnitude, and complexity of this project. The reuse-related activities and tasks shall be selected and mapped onto the life cycle process model chosen. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.1.1.)

5.3.1.2 The developer shall select and use standards, methods, tools, and computer programming languages that enable, support, and enforce the practice of reuse in the project. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.1.3.)

5.3.1.3 The developer shall create and document a project plan, reusing an applicable project plan template, if any exists, to define the resources and procedures to develop the software. The plans should include specific standards, methods, tools, and programming languages that have been selected to enable the practice of reuse in the project. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.1.4.)

5.3.1.4 The developer shall utilize the following project mechanisms:

a) Feedback mechanism to the domain engineer to communicate the use and impact of software products and assets on this project;
b) Communication mechanism with the asset manager to resolve problems, answer questions, and make recommendations concerning software products and assets that this project encounters;
c) Notification mechanism that makes the developer aware of the prevailing trade laws, the licensing properties of software products and assets, the organization’s restrictions that protect its proprietary interests, and the contract that may restrict or exclude the use of specific software products or assets by this project;
d) Notification mechanism that informs the developer about any changes or problems or other useful information concerning relevant assets.

5.3.2 System requirements analysis

The following reuse-related tasks are added to this activity:

5.3.2.1 The developer shall identify applicable domains and select their corresponding domain models, if they exist.

5.3.2.2 The developer shall select and reuse applicable system requirements specifications, if any exist, before writing new requirements specifications. When writing new requirements specifications, the developer shall use the language and concepts of the domain models for the domain to which this software system belongs.

5.3.2.3 The systems requirements specification shall describe system reusability requirements, including the domain architecture interface requirements. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.2.1.)

5.3.2.4 The system requirements specification shall be evaluated according to reusability criteria. The results of the evaluation shall be documented. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.2.1.)

NOTE—Reusability criteria include, but are not limited to
a) Usability and reusability of the system requirements;
b) Identification of customized requirements that can be replaced by reused requirements if the acquisition needs are adjusted;
c) Reuse potential of the requirements to be used in multiple contexts.

5.3.2.5 The developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report reuse information about requirements specifications.

NOTE—Reuse information includes, but is not limited to
a) Missing assets for the domain;
b) Nominally reusable assets for the domain rejected for this project and the rejection reasons;
c) Suggested modifications to assets;
d) List of assets reused by this project;
e) List of new candidate assets suggested by this project.

5.3.3 System architectural design

The following reuse-related tasks are added to this activity:

5.3.3.1 The developer shall select applicable domain architectures consistent with the domain models, if any exist.

5.3.3.2 The system architecture shall be derived from the selected domain architectures. The allocation of the system requirements shall follow the domain models that correspond to the domain architectures.

NOTE—The terms *software item, software component,* and *software unit* have the same meaning in this document as in IEEE/EIA Std 12207.0-1996. The term *asset,* as defined in this document, may refer to a software item, software component, and software unit, as well as other parts (including software sub-units) or products of the software life cycle that have been designed for use in multiple contexts.

5.3.3.3 The system architecture shall be evaluated according to reusability criteria. The results of the evaluation shall be documented. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.3.2.)

NOTE—Reusability criteria include, but are not limited to
a) Usability and reusability of the system architecture and its parts;
b) Reuse potential of the system architecture or its parts to be used in multiple contexts;
c) Compatibility of the system architecture with the domain models and domain architectures.

5.3.3.4 The system developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report reusability problems and recommendations for the system architectural design.

5.3.4 Software requirements analysis

The following reuse-related tasks are added to this activity:

5.3.4.1 The developer shall include software reusability requirements for assets in the quality characteristics specifications to assure the quality of the assets selected for reuse in the development of the software product. The following criteria may be used to evaluate the quality of reusability:

a) The reliability experience of assets reused in similar projects;
b) The results of inspecting the assets for defects;
c) The results of testing the assets against the system and software requirements.

(This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.4.1.)

5.3.4.2 The developer shall select and reuse applicable software requirements, if they exist and remain current, before establishing new ones. The developer shall define and document new software requirements to include, but not be limited to, reusability characteristics. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.4.1.)

5.3.4.3 The developer shall write new requirements using an applicable requirements template, if any exists, and using the language and concepts from the domain models of the domains to which this software product belongs. The developer shall document software requirements. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.4.1.)

5.3.4.4 The developer shall evaluate software requirements according to reusability criteria. The results of the evaluation shall be documented. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.4.2.)

NOTE—Reusability criteria include, but are not limited to
a) Compatibility with the domain models and domain architectures of the domains to which this software product belongs;
b) Compliance to the organization's reuse standards;
c) Usability and reusability of the software requirements;
d) Identification of customized software requirements that can be replaced by reused software requirements if the acquisition needs are adjusted.

5.3.4.5 The developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report reuse information about software requirements.

5.3.5 Software architectural design

The following reuse-related tasks are added to this activity:

5.3.5.1 The developer shall derive a software architecture that is based on selected, applicable domain architectures. If no such domain architectures exist, the developer may define an architecture for this software product that is consistent with the domain models and that describes the structure of the software product and the software components that comprise this structure. The developer shall allocate software requirements to this architecture following the domain models that correspond to the selected domain architecture, if it exists. The software architecture shall be documented reusing applicable documentation assets, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.5.1.)

5.3.5.2 Interfaces external to the software item and between the software components within the software item shall comply with the domain architecture interfaces and shall be derived from applicable assets, such as interface assets, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.5.2.)

5.3.5.3 Database designs shall be derived from selected domain models and database designs, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.5.3.)

5.3.5.4 User documentation shall be created reusing applicable documentation assets, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.5.4.)

5.3.5.5 Preliminary test requirements shall be created reusing applicable test requirements, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.5.5.)

5.3.5.6 The developer shall evaluate the software architecture, the interface, and database designs according to reusability criteria. The results of the evaluation shall be documented. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.5.6.)

NOTE—Reusability criteria include, but are not limited to
a) Compliance with the organization's reuse standards;
b) Usability and reusability of the software architecture and database designs;
c) Reuse potential of the software architecture and its components to be used in multiple contexts.

5.3.5.7 The developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report reuse information about the software architectural design.

NOTE—Reuse information about the software design includes, but is not limited to
a) Problems encountered with the domain architectures or the domain models;
b) The reasons for being unable to reuse the domain architectures or domain models;
c) Suggested modifications to the domain architectures or domain models;
d) List of domain architectures and domain models used by this project;
e) List of new candidate reusable assets suggested by this project.

5.3.6 Software detailed design

For each software item, the following reuse-related tasks are added to this activity:

5.3.6.1 The developer shall select and reuse an applicable detailed design for each software component, if any exists. If none exists, the developer shall develop a new detailed design for the software component. When writing new detailed designs, the developer shall use the language and concepts from the selected domain models. The developer shall use data structures and naming conventions from the domain models when describing detailed designs. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.6.1.)

5.3.6.2 The developer shall develop and document interfaces external to the software item, between software components, and between software units that are compliant with domain interface standards. The developer shall reuse applicable assets, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.6.2.)

5.3.6.3 The developer shall select and reuse applicable detailed designs and documentation for the database, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.6.3.)

5.3.6.4 The developer shall reuse applicable test assets to create test requirements, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.6.5.)

5.3.6.5 The developer shall evaluate the software detailed design and test requirements according to reusability criteria. The results of the evaluation shall be documented. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.6.7.)

NOTE—Reusability criteria include, but are not limited to

a) Consistency with the software architecture design and compliance to the domain architectures and domain models;
b) Compliance to the organization's reuse standards;
c) Usability and reusability of the software design and test requirements;
d) Identification of customized designs and test requirements that may be replaced by reusable designs and test requirements if the software requirements are adjusted;
e) Reuse potential of the software item designs, software component designs, software unit designs, and test requirements to be used in multiple contexts.

5.3.6.6 The developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report the reuse information about the software detailed design.

5.3.7 Software coding and testing

For each software item, the following reuse-related tasks are added to this activity:

5.3.7.1 The developer shall select and reuse applicable software units, databases, test procedures, and test data, if any exist, in order to produce and test each software unit. When producing new code for software units or a new database, the developer shall use the language and concepts from the domain models and domain architectures. Each software unit and database developed shall be documented reusing applicable documentation assets, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.7.1.)

5.3.7.2 The developer shall evaluate the software code and tests according to reusability criteria. The results of the evaluation shall be documented. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.7.5.)

NOTE—Reusability criteria include, but are not limited to

a) Compliance with the organization's reuse standards;
b) Usability and reusability of the code and test assets used to produce the tested code;
c) Reuse potential of the code and test assets to be used in multiple contexts.

5.3.7.3 The developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report reuse information about the new software code, test procedures, and test data.

5.3.8 Software integration

For each software item, the following reuse-related tasks are added to this activity:

5.3.8.1 The developer shall create and document an integration plan, reusing an applicable integration plan template, if any exists, to define the resources and procedures for integrating software units and components into the software item. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.8.1.)

5.3.8.2 The developer shall document the software item and the test results reusing applicable documentation assets, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.8.2.)

5.3.8.3 The developer shall develop a set of qualification tests, test cases, and test procedures, using applicable test assets, if any exist, to test each qualification requirement of the software item. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.8.4.)

5.3.8.4 The developer shall evaluate the integration plan, software design, code, tests, and user documentation according to reusability criteria. The results of the evaluation shall be documented. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.8.5.)

NOTE—Reusability criteria include, but are not limited to

a) Compliance to the organization's reuse standards;
b) Usability and reusability of the software item, test cases, and test procedures;
c) Reuse potential of the software design, code, test cases, test procedures, and user documentation to be used in multiple contexts.

5.3.8.5 The developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report reuse information about software integration.

5.3.9 Software qualification testing

For each software item, the following reuse-related tasks are added to this activity:

5.3.9.1 The developer shall document the results of software qualification testing reusing applicable documentation assets, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-199, task 5.3.9.1.)

5.3.9.2 The developer shall evaluate the software design, code, test assets, and user documentation according to reusability criteria. The results of the evaluation shall be documented. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.9.3.)

NOTE—Reusability criteria include, but are not limited to

a) Compliance to the organization's reuse standards;
b) Usability and reusability of the software design, code, test assets, and user documentation;
c) Reuse potential of the software design, code, test assets, and user documentation to be used in multiple contexts.

5.3.9.3 The developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report the reuse information about the software qualification testing.

5.3.10 System integration

The following reuse-related tasks are added to this activity:

5.3.10.1 For each qualification requirement of the system, a set of tests, test cases, and test procedures shall be developed, reusing applicable test assets, if any exist. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.10.1.)

5.3.10.2 The integrated system shall be evaluated according to reusability criteria. The results of the evaluation shall be documented. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.10.3.)

NOTE—Reusability criteria include, but are not limited to

a) Compliance to the organization's reuse standards;
b) Usability and reusability of the system and qualification tests;
c) Reuse potential of the system, its parts, and its qualification tests to be used in multiple contexts.

5.3.10.3 The developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report reuse information about the integrated system, set of tests, test cases, and test procedures.

5.3.11 System qualification testing

The following reuse-related tasks are added to this activity:

5.3.11.1 The system shall be evaluated according to reusability criteria. The results of the evaluation shall be documented. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.11.2.)

NOTE—Reusability criteria include, but are not limited to
a) Usability and reusability of the software portion of the system;
b) Reuse potential of the software parts of the system to be used in multiple contexts.

5.3.11.2 The developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report reuse information about system qualification testing.

5.3.12 Software installation

The following reuse-related tasks are added to this activity:

5.3.12.1 The developer shall create and document an installation plan, reusing an applicable installation plan template, if any exists, to define the resources and procedures to install the software product on its target environment(s). (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.3.12.1.)

5.3.12.2 The installation plan and installation plan template shall be evaluated according to reusability criteria. The results of the evaluation shall be documented.

NOTE—Reusability criteria include, but are not limited to
a) Usability and reusability of the installation plan and installation plan template;
b) Reuse potential of the installation plan to be used in multiple contexts.

5.3.12.3 The developer shall use the domain engineering feedback mechanism and the asset management communication mechanism to report reuse information about software installation.

5.4 Operation Process

The Operation Process contains the activities and tasks of the operator. The Operation Process is employed by an operator to operate a system. The process covers

— Operation of the system;
— Providing operation support to the users of the system.

When a reuse-based software life cycle is employed to develop the software product, the operator shall perform the Operation Process according to the activities and tasks that have been defined in the IEEE/EIA Std 12207.0-1996 Operation Process. In addition, to assure that the Operation Process supports reuse, the operator shall perform the following additional reuse-related tasks when performing this IEEE/EIA Std 12207.0-1996 Operation Process activity:

— Process implementation.

5.4.1 Process implementation

The following reuse-related tasks are added to this activity:

5.4.1.1 The operator shall create and document an operations plan, reusing an applicable operations plan template, if any exists, to define the resources and procedures for operation of the system. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.4.1.1.)

5.4.1.2 The operations plan and operations plan template shall be evaluated according to reusability criteria. The results of the evaluation shall be documented.

NOTE—Reusability criteria include, but are not limited to
a) Usability and reusability of the operation plan and operation plan template;
b) Reuse potential of the operation plan to be used in multiple contexts.

5.4.1.3 The operator shall use the domain engineering feedback mechanism and the asset management communication mechanism to report reuse information about software operations.

5.5 Maintenance Process

The Maintenance Process contains the activities and tasks of the maintainer. The Maintenance Process is performed to modify an existing software product due to a need to correct a deficiency in the product or to adapt the product to meet a new or revised requirement. The party that performs software maintenance is referred to as the maintainer.

The purposes of the Maintenance Process are to

— Modify an existing software product;
— Migrate an existing software product;
— Retire an existing software product.

When a reuse-based software life cycle is employed to develop the software product and/or a reuse-based approach is to be employed to maintain the software product, the maintainer shall perform the Maintenance Process according to the activities and tasks that have been defined in the IEEE/EIA Std 12207.0-1996 Maintenance Process. In addition, to assure that the Maintenance Process supports reuse, the maintainer shall perform the following additional reuse-related tasks when performing these IEEE/EIA Std 12207.0-1996 Maintenance Process activities:

— Process implementation;
— Problem and modification analysis;
— Modification implementation;
— Migration;
— Software retirement.

5.5.1 Process implementation

The following reuse-related tasks are added to this activity:

5.5.1.1 The maintainer shall develop a maintenance plan reusing an applicable maintenance plan template, if any exists. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.5.1.1.)

5.5.1.2 The maintainer shall establish procedures for receiving, recording, resolving, and tracking problems, and providing feedback to the asset manager whenever problems are encountered in or change requests are made for assets that were reused in the development of the software product.

5.5.1.3 Whenever problems with or change requests for assets are encountered, they shall be recorded in accordance with the IEEE/EIA Std 12207.0-1996 Problem Resolution Process so that all other software products that reused these assets can be examined to determine the impact that these problems or change requests may have on these software products.

5.5.2 Problem and modification analysis

The following reuse-related tasks are added to this activity:

5.5.2.1 The maintainer shall consider the reuse of assets to implement the maintenance modifications. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.5.2.3.)

5.5.2.2 The maintainer shall analyze the problem report or modification request for its impact on any assets reused in the development of the system. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.5.2.1.)

5.5.2.3 The maintainer shall provide the problem/modification request and the analysis results that concern assets to the asset manager.

5.5.3 Modification implementation

The following reuse-related task is added to this activity:

5.5.3.1 The maintainer shall receive, review, and accept any modified assets from the asset manager and integrate them into the modification implementation of the software product.

5.5.4 Migration

The following reuse-related tasks are added to this activity:

5.5.4.1 When a system that has been developed from assets is to be migrated to a new environment, migration planning activities shall include notification of the asset manager.

5.5.4.2 After a system that has been constructed from assets has been migrated to a new environment, notifications shall be sent to the asset manager.

5.5.4.3 When the migrated system has been constructed from assets, the results of a post-operation review that assesses the impact of changing to the new environment shall be sent to the asset manager and the domain engineer. (This task defines reuse-related requirements in addition to those requirements specified in IEEE/EIA Std 12207.0-1996, task 5.5.5.6.)

5.5.5 Software retirement

The following reuse-related tasks are added to this activity:

5.5.5.1 When a system that has been constructed from assets is to be retired, retirement planning activities shall include notifying the asset manager.

5.5.5.2 When a system constructed from assets is retired, notifications shall be sent to the asset manager.

6. Reuse supporting process

This clause defines the following supporting process related to the management of assets:

— Asset Management Process.

The activities and tasks in this supporting process are the responsibility of the party initiating and performing the Asset Management Process. This party assures that the process is in existence and is functional.

6.1 Asset Management Process

Regardless of their overall quality and potential for reuse, assets have little value to an organization unless potential reusers know of their existence and can easily locate and understand them.

This process contains the activities and tasks of the asset manager. Asset Management is the process of applying administrative and technical procedures throughout the life of an asset to identify, define, certify, classify, and baseline the asset; track modifications, migrations, and versions of the asset; record and report the status of the asset; and establish and control storage and handling of the asset, delivery of the asset to its reusers, and retirement of the asset.

This process consists of the following activities:

— Process implementation;
— Asset storage and retrieval definition;
— Asset management and control.

6.1.1 Process implementation

This activity consists of the following tasks:

6.1.1.1 The asset manager shall create and document an asset management plan, reusing an applicable asset management plan template, if any exists, to define the resources and procedures for managing assets. The plan should include the following:

a) Defining the requirements for an asset storage and retrieval mechanism;
b) Defining the asset storage and retrieval mechanism;
c) Establishing the asset storage and retrieval mechanism as an integral part of the software life cycle;
d) Naming the organization(s) responsible for managing and maintaining the asset storage and retrieval mechanism;
e) Defining asset acceptance, certification, and retirement procedures;
f) Defining the relationship of the asset manager to other parties such as developers, maintainers, and domain engineers;
g) Promoting the use of the asset storage and retrieval mechanism;
h) Defining an asset management communication mechanism;
i) Defining an asset classification scheme.

6.1.1.2 The asset manager shall

a) Document this process in accordance with the IEEE/EIA Std 12207.0-1996 Documentation Process;
b) Perform configuration management of assets in accordance with the IEEE/EIA Std 12207.0-1996 Configuration Management Process;

c) Document and resolve problems and nonconformances found in the assets and the Asset Management Process in accordance with the IEEE/EIA Std 12207.0-1996 Problem Resolution Process;
d) Conduct reviews of assets in accordance with the IEEE/EIA Std 12207.0-1996 Joint Review Process.

6.1.1.3 The asset management plan shall be reviewed in accordance with the IEEE/EIA Std 12207.0-1996 Joint Review Process. Domain engineers and reuse program administrators shall be included in the review.

6.1.2 Asset storage and retrieval definition

An asset storage and retrieval mechanism enables reusers to easily and quickly find and understand the assets.

This activity consists of the following tasks:

6.1.2.1 The asset manager shall implement and maintain an asset storage and retrieval mechanism.

6.1.2.2 The asset manager should develop, document, and maintain a classification scheme to be used in classifying the assets.

6.1.2.3 The asset manager shall conduct joint review(s) of the asset storage and retrieval mechanism in accordance with the IEEE/EIA Std 12207.0-1996 Joint Review Process. Reuse program administrators and domain engineers shall be included in the review(s).

6.1.3 Asset management and control

For each asset, this activity consists of the following tasks:

6.1.3.1 For each asset submitted to the asset manager, the asset shall be evaluated based on the asset acceptance and certification criteria.

6.1.3.2 For each asset accepted, it shall be made available for reuse through the asset storage and retrieval mechanism.

6.1.3.3 The asset shall be classified in accordance with the reuse classification scheme, if any exists.

6.1.3.4 The asset manager shall perform configuration management for the asset using the IEEE/EIA Std 12207.0-1996 Configuration Management Process.

6.1.3.5 The asset manager shall keep track of each reuse of the asset and report to the domain engineer information about actual reuses of the asset. Asset reuse information should include the reuser's name, project name, original developer or owner of the asset, cost of reusing the asset, and savings and benefits derived from reusing the asset.

6.1.3.6 The asset manager shall forward asset modification requests and problem reports received from asset reusers to the domain engineer for review and correction/modification plans and actions. Actions planned and taken to meet requests or to correct problems shall be reported to the asset manager making the request or filing the problem report.

6.1.3.7 The asset manager shall monitor and record these asset requests/reports and the subsequent actions taken. Whenever problems with an asset are encountered, they should be recorded and entered into the Problem Resolution Process, as specified in IEEE/EIA Std 12207.0-1996.

6.1.3.8 The asset manager shall notify all asset reusers, and the domain engineer of the problems detected in the asset, modifications made to the asset, new versions of the asset, and deletion of the asset from the asset storage and retrieval mechanism.

6.1.3.9 The asset manager shall retire assets from the asset storage and retrieval mechanism according to the asset retirement procedures and criteria.

7. Reuse organizational life cycle process

This clause defines the following organizational life cycle process related to the practice of reuse:

— Reuse Program Administration Process.

The activities and tasks in this process are the responsibility of the party initiating and performing the Reuse Program Administration Process. This party assures that the process is in existence and is functional.

The party employing and performing this organizational process shall establish, manage, and support the practice of systematic reuse in the organization in accordance with the IEEE/EIA Std 12207.0-1996 supporting life cycle processes and organizational life cycle processes.

7.1 Reuse Program Administration Process

Succeeding with the implementation of systematic reuse at the organization level requires careful planning and proper management. Because business, management, and people challenges often outweigh the technical challenges of implementing reuse, management leadership, commitment, and support, as well as a reuse-positive software culture should be emphasized in a reuse program. All individuals within the scope of the reuse program are expected to cooperate with one another in the establishment of reuse processes and to share reuse expertise and assets with one another.

The Reuse Program Administration Process contains the activities and tasks of the reuse program administrator. This process is used to plan, establish, manage, control, and monitor an organization’s reuse program.

This process consists of the following activities:

— Initiation;
— Domain identification;
— Reuse assessment;
— Planning;
— Execution and control;
— Review and evaluation.

7.1.1 Initiation

This activity consists of the following tasks:

7.1.1.1 The reuse program for an organization shall be initiated by establishing the organization’s reuse strategy that includes its reuse goals, purposes, objectives, and scope. Elements of the reuse program should address the following:

a) Reuse sponsor;
b) Reuse infrastructure (including hardware, software, tools, techniques, standards, metrics, and facilities for practicing reuse);

c) Reuse funding and other resources;
d) Reuse program support function;
e) Reuse communication, feedback, and notification mechanisms.

NOTE—The reuse program administrator defines the following mechanisms:
a) Feedback mechanism from each software development project to the domain engineer and the asset manager to communicate the use and impact of software products and assets on each project;
b) Communication mechanism between the software developer, operator, maintainer, domain engineer, asset manager, and the reuse program administrator to resolve problems, answer questions, and make recommendations concerning software products and assets that each project encounters;
c) Notification mechanism that makes the developer, maintainer, asset manager, and domain engineer aware of the prevailing trade laws, the licensing properties of software products and assets, the organization's restrictions that protect its proprietary interests, and the contract that may restrict or exclude the use of specific software products or assets by each software development, maintenance, or domain engineering project;
d) Mechanism for the domain engineer to obtain the participation and information needed from the appropriate sources to complete domain engineering activities.

7.1.1.2 A reuse sponsor should be named.

7.1.1.3 Reuse program participants shall be identified and their roles shall be assigned.

7.1.1.4 A reuse steering function shall be established to assume the authority and responsibility for the organization's reuse program. Its functions should include the following:

a) Investigation of the practice of reuse in the organization;
b) Identification of the areas in the organization where there are potential reuse opportunities;
c) Assignment of the responsibilities for reuse in the organization;
d) Redefinition of the organization's incentives, disincentives, and culture to support and encourage reuse.

NOTE—Members of the reuse steering function include the reuse sponsor, software development manager, operations manager, software maintenance manager, and a reuse expert.

7.1.1.5 A reuse program support function shall be established. The responsibilities of the reuse program support function should include the following:

a) Participating in the creation and implementation of a reuse program implementation plan;
b) Identifying, documenting, and conveying to all reuse program participants the reuse strategy;
c) Promoting the practice of reuse to encourage a reuse-positive software culture;
d) Seeking out opportunities to practice reuse in current and future software projects;
e) Establishing and maintaining the reuse infrastructure;
f) Providing reuse consulting support to software projects that practice reuse.

7.1.2 Domain identification

A domain characterizes a set of systems in terms of common properties that can be organized into a collection of reusable assets that may be used to construct systems in the domain.

This activity consists of the following tasks:

7.1.2.1 The reuse program administrator, aided by the appropriate manager, domain engineers, users, and software developers, shall identify and document the domains in which to investigate reuse opportunities or in which the organization intends to practice reuse.

7.1.2.2 The reuse program administrator, aided by the appropriate managers, domain engineers, users, and software developers, shall evaluate the domains to assure that they accurately reflect the organization's reuse strategy. The results of the evaluation shall be documented.

7.1.2.3 The reuse program administrator shall conduct joint reviews in accordance with the IEEE/EIA Std 12207.0-1996 Joint Review Process. Software developers, domain engineers, and users shall be included in the reviews.

7.1.2.4 As more information about the organization's domains and plans for future software products becomes available or when the domains are analyzed, the domains may be refined and rescoped by the reuse program administrator.

7.1.3 Reuse assessment

The reuse assessment provides the baseline against which the practice of reuse in the organization can be measured. Without this reuse assessment, the benefits which result from the practice of reuse in the organization are difficult to measure.

The purposes of this activity are to

- — Gain an understanding of the reuse maturity of the organization;
- — Assess the reuse potential of the target domains of the organization;
- — Make recommendations on how to proceed with the practice of reuse in the organization;
- — Motivate and direct incremental improvements in the many areas of the organization's reuse program, including reuse training and infrastructure.

This activity consists of the following tasks:

7.1.3.1 The reuse program administrator shall assess the organization's systematic reuse capability. The results of the assessment shall be documented and provided to the reuse steering function.

7.1.3.2 The reuse program administrator shall assess each domain being considered for reuse to determine the potential for reuse success in the domain. The results of the assessment shall be documented and provided to the reuse steering function.

7.1.3.3 The reuse program administrator shall make recommendations for refining the organization's reuse strategy and reuse program implementation plan based on the results of the reuse assessments. The recommendations shall be documented and provided to the reuse steering function.

7.1.3.4 The reuse program administrator, in conjunction with the appropriate acquirers, suppliers, developers, operators, maintainers, asset managers, and domain engineers, shall use the IEEE/EIA Std 12207.0-1996 Improvement Process to incrementally improve the skills, technology, reuse processes, organizational structure, and metrics that together comprise the reuse infrastructure.

7.1.4 Planning

This activity consists of the following tasks:

7.1.4.1 A reuse program implementation plan shall be created, documented, and maintained, reusing an applicable reuse program plan template, if any exists, to define the resources and procedures for implementing a reuse program. The plan shall describe the following:

a) The reuse program activities;
b) Procedures and schedules for performing these activities;

c) The parties responsible for performing these activities;
d) The relationships with other parties, such as software developers or domain engineers;
e) The resources needed for the reuse program;
f) All other items listed in 5.2 of IEEE/EIA Std 12207.1-1997.

7.1.4.2 The plan shall be reviewed and evaluated considering the following criteria:

a) Completeness;
b) Ability to realize the organization's reuse strategy;
c) Feasibility of implementing the plan.

The results of the evaluation shall be documented. Those evaluating the plan should include members of the reuse steering function.

7.1.4.3 Approval and support for the reuse program implementation plan shall be obtained from the reuse steering function, and the appropriate managers.

7.1.4.4 The reuse program administrator shall conduct joint review(s) in accordance with the IEEE/EIA Std 12207.0-1996 Joint Review Process. Members of the reuse steering function and the appropriate managers shall be included in the reviews.

7.1.5 Execution and control

This activity consists of the following tasks:

7.1.5.1 Activities in the reuse program implementation plan shall be executed in accordance with the plan.

7.1.5.2 The reuse program administrator shall monitor the progress of the reuse program against the organization's reuse strategy, and make and document any necessary adjustments to the plan to realize the strategy.

7.1.5.3 Problems and nonconformances that occur during the execution of the reuse program implementation plan shall be recorded and entered into the Problem Resolution Process, as specified in IEEE/EIA Std 12207.0-1996.

7.1.5.4 The reuse program administrator shall periodically reaffirm management sponsorship, support, and commitment to the reuse program.

7.1.6 Review and evaluation

This activity consists of the following tasks:

7.1.6.1 The reuse program administrator shall periodically assess the reuse program for achievement of the organization's reuse strategy, and the continued suitability and effectiveness of the reuse program.

7.1.6.2 The reuse program administrator shall provide assessment results and lessons learned to the reuse steering function, and to the appropriate managers.

7.1.6.3 The reuse program administrator shall recommend and make changes to the reuse program, expand the reuse program, and improve the reuse program in accordance with the IEEE/EIA Std 12207.0-1996 Improvement Process.

8. Reuse cross-project life cycle process

This clause defines the following cross-project process related to the practice of reuse:

— Domain Engineering Process.

The activities and tasks in this cross-project process are the responsibility of the party initiating and performing the Domain Engineering Process. This party assures that the process is in existence and is functional.

The party employing and performing this cross-project process manages it in accordance with the IEEE/EIA Std 12207.0-1996 Management Process, Improvement Process, and Training Process; and establishes an infrastructure that follows the IEEE/EIA Std 12207.0-1996 Infrastructure Process.

8.1 Domain Engineering Process

The Domain Engineering Process contains the activities and tasks of the domain engineer. The process covers the development and maintenance of the domain models, domain architecture, and other assets for this domain.

This process consists of the following activities:

— Process implementation;
— Domain analysis;
— Domain design;
— Asset provision;
— Asset maintenance.

NOTE—These activities and tasks may overlap or interact and may be performed iteratively or recursively. Also, the Domain Engineering Process may overlap with development and maintenance processes that use assets produced by the Domain Engineering Process.

8.1.1 Process implementation

This activity consists of the following tasks:

8.1.1.1 The domain engineer shall create and document a domain engineering plan, reusing an applicable domain engineering plan template, if any exists, to define the resources and procedures for performing domain engineering. The plan should include standards, methods, tools, activities, assignments, and responsibilities for performing domain engineering. To create the domain engineering plan, the domain engineer should consult the literature and/or data resources about the domain and should consult with domain experts, developers, and users of software products within the domain. The domain engineering plan shall be executed.

8.1.1.2 The domain engineer shall select the representation forms to be used for the domain models and domain architectures, in accordance with the organization’s reuse standards, and by consulting domain experts, developers, and users of software products within the domain.

8.1.1.3 The domain engineer shall

a) Document this process in accordance with the IEEE/EIA Std 12207.0-1996 Documentation Process;

b) Perform configuration management of the domain engineering outputs in accordance the IEEE/EIA Std 12207.0-1996 Configuration Management Process;

c) Document and resolve problems and nonconformance found in the assets and Domain Engineering Process in accordance with the IEEE/EIA Std 12207.0-1996 Problem Resolution Process;

d) Conduct joint reviews in accordance with the IEEE/EIA Std 12207.0-1996 Joint Review Process and include in the review experts of the domain, and software developers and users of software products within the domain;

e) Establish procedures for receiving, resolving, and providing feedback to the asset manager whenever problems or change requests occur for assets developed by the domain engineer.

8.1.2 Domain analysis

Domain analysis is the activity that discovers and formally describes the commonalities and variabilities within a domain. The domain engineer captures this information in a set of domain models.

This activity consists of the following tasks:

8.1.2.1 The domain engineer shall define the boundaries of the domain and the relationships between this domain and other domains.

8.1.2.2 The domain engineer shall identify the current and anticipated needs of developers of software products within this domain.

8.1.2.3 The domain engineer shall build the domain models using the representation forms selected in the process implementation activity for this process.

8.1.2.4 The domain engineer shall construct a vocabulary that provides the terminology to describe the important domain concepts and the relationships among similar or common assets of the domain.

8.1.2.5 The domain engineer shall classify and document the domain models.

8.1.2.6 The domain engineer shall evaluate the domain models and domain vocabulary in accordance with the provisions of the modeling technique selected and in accordance with the organization’s asset acceptance and certification procedures. The results of the evaluation shall be documented.

8.1.2.7 The domain engineer shall conduct domain analysis joint review(s) in accordance with the IEEE/EIA Std 12207.0-1996 Joint Review Process. Software developers, asset managers, domain experts, and users shall be included in the reviews.

8.1.2.8 The domain engineer shall submit domain models to the asset manager.

8.1.3 Domain design

The domain design activity defines the domain architecture and specifies the assets that can be used to build software products. The domain architecture is a high-level design in which asset interfaces are formally specified. The domain architecture serves as the framework for reusing assets to construct software products.

This activity consists of the following tasks:

8.1.3.1 The domain engineer shall create and document the domain architecture, consistent with the domain model and following the organization’s standards.

8.1.3.2 The domain architecture shall be evaluated in accordance with the provisions of the architecture design technique selected and the organization’s asset acceptance and certification procedures. The results of the evaluation shall be documented.

8.1.3.3 For each entity selected to be designed for reuse, the domain engineer shall develop and document an asset specification.

8.1.3.4 For each asset specified, the specification shall be evaluated in accordance with the provisions of IEEE/EIA Std 12207.0-1996, subclause 5.3.6.7, and in accordance with the organization's asset acceptance and certification procedures. The results of the evaluation shall be documented.

8.1.3.5 The domain engineer shall conduct domain design joint review(s) in accordance with the IEEE/EIA Std 12207.0-1996 Joint Review Process. Software developers, domain experts, and asset managers shall be included in the reviews.

8.1.3.6 The domain engineer shall submit the domain architecture to the asset manager.

8.1.4 Asset provision

The asset provision activity develops or acquires assets that can be used to assemble software products. For each asset developed or acquired, this activity consists of the following tasks:

8.1.4.1 The domain engineer shall develop the asset, thus

a) Executing the Acquisition Process (see 5.1) to cause a contract for the asset to be put in place if the asset is to be acquired; or
b) Executing the Development Process (see 5.3) if the asset is to be developed internally.

8.1.4.2 The asset shall be documented and classified.

8.1.4.3 The domain engineer shall evaluate the asset in accordance with the organization's asset acceptance and certification procedures. The results of the evaluation shall be documented.

8.1.4.4 The domain engineer shall conduct asset joint review(s) in accordance with the IEEE/EIA Std 12207.0-1996 Joint Review Process. Software developers and asset managers shall be included in the reviews.

8.1.4.5 The domain engineer shall submit the asset to the asset manager.

8.1.5 Asset maintenance

The asset maintenance activity contains the tasks for modifying assets, including domain models and domain architectures. An asset undergoes modification to correct a deficiency in the asset or to adapt the asset to meet a new or revised requirement.

The domain engineer shall modify the asset, executing

— The Maintenance Process (see 5.5).

In addition, the following reuse-related tasks are added to this Maintenance Process when it is applied to maintain an asset:

8.1.5.1 When analyzing requests for asset modification and choosing implementation options, the domain engineer shall consider

a) Conformance with the domain models and the domain architecture;
b) Impact on the systems and software products that use the asset;
c) Impact on future users of the asset;
d) Impact on the reusability of the asset.

8.1.5.2 The domain engineer shall obtain approval for the selected implementation option, schedule, and plans to modify the asset.

8.1.5.3 The domain engineer shall notify the asset manager who sent the asset modification request about whether the asset modification was approved and the plans and schedule for those approved modifications. When a modification request is not approved, it shall be recorded and entered into the Problem Resolution Process, as specified in IEEE/EIA Std 12207.0-1996.

8.1.5.4 After approval is obtained, the domain engineer shall enter the Domain Engineering Process (Clause 8) to implement the modifications to an asset.

8.1.5.5 The domain engineer shall send the completed modified asset along with any usage instructions and test assets to the asset manager who sent the asset modification request.

Annex A

(informative)

Bibliography

[B1] Basili, V. R. et al., “A reference architecture for the component factory,” *ACM Transactions on Software Engineering and Methodology,* vol. 1, no. 1, Jan. 1992.

[B2] IEEE Std 100-1996, IEEE Standard Dictionary of Electrical and Electronics Terms.

[B3] IEEE Std 1062, 1998 Edition, IEEE Recommended Practice for Software Acquisition.

[B4] IEEE Std 1420.1-1995, IEEE Standard for Information Technology—Software Reuse—Data Model for Reuse Library Interoperability—Basic Interoperability Data Model (BIDM).

[B5] Jacobson, Ivar; Griss, Martin; and Jonsson, Patrik; *Software Reuse.* Reading, Mass: Addison Wesley, 1997.

[B6] Karlsson, Even-Andre, *Software Reuse: A Holistic Approach.* West Sussex, England: John Wiley & Sons, 1995.

[B7] NIST Special Publication 500-222, 1994, Glossary of Software Reuse Terms.

[B8] Poulin, J. S., *Measuring Software Reuse.* Reading, Mass: Addison Wesley, 1997.

Annex B

(informative)

Basic concepts

This annex contains the concepts upon which this standard was developed. The information in this annex is intended as an aid in understanding the concepts underlying this standard.

B.1 Software processes

Software processes are a road map or framework to developing systems. Software processes should provide a framework for practicing reuse that lays out reuse activities. Only when reuse considerations are integrated into the software life cycle can reuse become a natural and normal way of working.

A reuse-based software life cycle describes how software products are built through the use of assets and how to build and manage assets.

In order to practice reuse in the context of the software life cycle, determinations must be made about generally how reuse changes the software life cycle, exactly where to fit reuse into the software life cycle processes, and specifically what reuse processes, activities, and tasks should be included in the software life cycle.

B.1.1 Reuse changes the software life cycle

Reuse changes the software life cycle because reuse is a fundamentally different software paradigm.

Certainly, there can be no argument that reuse is a different paradigm. The traditional approach to software development is altered by reuse because reuse forces a different way of thinking and working. Its basic operating premise is: *Do not build from the first principle; that is, build software from scratch only as a last resort.* This is directly contrary to the basic operating premise found in most conventional software life cycle models where it is assumed, often by default, that life cycle deliverables are produced from scratch. For example, most software life cycle models do not describe how to create a design by reusing a domain architecture; nor do they describe how to build a domain architecture that can be reused in building a set of related software products.

To enable reuse, the software life cycle must start with reuse, end with reuse, and include reuse in all the phases in between. A common mistake has been to postpone reuse until the end of the life cycle. However, treating reuse strictly as an end life cycle activity severely limits the possibilities of what can be reused and the benefits that can be derived from reuse. This is because reuse at one phase not only avoids work during that phase but it also often opens up opportunities for reuse in other phases of the life cycle. For example, requirements reuse avoids inventing requirements that ignore what is already available and enables design reuse that can lead to subsequent code and test reuse. To maximize the benefits that reuse can deliver, it is important to “front-end load” the software life cycle with reuse thinking and planning.

The following three basic reuse characteristics should be incorporated into the software life cycle to promote and enable the practice of reuse:

— *Reuse themes* that govern software development decision making;

— *Reuse requirements* that broaden the focus from one software product at a time to multiple, related software products over time;

— *Reuse views* that slice the software process into two separate sides or views of reuse.

B.2 Reuse themes

What is meant by “reuse thinking” can be described as the following three reuse themes that developers must constantly keep in mind when developing software products and making development decisions:

— Take advantage of existing assets;
— Identify reuse opportunities;
— Prepare for reuse down the process road.

Software developers must think about each of these reuse themes when making every type of development decision ranging from planning strategies to implementation tools and language choices. Reuse thinking means being aware of what has been previously built, and if at all possible, reusing existing assets to produce new software products and enhancements to existing legacy systems.

Reuse thinking avoids the “copy and modify” syndrome that has typified much software development. While any software asset is likely to change over time, its effective reuse does not result in a proliferation of similar but subtly different versions, each of which must be maintained as a separate asset.

The first reuse theme emphasizes leveraging from work done in past software projects to the extent that reuse may even be given precedence over current software product requirements. In other words, the trade-offs between custom building software product deliverables from scratch in order to exactly match software product requirements are weighed against modifying software product requirements to enable the reuse of available assets to produce the software product in a faster, more cost-effective manner. With reuse, software product requirements specifications are adjusted to reuse available assets unless sound business and/or technical reasons that override taking advantage of reuse opportunities can be demonstrated. Reuse thinking allows software product requirements, especially optional requirements, to be influenced by what exists and can be reused.

The second reuse theme requires constantly looking for opportunities to practice reuse. Every project deliverable to be produced is viewed as potentially reusable in other software products and projects. An integral part of building each new project deliverable is to determine its reuse potential. If its potential for reuse is deemed great enough, then the asset’s specification, design, documentation, and so on should be created in a manner that enables its effective reuse. It may be necessary to look beyond project boundaries to determine reuse potential.

Whereas the second reuse theme requires a mindset concerned with one life cycle process, the third reuse theme requires also thinking about other life cycle processes. For example, reuse requires that the software design is examined with respect to not only its quality, but also with respect to how well it maximizes implementation reuse opportunities (e.g., reusing existing code or test cases, or using generator tools).

B.3 Reuse requirements

Another way in which reuse is different is the set of the unique demands it makes on the software life cycle. The requirements that reuse places on the software life cycle include

— Multi-system perspective where the project focus broadens beyond the requirements of a single system or single software product to take into account common requirements shared with other systems or software product implementations now and in the future.

- References to the past to leverage legacy systems by their reuse in part or in whole, and references to the future to leverage strategic systems plans to define a business-based scope for the reuse practice in an organization.
- Exploitation of commonality and accommodation of variability across a set of software products to identify assets that are common and reusable.

When practicing reuse, developers no longer can simply tackle one software product at a time. They must think in terms of a group or family of products related by the requirements, features, functions, and so forth that they have in common. Whenever developers build assets to implement these commonalities, they attempt to build the assets once in a way that allows them to be reused wherever the requirements occur. This often entails a generalization of the asset, compliance to standards, and a design strategy that stresses consistency across software product implementations.

Commonality/variability analysis is the process of identifying common assets in a set of existing and future software products. The purpose is to determine if it is advantageous to develop, re-engineer, or acquire reusable assets that then can be used in the creation and maintenance of software products in this set. Both the similarities and differences in the common assets are examined as the term “commonality/variability analysis” suggests. A reusable asset must be designed to be easily adjusted to handle possible variations in the asset that each reuse requires.

A reuse software life cycle must incorporate specific ways to handle each of the above requirements that reuse places on the software life cycle. Like reuse themes, reuse requirements run through and impose changes to the entire life cycle. For each software life cycle process, specific activities and tasks critical to achieving these reuse requirements should be added.

B.4 Reuse views

Finally, reuse is different because it changes how software developers view the software life cycle. Reuse divides the software life cycle into the following two distinct sides or views:

- Activities for using reusable assets in the creation of new software products (i.e., development *with* reuse view);
- Activities for creating, acquiring, or re-engineering reusable assets (i.e., development *for* reuse view).

B.4.1 Development *with* reuse

Development *with* reuse requires that reuse thinking occurs in every development activity, including project planning, implementation, and testing. Every opportunity to develop the software product and its related project deliverables, such as user documentation or test plans, from assets should be explored. The project banner reads “Create from scratch only as a last resort.” Each development decision should be made in the context of what assets are available to reuse.

Although development *with* reuse does not include creating new assets, all software deliverables should be created with reuse in mind (i.e., with the assumption that any deliverable may eventually be reused within or outside of the project).

In general, the following reuse activities should be integrated into a software life cycle to enable development *with* reuse:

— Extending the software development plan to include a software reuse plan that defines

  • What assets are available to reuse in the production of the software product;
  • Reuse target levels for the software project;
  • Tools needed to support the practice of reuse in the project;
  • How to measure the impact of reuse on the project.

— Searching for and evaluating application packages for use in the project;

— Evaluating the benefits and costs associated with practicing reuse in the project;

— Searching the reuse libraries and other internal and external sources for assets that can be reused to produce the software product;

— Reusing assets possibly by deriving modified/specialized instances in the resulting project deliverables while maintaining the deliverables from the generic asset level;

— Including reuse checks in project reviews, inspections, and audits.

B.4.2 Development *for* reuse

Whereas development *with* reuse is only indirectly concerned with the creation of new assets, this is the primary focus of development *for* reuse. Developing *for* reuse entails conducting projects, such as building domain models and architectures, and building new assets, or re-engineering existing assets to improve their reusability.

The following activities should be added to the software life cycle to support development *for* reuse:

— Performing domain analysis;
— Performing domain design;
— Building reusable assets.

B.5 Observations about practicing reuse

The three general observations to note about making reuse explicit within the software life cycle are as follows:

— *Practicing reuse (requires a broad-based, multi-system perspective).* The project team should broaden its vision from the current project and software product needs to also consider what is being produced and what is needed by other concurrent and future projects and software products. This enables project teams to take advantage of what is available to reuse and to recognize reuse opportunities now and in the future.

— *Planning is key.* The sooner a reuse opportunity is recognized, the better the chance the project team has to incorporate it into the project. The emphasis should be on higher-level, early-life-cycle deliverables and bigger assets (e.g., domain architectures) since this approach brings bigger reuse benefits to the project.

— *Good and continual communication within and across project teams is essential.* Reuse, even at the project level, demands that communication links be established between the project and other projects, and between software developers and domain engineers.

Annex C

(informative)

Reuse process objectives

The purpose of this annex is to describe the reuse objectives that the executor of the process can expect to achieve from performing the life cycle process. The reuse objectives of each life cycle process are listed in C.1 through C.8.

C.1 Acquisition Process

The use of the Acquisition Process should achieve the following reuse objectives:

— Review current assets to determine if what is needed exists or can be assembled in an effective manner to develop the needed software.

— Consider reuse as an option for acquisition as part of the analysis of risk, cost, and benefit criteria.

— Reuse an existing acquisition plan template, if an appropriate one is available, to create the acquisition plan.

— Include in the acquisition documentation any requirements for practicing reuse that the supplier will be required to meet.

— Include reusability tests and domain architecture compatibility tests in the acceptance test cases.

C.2 Asset Management Process

The use of the Asset Management Process should achieve the following reuse objectives:

— Develop and document an asset management plan, reusing an applicable asset management plan template, if any exists.

— Develop, document, and maintain an asset classification scheme.

— Define the organization’s requirements for an asset storage and retrieval mechanism.

— Design, implement, and maintain an asset storage and retrieval mechanism.

— Identify the organization(s) responsible for populating, managing, and maintaining the asset storage and retrieval mechanism.

— Define the access, acceptance, and retirement procedures and criteria for the asset storage and retrieval mechanism.

— For each asset submitted, evaluate the asset according to the acceptance criteria and procedures.

— For each accepted asset, classify the asset according to the reuse classification scheme and make it available through the asset storage and retrieval mechanism.

— Keep track of each reuse of the asset, modification requests for the asset, and problem reports concerning the asset; and send notifications to the reuse program administrator, the domain engineer, and the asset reusers.

— Perform configuration management for the asset.

C.3 Development Process

The use of the Development Process should achieve the following reuse objectives:

— Keep reuse considerations in mind when selecting a process life cycle model for use in the project.

— Select and use standards, methods, tools, and computer programming languages that support, enforce, and enable the practice of reuse in the project.

— Develop a project plan, integration plan, and installation plan, reusing appropriate plan templates, if any exist.

— Select and reuse applicable reusable system requirements specifications, if any exist, to create the system requirements specifications.

— Gather candidate software products and assets that may be used to construct a software product that satisfies the system requirements specifications.

— Identify and document in the project plans new software products and assets that may be developed within the scope and context of this project.

— Define, document, and evaluate software requirements, reusing applicable software requirements, if any exist.

— Design, document, and evaluate the database, based on the domain model, reusing applicable design and documentation assets, if any exist.

— Create user documentation, reusing applicable documentation assets, if any exist.

— Define testing requirements, reusing applicable test assets, if any exist.

— Produce a detailed design for each software item in the software product, reusing applicable design and documentation assets, if any exist.

— Produce code and documentation for each software unit and database, reusing applicable code and documentation assets, if any exist.

— Test each software unit and database, reusing applicable test assets, if any exist.

— Perform software qualification testing and system qualification testing, reusing applicable qualification tests, if any exist.

— Evaluate the reusability of the system requirements specifications, system architecture, software requirements specifications, software architecture, the software product, the software design, the code, and the test assets.

C.4 Domain Engineering Process

The use of the Domain Engineering Process should achieve the following reuse objectives:

— Select the representation forms for the domain models and the domain architectures.

— Establish the boundaries of this domain and its relationships to other domains.

— Define, classify, and document a set of domain models that capture the essential common and different features, capabilities, concepts, and functions in the domain.

— Evaluate the domain models.

— Specify assets that belong to the domain.

— Define, classify, and document domain architectures.

— Evaluate the domain architecture.

— Develop or acquire, document, and classify assets.

— Evaluate assets.

— Submit the domain models, domain architectures, and assets to the asset manager.

C.5 Maintenance Process

The use of the Maintenance Process should achieve the following reuse objectives:

— Analyze modification requests, problem reports, and implementation options in terms of their impact on any assets and opportunities to use assets to make the modifications.

— Provide modification requests/problem reports, and maintenance analysis information to the asset manager.

— Notify the asset manager of plans to migrate or retire software products that have been constructed from assets.

C.6 Operation Process

The use of the Operation Process should achieve the following reuse objective:

— Develop an operations plan for the system reusing an operations plan template, if an appropriate one exists.

C.7 Reuse Program Administration Process

The use of the Reuse Program Administration Process should achieve the following reuse objectives:

— Define the organization’s reuse strategy which includes its purpose, scope, goals, and objectives.

— Name a reuse sponsor.

— Identify reuse program participants.

— Establish a reuse steering function.

— Establish a reuse program support function.

— Prepare a reuse plan for implementing the practice of reuse in the organization.

— Establish feedback, communication, and notification mechanisms that operate between reuse program administrators, asset managers, domain engineers, developers, operators, and maintainers.

— Identify the domains in which the organization intends to investigate reuse opportunities or in which it intends to practice reuse.

— Assess the organization’s systematic reuse capability.

— Assess each domain to determine its reuse potential.

— Refine the organization’s reuse strategy and plan based on the results of the reuse assessments.

— Execute the reuse program implementation plan.

— Monitor and evaluate the reuse program.

— Improve the reuse program.

C.8 Supply Process

The use of the Supply Process should achieve the following reuse objectives:

— Prepare a proposal using a reusable proposal template, if an appropriate one exists, to respond to an RFP from an acquirer.

— Prepare a contract using a reusable contract template, if an appropriate one exists, to provide a system, software product, or asset to an acquirer.

— Determine the procedures and resources needed to manage a project to construct and deliver a system, software product, or asset with the use of assets to an acquirer.

Annex D

(informative)

Relationships

IEEE/EIA Std 12207.0-1996 establishes a common framework for the software life cycle by defining

— Primary processes that can be used to acquire, supply, develop, operate, and maintain software;

— Supporting processes that can be used to provide documentation, configuration management, quality assurance, verification, validation, joint review, and audit services for the software life cycle process;

— Organizational processes that can be used to provide management, improvement, infrastructure, and training services for the software life cycle process.

IEEE/EIA Std 12207.0-1996 places requirements upon software life cycle processes, but does not specify the details of how to implement these process requirements. IEEE/EIA Std 12207.0-1996 defines the software life cycle processes that are referenced in this standard. In addition, this standard uses the IEEE/EIA Std 12207.0-1996 processes as the foundation for defining reuse life cycle processes and integrates reuse activities and tasks into the IEEE/EIA Std 12207.0-1996 processes.

Annex E

(informative)

Tools

Tools that provide automated support for reuse make reuse easier to practice and help improve the quality of assets. Reuse-oriented tools extend or complement software development tools to the extent that they handle the reuse properties of assets. Table E.1 describes only the types of tools needed to support reuse, but not specific examples of such types of tools. While some of the categories of reuse tools listed in Table E.1 have been manual activities, and may continue to be, all categories have the potential to benefit from automation.

Table 1—Reuse support tool categories

Tool category	Type of tool	Functions performed by tool
Analysis and design	Reuse-oriented domain analysis and design	Assist domain engineers to recognize similarities among domain elements and to trial-fit elements into existing models and architectures. Assist developers to extend and improve their inventory of domain models and architectures.
	Legacy-asset salvage analysis	Analyze legacy assets in order to determine structural and functional patterns of similarity.
	Applications requirements analysis	Cross-match requirements to existing assets in order to minimize the deltas between what is available and what is needed.
	Reuse-oriented application design	Interrogate selected domain architectures in order to present developers with a list of options for instantiating the architectures' components. The result is a formal specification of the design of a software product sufficient to drive both documentation and construction tools.
Asset constructors	Smart editors	Find appropriate assets, and parse them so a developer can instantiate them to a particular context.
	Generators	Construct assets by combining design specifications with domain information contained within the tool.
	Assemblers	Construct assets by combining design specifications with assets external to the tool.
	Legacy-asset reconditioners	Package desired patterns, extracted by salvage analysis tools, into assets.
Asset testers	Adaptability testers	Assist domain engineers to determine and improve the ease-of-reuse of given assets.
	Generality testers	Assist domain engineers to determine and modify the domain of applicability of given assets.
Reuse management	Measuring reuse cost/benefits: • Reuse asset life cycle • Application life cycle	 • Determine costs to manage the asset storage and retrieval mechanisms and to amortize assets over time and over software products. Determine the relative costs and benefits of having various assets. • Determine project development and maintenance time and effort expended/avoided due to reuse.
	Asset configuration and version management	Keep track of how to access needed assets, ownership, and servicing responsibilities, and which version of an asset applies to which software products.
	Impact analysis of asset modification	Keep track of where assets are reused, and dependencies among assets.
	Asset inventory analysis	Determine the orthogonality (duplication and/or overlap) of the inventory, and the age and status of inventory items.
	Asset cataloging	Formal registration of assets into various asset storage and retrieval mechanisms, including updating browsing and retrieval tools with appropriate descriptors and search criteria.
	Asset search and retrieval	Browse and access assets, possibly allowing developers to enter appropriate parameter settings for both construction-time adaptation and runtime execution.
	Asset certification	Support the secure certification of an asset's status in terms of its scope of reusability—project, department, enterprise, industry, etc.

Annex F

(informative)

Glossary

This annex contains definitions for software reuse terms that are of general interest, but are not specifically referenced in this standard.

F.1 assembly: The part or compileable (or interpretable) software unit that results from assembling other parts.

F.2 asset pattern: An asset that formally describes the characteristics of a set of assets.

F.3 black box reuse: The reuse of unmodified software components [B8].

F.4 consumer: A person or organization that reuses software [B8].

F.5 context analysis: The process of determining the scope and boundary of a domain [B7].

F.6 domain implementation: The process of creating adaptable assets that can be reused in the development of software systems within a domain. Domain implementation may also include the specification of a software development process that describes how software systems in the domain are developed through reuse of assets [B7].

F.7 domain-specific language: A problem- or task-oriented modeling language used to simplify the process of assembling, customizing, generating, or configuring a system or component [B5].

F.8 domain-specific reuse: The reuse of assets within an application domain.

F.9 frame: A parameterized part that allows instantiation by other frames.

F.10 framework: A set of cooperating classes or frames that makes up a reusable design for a specific class of software.

F.11 generator: A mechanism that creates assets by inclusion of information that is internal to the generator.

F.12 gray box reuse: Reuse of a part by applying a few changes to the part.

F.13 horizontal reuse: A type of reuse where assets are used across domains.

F.14 life-cycle reuse%: The reuse level of a project measured by summing the actual levels of reuse during each phase of the software life cycle and adjusting those levels for the amount of effort expended in that life cycle phase [B8].

F.15 opportunistic reuse: The practice of reuse in an informal way without taking place in a global reuse improvement strategy [B6].

F.16 organized reuse: The practice of reuse according to a long-term plan [B6].

F.17 part: An asset that can be assembled into a compileable or interpretable software unit.

F.18 planned reuse: *See:* **organized reuse.**

F.19 relative cost of reuse: The portion of the effort that it takes to reuse a component without modification versus writing it new [B8].

F.20 relative cost of writing for reuse: The portion of the effort that it takes to write a reusable component versus writing it for one-time use only.

F.21 reusable asset: *See:* **asset** (see 3.3).

F.22 reusable component: *See:* **part.**

F.23 reuse catalog: A set of descriptions of assets with a reference or pointer to where the assets are actually stored or information about how they can be acquired.

F.24 reuse interface: A set of parameters that governs the adaptation of assets.

F.25 reuse interoperability: An indication of the capability to exchange assets, asset descriptions, and other information among reuse libraries.

F.26 reuse library: A classified collection of assets that allows searching, browsing, and extracting.

F.27 reuse maturity: A variation of the Software Engineering Institute (SEI) Process Capability Maturity Model (CMM) used to describe how well an organization has developed and practices reuse [B8].

F.28 reuse ratio: The portion of the software in a given software product that came from reusable parts.

F.29 salvage: The process of finding and re-engineering an existing asset so that it may potentially be reused in subsequent software products, developments, or maintenance [B7].

F.30 scalability: The degree to which assets can be adapted to support application engineering products for various defined measures.

F.31 systematic reuse: The practice of reuse according to a well-defined, repeatable process. *See also:* **organized reuse.**

F.32 vertical reuse: A type of reuse in which assets are reused within an application domain.

F.33 white box reuse: Reuse of a component by applying major changes to the component [B6].