Glossary of Software Reuse Terms

Susan Katz
Christopher Dabrowski
Kathryn Miles
Margaret Law

Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-0001

December 1994

U.S. Department of Commerce
Ronald H. Brown, Secretary

Technology Administration
Mary L. Good, Under Secretary for Technology

National Institute of Standards and Technology
Arati Prabhakar, Director
The National Institute of Standards and Technology was established in 1988 by Congress to "assist industry in the development of technology . . . needed to improve product quality, to modernize manufacturing processes, to ensure product reliability . . . and to facilitate rapid commercialization . . . of products based on new scientific discoveries."

NIST, originally founded as the National Bureau of Standards in 1901, works to strengthen U.S. industry's competitiveness; advance science and engineering; and improve public health, safety, and the environment. One of the agency's basic functions is to develop, maintain, and retain custody of the national standards of measurement, and provide the means and methods for comparing standards used in science, engineering, manufacturing, commerce, industry, and education with the standards adopted or recognized by the Federal Government.

As an agency of the U.S. Commerce Department's Technology Administration, NIST conducts basic and applied research in the physical sciences and engineering, and develops measurement techniques, test methods, standards, and related services. The Institute does generic and precompetitive work on new and advanced technologies. NIST's research facilities are located at Gaithersburg, MD 20899, and at Boulder, CO 80303. Major technical operating units and their principal activities are listed below. For more information contact the Public Inquiries Desk, 301-975-3058.

Office of the Director
- Advanced Technology Program
- Quality Programs
- International and Academic Affairs

Technology Services
- Manufacturing Extension Partnership
- Standards Services
- Technology Commercialization
- Measurement Services
- Technology Evaluation and Assessment
- Information Services

Materials Science and Engineering Laboratory
- Intelligent Processing of Materials
- Ceramics
- Materials Reliability
- Polymers
- Metallurgy
- Reactor Radiation

Chemical Science and Technology Laboratory
- Biotechnology
- Chemical Kinetics and Thermodynamics
- Analytical Chemical Research
- Process Measurements
- Surface and Microanalysis Science
- Thermophysics

Physics Laboratory
- Electron and Optical Physics
- Atomic Physics
- Molecular Physics
- Radiometric Physics
- Quantum Metrology
- Ionizing Radiation
- Time and Frequency
- Quantum Physics

Manufacturing Engineering Laboratory
- Precision Engineering
- Automated Production Technology
- Intelligent Systems
- Manufacturing Systems Integration
- Fabrication Technology

Electronics and Electrical Engineering Laboratory
- Microelectronics
- Law Enforcement Standards
- Electricity
- Semiconductor Electronics
- Electromagnetic Fields
- Electromagnetic Technology
- Optoelectronics

Building and Fire Research Laboratory
- Structures
- Building Materials
- Building Environment
- Fire Safety
- Fire Science

Computer Systems Laboratory
- Office of Enterprise Integration
- Information Systems Engineering
- Systems and Software Technology
- Computer Security
- Systems and Network Architecture
- Advanced Systems

Computing and Applied Mathematics Laboratory
- Applied and Computational Mathematics
- Statistical Engineering
- Scientific Computing Environments
- Computer Services
- Computer Systems and Communications
- Information Systems

1 At Boulder, CO 80303.
2 Some elements at Boulder, CO 80303.
PREFACE

The Computer Systems Laboratory (CSL) within the National Institute of Standards and Technology (NIST) has a mission under Public Law 89-306 (Brooks Act) to promote the "economic and efficient purchase, lease, maintenance, operation, and utilization of automatic data processing equipment by federal departments and agencies." When a potentially valuable innovation in information technology first appears, CSL may be involved in research and evaluation. Later on, CSL may serve government interests by participating in standardization of the results of such research, in cooperation with voluntary industry standards bodies. Finally, CSL helps federal agencies make practical use of existing standards and technology through consulting services and the development of supporting guidelines and software.

If certain commercial software products and companies are identified in this report for purposes of specific illustration, such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.
Reports on Computer Systems Technology

The National Institute of Standards and Technology (NIST) has a unique responsibility for computer systems technology within the Federal government. NIST's Computer Systems Laboratory (CSL) develops standards and guidelines, provides technical assistance, and conducts research for computers and related telecommunications systems to achieve more effective utilization of Federal information technology resources. CSL's responsibilities include development of technical, management, physical, and administrative standards and guidelines for the cost-effective security and privacy of sensitive unclassified information processed in Federal computers. CSL assists agencies in developing security plans and in improving computer security awareness training. This Special Publication 500 series reports CSL research and guidelines to Federal agencies as well as to organizations in industry, government, and academia.

National Institute of Standards and Technology Special Publication 500-222
CODEN: NSPUE2
ABSTRACT

One method proposed for increasing the efficiency of software production in the development of large, reliable software applications is the systematic reuse of existing software products. Effective software reuse will require new techniques to supplement traditional software engineering practices. Preliminary research has already produced new methods and reports. As a result, new terminology has emerged. This report provides a baseline set of recommended definitions for terms commonly used in the software reuse community. The glossary will be expanded as further research results become available and are evaluated for use in software reuse programs.
ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of Carmelo Montanez-Rivera and Rebecca Segal of NIST in doing early work in developing this glossary. Dr. C. Ronald Green and the CRWG Reuse Committee, Linda Brown, Stan Levine, Doug Rosson, Mason Myers, and members of the MIWG/RAAT/IRAG committee reviewed versions of this paper and provided valuable advice. We would also like to thank Thomasin Kirkendall, Dolores Wallace, Jay Watkins, Elwood Schneider, Arnold Johnson and Dr. Selden Stewart of NIST, as well as Spencer Peterson, Dr. Hasan Sayani, and David Wheeler. We would also like to thank Jeni Lindeman, Peggy Himes, and Dorothy Snyder for proofreading this document.
1. INTRODUCTION

One method proposed for increasing the efficiency of software production in the development of large, reliable software applications is the systematic reuse of existing software products. Successful software reuse will require techniques to supplement traditional software engineering practices. Research has already produced new methods, a growing body of technical literature, and a number of software systems developed through reuse. As a result, new terminology has emerged. The glossary provided in this report will serve as a baseline set of recommended definitions for use by the software reuse community.

1.1 Background

In 1992, the Ballistic Missile Defense Organization (BMDO) established a Manufacturing Operations Development and Integration Laboratory (MODIL) for software producibility at NIST to transition emerging technology from the research community to government and industry. The purpose of the MODIL was to promote software producibility through (1) integration and extension of new methods and techniques into operational products and (2) the transfer of these products from the laboratory environment to the field. Earlier versions of this report, under the BMDO project, were produced from January, 1993, through November, 1993.

In 1994, the Department of Defense (DoD) Software Reuse Management Issues Working Group (MIWG), the Special Interest Group - Ada (SIGAda) Reuse Working Group Reuse Acquisition Action Team (RAAT), and the Council of Defense and Space Industry Associations (CODSIA) Industry Reuse Advisory Group (IRAG) established a working group to address adoption of the glossary as a baseline document. Funding support was provided by the Defense Information Services Agency/Center for Information Management (DISA/CIM). A version of this report, dated October, 1994, was produced for use by the MIWG/RAAT/IRAG.

1.2 Purpose

The purpose of this report is to present a baseline glossary of software reuse terms. The report also outlines plans for maturing the glossary. The purpose of developing and maintaining the glossary is to promote a consistent understanding of software reuse terms used in DoD and industry and to provide common definitions for use in the software reuse community.

1.3 Audience

The intended audience of this report is:

- The DoD software reuse community and other government agencies, industry, and academia involved in software reuse.
Personnel practicing reuse (e.g., software engineers).
- Personnel engaged in management and planning.
- The acquisition community.

1.4 How the Glossary Was Developed

DoD standards and program documents provided the background information base for developing the initial glossary. Recent reports from industry and academia were reviewed for additional terminology and usages. After this initial survey work was completed, the set of multiple definitions for each term was reviewed, and a recommended definition was either chosen or developed from the information obtained. Similar terms were compared. Related terms were cross-referenced. Where there were multiple terms for the same idea, the best term was chosen as the recommended term on the basis of the literature survey. The alternate terms are still captured in the glossary, to reflect real-world usage, but they refer the reader back to the definition of the preferred term.

2. USING THE GLOSSARY

The glossary is intended as a reference dictionary for anyone involved in software reuse. For any subject area, it provides a number of terms which are commonly in use, rather than just the recommended terms. Terms which may be synonyms of other terms are listed in their own right, so that they may be looked up alphabetically by users who encounter them in reuse literature or discussions. In these cases, the definition of the synonym will refer the reader to the recommended term where the actual definition is found.

Highlighted terms are defined elsewhere in the glossary. Highlighted/italicized words are defined in Appendix A - Related Software Engineering Terms. Where applicable, the glossary refers to related terms so that the reader can gain a broader understanding of the usage of the terminology (e.g., "See also...", "Compare..."). Where it was found in the reuse literature that several different terms were being used to mean the same thing, the glossary defines the recommended term and provides references to the recommended term from the alternate terms.

The reference provided for each definition represents the main source for that definition. The references are intended to acknowledge contribution and not necessarily to indicate an exact quote. Where no reference is attributed, the definition was synthesized from a number of sources and/or from a broad understanding of the topic area.

Generally, the glossary provides a single recommended definition for each term. Occasionally, alternate definitions are provided when terms are used in different contexts. In the future,
further alternative definitions may be added. A discussion of plans for maturing the glossary is provided in section 3.

Certain terms may have different meanings in different disciplines, such as in general software engineering, in domain analysis, or in reuse library systems. The glossary specifies the discipline context by the phrase "As used in . . . ," for example, "As used in Domain Analysis, . . . ."

For some terms, the definitions are followed by a paragraph marked "Discussion," explaining how the term is related to software reuse or to other terms in the glossary.

The glossary does not provide definitions for products or methods developed either commercially or in research institutions.

Appendix A contains definitions for related software engineering terms. It is provided as a quick reference for terms which do not have a definition specific to the context of reuse but are used in definitions found in the main glossary.

3. PLANS FOR MATURING THE GLOSSARY

Definitions will continue to be obtained from an ongoing survey of pertinent reuse literature and electronic access systems. As software reuse evolves and expands, the glossary will grow. It is important that the accumulation and evolution of terms occur in a consistent and controlled manner. To this end, the following guidelines for maturing the glossary are proposed:

- New definitions will be added and existing definitions may be refined. Changes will be based on input from reviewers and on new literature that reflects experience with software reuse.

- The number of sources from which terms are taken will be expanded. Sources especially relevant to DoD will receive primary consideration.

- DoD directives, military standards, and documents will be used for guidance on selecting the definitions recommended for usage.

- Cross-referencing will be expanded, emphasizing the relationships between terms.

- Where it may be important, the origin and background of a term may be provided to give the reader greater insight into the usage of the term.
GLOSSARY

Adaptability:
The ease with which software can be modified to meet new Requirements. [DODSTR,p47]

Adaptation:
The process of modifying a Software System or Asset to perform its function in a different manner or on different data than was originally intended. [COHEN92,p14]

Architecture:
1. Organizational structure of a System or Asset. [ANSI90, p10]
2. The structure of Components, their interrelationships, and the principles and guidelines governing their Design and evolution over time.

Discussion:
It is recommended that the more specific terms (Domain Architecture, Software System Design) be used. There are also other types of Architectures, such as strategic Architectures, enterprise Architectures, standards Architectures, logical and physical Architectures, and hardware and software Architectures. Each has unique characteristics and may have unique Applications.

Asset:
Any product of the software life cycle that can potentially be Reused. This includes: Domain Model, Domain Architecture, Requirements, Design, code, Databases, Database Schemas, documentation, user manuals, test suites, etc. [DODV&S]

Discussion:
In Reuse, an Asset may be a distinct piece of information, or describe or perform a distinct function, or be a Feature of a Software System. An Asset is a part which is marked by its ability to be integrated into different wholes (e.g., Software Systems, Domain Models, Designs, etc.). Each of these wholes may, in turn, be considered an Asset. The term Component is a generic Software Engineering term that is often used as a synonym for Asset.

Compare: Reusable Software Asset

Asset Certification:
1. The process of assessing that an Asset correctly performs its stated function(s), adheres to quality and Reuse standards, and, possibly, is formally proven correct. [STARS1,p51]
2. Alternatively, the process of confirming that the information about an Asset, as given by the Reuse Library, is correct (including known errors).

Discussion:
To become part of a Reuse Library, an Asset, must be Cataloged. Some Reuse Libraries may also require Asset Certification.

Asset Evaluation:
The process of determining whether a particular Asset fits Requirements and Constraints of a particular Software System (as documented in its Architecture, or in its Domain Model). [STARS1,p51]

Discussion:
Asset Evaluation may occur when an Asset is about to be Reused, as opposed to Asset Certification, which occurs when the Asset is entered into the Reuse Library. Evaluation may also occur when a determination is made by a Reuse Library that an Asset fits a Domain Model or generic Architecture.

Asset Library:
Synonym for Reuse Library.

Catalog:
The index of information about Assets that is maintained for a Reuse Library's contents. [RIG93,p2]

Cataloging:
Placing information about an Asset into a Software Reuse Library. The Asset plus its Catalog information become a Reusable Software Asset. [DODSTR,pA-3]

Certification:
See Asset Certification

Cohesion:
1. The manner and degree to which the tasks performed by a single Asset are related to one another. [ANSI90,p11]

2. The degree to which an Asset's structure is unified in support of its function. [DODSTR,p47]

3. The degree of functional relatedness of processing elements within a single Module. [YOURDON,p407]

Discussion:
A high level of Cohesion in an Asset may aid in its Reusability.

Compare: Coupling
Cohesiveness:
See Cohesion

Commonality:
1. In Domain Analysis, an Asset or a distinct part, function, or Feature that is characteristic of the class of Systems within a Domain and that is represented in a Domain Model or Domain Architecture.

2. A measure of how common the problem (or Application) is that is being solved with a given System; one of the factors in program generality (and in Reusability). [YOURDON,p407]

Compare: Difference

Component:
One of the parts, either hardware or software, that make up a system. [ANSI90,p18] Often used as a synonym for Asset.

Constraint:
1. As used in software Reuse and Reuse Libraries, a rule or restriction governing the Reuse of Assets, particularly as relates to Environment.

2. As used in Software Engineering, a functional or operational Requirement for a Software System that limits the possible solution space. [STARS1,p51]

Context:
1. The circumstances, situation, or Environment in which a particular Software System or a Domain exists. [KANG90,p2][DODINIT,pB1][PETER91]

2. The academic discipline in which a particular term has meaning (e.g., the disciplines of Software Engineering, Domain Analysis).

Context Analysis:
The process of determining the scope and boundary of a Domain, as described in the Feature-Oriented Domain Analysis (FODA) Method. [KANG90] See Domain Definition.

Discussion:
In the Domain Engineering process, Context Analysis occupies the same place as Domain Definition, except that in FODA, Context Analysis emphasizes analysis of interaction between Systems in a Domain and the domain environment.

Coupling:
1. The degree of data or control connectivity between different Assets of a Software System. [DODSTR,p48]
2. The manner and degree of interdependence between Assets. [ANSI90,p22]

3. A measure of the strength of interconnection between one Asset and another. [YOURDON,p409]

Discussion:
A high degree of Coupling between Assets in a Software System may limit their Reusability. Coupling makes modification to the System complex and difficult. Because Coupled Assets cannot be separated from each other easily and used alone, the scope of Reuse is narrowed.

Compare: Cohesion

Decoupling:
The process of making Assets more independent of one another to decrease the impact of changes to, and errors in, the individual Assets. [ANSI90,p25]

Compare: Coupling

Difference:
In Domain Analysis, an Asset or a distinct part, function, or Feature that distinguishes Systems within a domain from each other and that is represented in a Domain Model or Domain Architecture.

Compare: Commonality

Domain:
A distinct functional area that can be supported by a class of Software Systems with similar Requirements and capabilities. A Domain may exist before there are Software Systems to support it. [PRIET91,p14], [DODV&S,p2]

Domain Analysis (DA):
1. The analysis of Systems within a Domain to discover Commonalities and Differences among them. [DODV&S,p2]

2. 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 [PRIET90].

3. The result of the process in (1) and (2).

Discussion:
Domain Analysis can be viewed as systems analysis applied across a class of Software Systems in a Domain. [PREIT91,p14]
The principal products of Domain Analysis are the Domain Model, and in some Domain Analysis methods (but not all), the Domain Architecture.

**Domain Analyst:** A person who performs Domain Analysis. [DABR92,p15]

**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. 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. The Domain Architecture documents Design, whereas the Domain Model documents Requirements.

**Domain Description:**
See Domain Definition

**Domain Definition:**
1. The process of determining the scope and boundaries of a Domain.
2. The result of the process in (1).

**Discussion:**
The process establishes what major functions and capabilities are within the Domain, what functions and capabilities are excluded from the Domain, and what interactions exist with external Domains. In the Domain Engineering process, a Domain Definition is established prior to beginning the development of the Domain Model.

**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 can include the following activities: Domain Definition, Domain Analysis, developing the Domain Architecture, and Domain Implementation.

**Domain Expert:**
Individual who is intimately familiar with the Domain and can provide detailed information to the Domain Analysts. [DABR92,p16]

**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.

Domain Manager:
Individual or organization responsible for managing the definition, use, evaluation, and evolution of Assets within the Domain. [DABR92,p15]

Domain Model:
A product of Domain Analysis which 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 Commonalities and variabilities among Requirements for Software Systems in the Domain. [DODV&S,p4]

Discussion:
The Domain Model documents Requirements, whereas the Domain Architecture documents Design.

Domain Scoping:
See Domain Definition

Environment:
The circumstances or conditions in which a Software System executes. This includes interfaces with an operating system, interfaces with other Systems, dependency on Database management systems, hardware or network Constraints, or any factor that affects the functioning of the Software System.

Extraction:
The retrieval of Assets from a Reuse Library. [RIG93,p2]

Faceted Classification:
A method derived from the field of library science which can be used to provide multiple access routes to Reusable Software Assets in a Reuse Library. [PRIET90], [DODSTR,pA-3]

Discussion:
Each facet in the scheme represents a particular aspect of a software Asset such as its function, its operating environment, or other significant attributes. Each user of a Reuse Library may use different facets, or key information, in searching for relevant Assets.

Feature:
An attribute or characteristic of a System that is meaningful to, or directly affects, the user, developer, or other entity that interacts with a System.
Discussion:
Feature concepts are defined differently by several Domain Analysis methods, including [KANG90], [MOORE91], and [STARS1].

Horizontal Domain:
A Domain that provides information or services to more than one Domain. Examples of Horizontal Domains include communications, graphical user interfaces, and Databases.

Compare: Vertical Domain

Horizontal Reuse:
Reuse of Assets in more than one Vertical Domain.

Compare: Vertical Reuse

Interoperability:
See Reuse Library Interoperability

Library:
See Reuse Library

Library Metric:
A standard of measure that can support quantitative comparisons and evaluations related to Reuse Library operations. [RIG93,p3]

See also: Metric

Library Mechanism:
See Reuse Library System

Metric:
1. A quantitative measure of the degree to which a System, Asset, or process possesses a given attribute. [ANSI90,p47]

2. The definition, algorithm, or mathematical function used to make a quantitative assessment of a software product or process. [PENG93]

Discussion:
Metrics are used in making quantitative assessments of such topics as the amount of Reuse, the reliability of Assets, the effort associated with reusing Assets or other characteristics of a Domain or Software System.

Opportunistic Reuse:
The ad hoc Reuse of Assets in the development of Software Systems using a software development process that has not been altered to accommodate Systematic Reuse. In Opportunistic Reuse, the developer determines where Reuse can be applied to develop a Software System without the organized use of Domain
Engineering products during successive stages of a Software Engineering process.

Compare: Systematic Reuse

Planned Reuse:
See Systematic Reuse

Replication:
The repetition or duplication of an Asset within a Software System.

Reusability:
1. The degree to which an Asset can be used in more than one Software System, or in building other Assets, with little or no Adaptation. [PETER91]

2. In a Reuse Library, the characteristics of an RSA that make it easy to use in different Contexts. [RIG93,p3]

Reusable Asset:
See Reusable Software Asset (RSA)

Reusable Component:
See Reusable Software Asset (RSA)

Reusable Software:
Software Designed and implemented for the specific purpose of being Reused. [PETER91] Reusable Software is a broad term applying to Assets, Applications, or Software Systems. The recommended term is Reusable Software Asset (RSA).

Reusable Software Asset (RSA):
An Asset that has been Catalogued and is stored in a Reuse Library. [RIG93,p1]

See also Asset and Asset Certification

Discussion:
The term Reusable Software Asset connotes more value added than an Asset. This value lies in the meta-information regarding the Asset in its Catalog entry, and in the Reuse Library or Reuse Library System in which it is contained. These additions make it easier to locate appropriate Assets which have been evaluated for Reuse and to determine information about their suitability for Reuse in a particular context. In contrast, an Asset is a software product which may or may not be locatable and/or reusable, and an Asset may or may not have to be certified, depending on the Requirements of the Reuse Library. An Asset may be stored in a Repository.
**Reusable Software Component (RSC):**

Synonym for **Reusable Software Asset (RSA).**

**Reusable Software Library:**

See **Reuse Library**

**Reuse:**

1. To use again. [Webster]

2. The process of implementing or updating **Software Systems** using existing software **Assets.** [DODSTR,p2] [COHEN92,p5] [KANG90,p3] [DODINIT,pB3] [PETER91]

**Discussion:**

Reuse is the application of **Reusable Software Assets,** with or without adaptation, to more than one **Software System.** **Reuse** may occur within a **Software System,** across similar **Software Systems,** or in widely different **Software Systems.** [DODV&S,p2]

**Reuse-Based Development:**

The use of a disciplined, systematic, quantifiable approach to the development, operation, and maintenance of software (where **Reuse** is a primary consideration in the approach). [PETER91]

**Discussion:**

**Reuse-Based Development** uses **Domain Engineering** products during systems/Software Engineering.

**Reuse Library:**

A controlled collection of **Reusable Software Assets,** together with the procedures and support functions required to provide the **RSAs** for **Reuse.** The procedures and support functions may be automated via a **Reuse Library System.** If this is the case, then the **Reuse Library** contains both the **RSAs** and the **Reuse Library System.** [RIG93,p4]

Compare: **Software Repository**

**Reuse Library Interoperability:**

The ability of two or more distinct, heterogeneous **Software Reuse Libraries** to dynamically provide access to the other’s **Assets,** **Asset descriptions,** and other available information. [STARS1], [DODV&S]

**Reuse Library System:**

A **Software System** that automates the procedures and support functions of a **Reuse Library.** A **Reuse Library System** includes the storage capabilities of a **Software Repository,** but it is more than a storage facility. It provides capabilities that assist the user in accessing the contents of what is stored.
(e.g., browsing, hypertext, etc.). Reusable Assets are stored in a Reuse Library which is supported by a Reuse Library System. [BRAUN92,p8]

**Reverse Engineering:**
The process of analyzing an existing Software System to derive its Design, Requirements, and other Software Engineering products.

**Salvage:**
The process of finding and Reengineering an existing Component so that it may potentially be Reused in subsequent Applications, developments, or maintenance. [GPALS92,p21]

**Software Architecture:**
See Architecture

**Software Application:**
See Application

**Software Asset:**
See Asset

**Software Component:**
See Component

**Software Engineering Environment:**
The supporting hardware, software, and firmware used in the production of software throughout its life cycle. Typical elements include computer equipment, compilers, assemblers, operating systems, debuggers, simulators, emulators, test tools, documentation tools, Requirements, Designs, CASE tools, and Database management systems. [ANSI90,p67]

**Discussion:**
A Reuse Library System can be among the tools in a Software Engineering Environment. Other tools, such as CASE tools, can enhance Reuse and improve the quality of information about Assets in a Catalog.

**Software Repository:**
A permanent, archival storage place for software and related documentation. [ANSI90,p68] [PETER91]

**Discussion:**
A Software Repository is simply a storage mechanism for Assets, as opposed to a Reuse Library System, which provides both storage and user-friendly mechanisms to find and access stored RSAs.

Compare: Reuse Library
Software Reuse:
See Reuse

Software Reuse Library:
See Reuse Library

Software System:
An organized collection of computer programs, procedures, associated documentation, and data (i.e., Assets) pertaining to the operation of a computer system that accomplish a specific function or set of functions. [ANSI90,p66,74]

Subdomain:
A Domain that can be viewed as part of a larger Domain (Context-dependent).

Support Domain:
Synonym for Horizontal Domain.

System:
1. A collection of Assets organized to accomplish a specific function or set of functions. [ANSI90, p73]

2. A set of interrelated parts, which may include people, methods, hardware, software, and/or firmware, that function together to achieve an overall purpose.

Discussion:
Generally more inclusive than a Software System, although the terms are sometimes used synonymously.

Systematic Reuse:
Reuse of Assets in which Software Systems are developed using a Software Engineering process that is specifically structured for Reuse. Systematic Reuse means that software development is guided by an organized use of Domain Engineering products (including a Domain Model, Domain Architecture, and other Assets) during successive stages of a Software Engineering process.

See Domain Engineering

Compare: Opportunistic Reuse

Traceability:
1. The degree to which a relationship can be established between two or more products having a predecessor-successor or superior-subordinate relationship to one another. [ANSI90,p78]

2. In Software Engineering, the successful cross-referencing across a Software System’s related Assets, from operational software Modules back to their original Requirements,
including all intervening stages of development and testing, and related documentation.

3. In Domain Engineering, the characteristic of Domain models, Domain Architectures, Designs, and Software Systems that, for any particular Asset, identifies and documents the derivation path (from the preceding phase in the Software Engineering process) and allocation/flowdown path (to the successive phase) of Requirements and Constraints. [STARS1,p56]

**Vertical Domain:**
A Domain which addresses aspects of a single function or Application area. [BRAUN92,p8] Examples include payroll Systems, automated weapons Systems, robotic control Systems. A Vertical Domain draws on capabilities from any Horizontal Domains that support its purpose.

Compare: Horizontal Domain

**Vertical Reuse:**
Reuse of Assets within a Vertical Domain.

Compare: Horizontal Reuse
APPENDIX A

RELATED SOFTWARE ENGINEERING TERMS

Application:
1. Synonym for Software System.
2. A Software System that interacts directly with some non-software system (e.g., human, robot, etc.).

Application Generator:
A type of tool that uses software Designs and/or Requirements to generate entire software Applications automatically, including program source code and program control statements.

Discussion:
An Application Generator may be one of the tools in a Software Engineering Environment, or it may be used independently.

Compare: Source Code Generator

Code Generator:
1. A synonym for Source Code Generator.
2. In compiler technology, the back end of a compiler, which builds object code or assembly code.

Configuration Item:
1. In DoD software development, an aggregation of hardware or software that satisfies a function and is designated by the Government for separate Configuration Management. [MIL-STD-973]
2. In DoD software development, a digital data file [MIL-HDBK-59A] that is designated by the Government for separate Configuration Management.
3. An aggregation of hardware, software, or both, that is designated for Configuration Management and treated as a single entity in the Configuration Management process. [ANSI90, p20]

Configuration Management:
In DoD software development, the discipline applying technical and administrative direction and surveillance over the life cycle of Configuration Items to:

a. Uniquely identify Configuration Items, including versions and their status.
b. Identify and document the functional and physical
characteristics of Configuration Items.

c. Control changes to Configuration Items and their related documentation.

d. Record and report information needed to manage Configuration Items, including the approval status of proposed changes and the implementation status of approved changes.

e. Audit Configuration Items to verify conformance to specifications.

[MILSTD-973]

Discussion:
In Software Reuse, the process of cataloging and classifying all Assets, recording and controlling the release and change of these assets throughout the system life cycle, recording and reporting the status of Assets and change requests, and verifying the completeness and correctness of Assets.

Database:
A collection of interrelated data stored together in one or more computerized files. [ANSI90, p25]

Discussion:
A Database could be an Asset, the mechanism for storing a set of related Assets, or a Catalog.

Database Schema:
The information that describes the structure of the data in a Database.

Discussion:
The Design of a Catalog or Library could contain a Database Schema. However, the Database refers to a Database Schema that is populated with actual instances of data. Similarly, for a delivered Software System, the Database Schema and the Database represent separate Assets, and support different types of Reuse.

Design:
1. The process of defining the Architecture, Components, interfaces, and other characteristics of a System or Component. [ANSI90, p25]

2. The specification (n.) defining a solution for meeting Requirements.

3. The process (v.) of defining a solution for meeting Requirements.
Extensibility:
The ease with which a System or Component can be modified to increase its storage or functional capacity. [ANSI90, p32]

Functional Baseline:
1. The version of an Asset that is established after initial completion of the definition of the Software System functions and associated data, interface characteristics, functional characteristics for key Configuration Items, and tests required to demonstrate achievement of each specified characteristic. This baseline is normally controlled by the customer; e.g., the Government. [MIL-STD-490A]

2. In Configuration Management, the initial, approved technical documentation for a Configuration Item. [ANSI90,p35]

Module:
A Component which is either code or Design. Examples of Modules include source code Modules, object code Modules, and Design Modules.

Portability:
The extent to which a Module originally developed on one computer or operating system can be used on another computer or operating system. [BRAUN92,p8]

Discussion:
The greater the Portability of an Asset, the greater the potential for reuse. Therefore, Portability should be one of the items of meta-information accompanying each Asset in a Catalog.

Reengineering:
The process of examining, altering, and re-implementing an existing Software System to reconstitute it in a new form. [STARS1,p55]

Requirement:
1. A condition or capability needed by a user to solve a problem or achieve an objective. [ANSI90,p62]

2. A condition or capability that must be met or possessed by a System or system Component to satisfy a contract, standard, specification, or other formally imposed document. [ANSI90,p62]

Discussion:
The set of all Requirements forms the basis for subsequent development of the System or Asset. When considering an Asset for Reuse, the Catalog information about that Asset must be evaluated to determine whether or not the Asset meets the Requirements of the System for which it is being considered.
Software Engineering:
The use of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the use of engineering principles in the development of software. [ANSI90,p67]

Source Code Generator:
A tool that uses software Requirements and/or Designs to automatically generate source code. An Application Generator generates entire Applications, whereas a Source Code Generator may generate smaller pieces of source code. Synonym for Code Generator (however, Source Code Generator is the recommended term). Compare with Application Generator.
BIBLIOGRAPHY


INDEX

Adaptability (4)
Adaptation (4), (11), (12)
Application (6), (12), (13), (15), (16), (19)
Application Generator (16), (19)
Applications (ii), (1), (4), (8), (11), (13), (16), (19)
Architecture (4-9), (13), (14), (17)
Architectures (4), (15)
Asset (4-7), (9-13), (15), (17-18)
Asset Certification (4), (5), (11)
Asset Evaluation (5)
Asset Library (5)
Asset’s (5)
Assets (5-15), (17)

Catalog (5), (11), (13), (17), (18)
Cataloged (5)
Cataloging (5), (17)
Certification (4), (5), (11)
Code Generator (16), (19)
Cohesion (5-7)
Cohesiveness (6)
Commonalities (7)
Commonality (6), (7)
Component (4), (6), (11-13), (17), (18)
Configuration Item (16), (18)
Configuration Management (16), (18)
Constraint (6)
Constraints (5), (9), (15)
Context (3), (6), (11), (14)
Context Analysis (6)
Contexts (2), (11)
Coupled (7)
Coupling (5-7)

Database (4), (9), (13), (17)
Database Schema (17)
Database Schemas (4)
Databases (4), (10)
Decoupling (7)
Design (4), (8), (9), (13), (17), (18)
Designed (11)
Designs (4), (8), (13), (15), (16), (19)
Difference (6), (7)
Differences (7)
Domain (3-10), (12), (14), (15)
Domain Analysis (3), (6-10)
Domain Analyst (8)
Domain Analysts (8)
Domain Architecture (4), (6-9), (14)
Domain Definition (6), (8), (9)
Domain Engineering (6), (8), (11), (12), (14), (15)
Domain Expert (8)
Domain Implementation (8)
Domain Manager (9)
Domain Model (4-9), (14)
Domain Models (4)
Domain Scoping (9)
Domains (8), (10), (15)

Environment (1), (6), (9), (13), (16)
Extensibility (18)
Extraction (9)

Faceted Classification (9)
Feature (4), (6), (7), (9), (10)
Functional Baseline (18)

Horizontal Reuse (10), (15)
Interoperability (10), (12)

Libraries (5), (6), (12)
Library (3), (5), (9-14), (17)
Library Mechanism (10)
Library Metric (10)
Library’s (5)

Metric (10)
Module (5), (18)
Modules (14), (18)

Opportunistic Reuse (10), (14)

Planned Reuse (11)
Portability (18)

Reengineering (13), (18)
Replication (11)
Requirement (6), (18)
Requirements (4), (5), (7-9), (11), (13-18)
Reusability (5-7), (11)
Reusable Asset (11)
Reusable Component (11)
Reusable Software (4), (5), (9), (11), (12)
Reusable Software Asset (4), (5), (11), (12)
Reusable Software Component (12)
Reusable Software Library (12)
Reuse (1), (ii), (iii), (1-15), (17), (18)
Reuse Libraries (5), (6), (12)