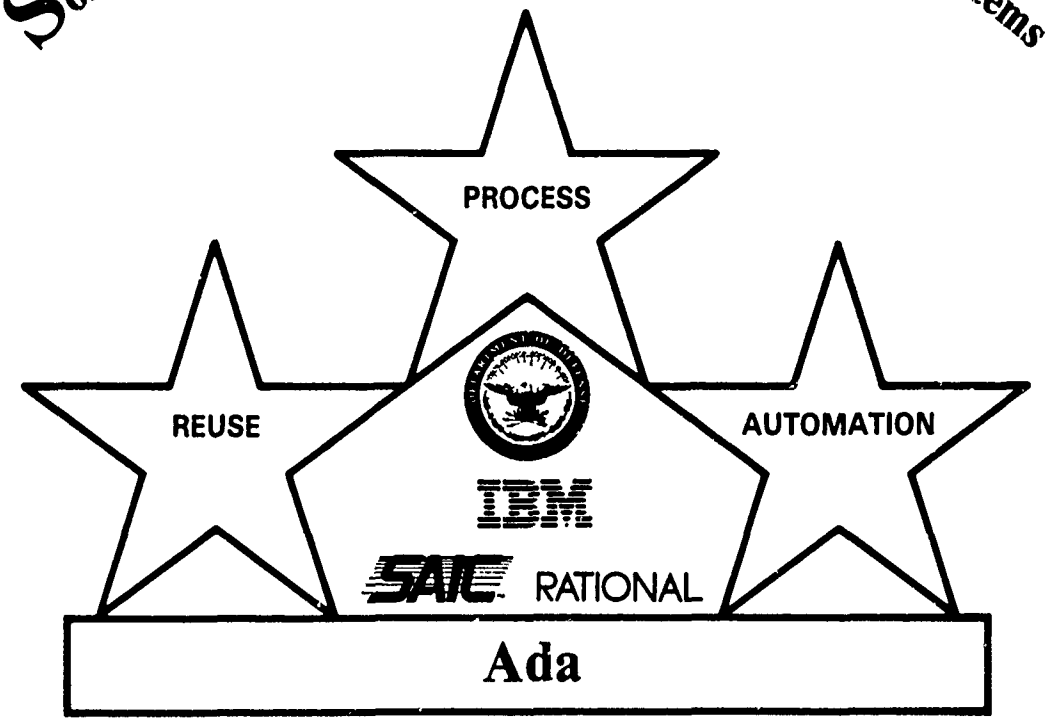
Document No. 0460-001
17 March 1989

Repository Guidelines & Standards

for the

Software Technology for Adaptable Reliable Systems



Contract No. F19628-88-D-0032
CDRL Sequence No. 0460

17 March 1989

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 17, 1989	Final

4. TITLE AND SUBTITLE
Repository Guidelines and Standards

5. FUNDING NUMBERS
C: F19628-88-D-0032

6. AUTHOR(S)
T. Ward

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
IBM Federal Sector Division
800 N. Frederick Avenue
Gaithersburg, MD 20879

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
CDRL Sequence No. 0460

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document recommends specific ways to collect, store, manage select, and retrieve reusable software. It defines guidelines and standards for the contents of the a software reuse library, and proposed tools to enforce these standards. It outlines the criteria that reuse library processes and guidelines must meet. Also addressed are the underlying reuse library concepts.

(CR)

14. SUBJECT TERMS
Software reuse, guidelines, standards, software reuse library, STARS

15. NUMBER OF PAGES
42

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Repository Guidelines & Standards
for the
Software Technology for Adaptable, Reliable Systems (STARS) Program

Contract No. F19628-88-D-0032

CDRL Sequence No. 0460

17 March 1989

Prepared for:

Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

IBM Systems Integration Division
18100 Frederick Pike
Gaithersburg, MD 20879

Repository Guidelines for the Software Technology for Adaptable, Reliable Systems (STARS) Program

Document Number CDRL Sequence No. 0460

IBM
Contract No. F19628-88-D-0032

Prepared for:

Electronic Systems Division (PKG-1)
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

IBM Systems Integration Division
18100 Frederick Pike
Gaithersburg, MD 20879

Contents

Figures

Tables

1. Scope

1.1 Identification

These Repository Guidelines (CDRL 00460) address STARS SOW subtasks 4.2, 5.6, and subtask Q12.1 (Guidelines) of Delivery Order Q12 (Guidelines and Repository).

1.2 Purpose

To support the STARS program objective of institutionalizing the reuse of software parts, a software engineering environment consisting of many tools as well as a large repository of reusable Ada software and documentation will be built.

This technical report defines the guidelines and standards for contents of the STARS Repository and for processes governing its use. It also proposes tools to process and enforce these standards. CDRL 0360, Reusability Guidelines (Q9), is a complimentary set of guidelines that cover the subject of developing or altering components such that they can be reused. This document covers the aspects of collecting, storing, managing, storing, selecting, and retrieving reusable components.

1.3 Introduction

This document establishes a solid base for the definition of guidelines and standards that meet the evolutionary needs of the STARS Program. It provides an overview of the underlying repository concepts and the issues to be addressed. It also outlines the criteria to be met by repository process and content guidelines. (KR)

1.4 Peer Review

The guidelines presented in this document will mature by incorporating the lessons learned during the STARS experience. Peer reviews are a vital part of this evolution. To this end, IBM actively solicits peer comments and recommendations.

1.5 Referenced Documents

1.5.1 Government Documents

Document Number	Title
(Q12TP)	Software Technology for Adaptable, Reliable Systems (STARS), Part II, Technical Proposal, Repository Task, and Delivery Order Proposal Q12: Guidelines and Repository, 5 January 1988.
F19628-88-R-0011	Software Technology for Adaptable, Reliable Systems (STARS), Competing Primes Lead Contract, Request for Proposal , 6 November 1987.
STARS CDRL 0370	Reusable Component Data Requirements and Standards , 31 January 1989.

1.5.2 Non-Government Documents

None

2. Overview

This section introduces key concepts and issues to be addressed by the Repository Guidelines. Information presented in this section establishes a context within which specific guidelines and standards can be developed. It will be used to identify the processes, information, and product standards needed for effective repository operation.

2.1 Definitions

The following terms introduced in this section are defined as they are used.

Repository System
An instance of the software engineering environment, including computer hardware, software tools and environment, standards and procedures, that together provide a complete repository capability.

Repository
An element of the software engineering environment in which software work products and information about them are stored.

Library
A subdivision of repository contents that forms a separately managed collection. A repository may contain several libraries.

Software Work Product
An intermediate or finished work product of software development or maintenance. Software work products include plans, specifications, designs, source and object code, test procedures, reports, demonstrations and other deliverable items.

Software work products can be either (a) a set of inputs to the repository or, after appropriate selection and retrieval, (b) a selected set of reusable component parts used to generate an output work product.

Qualification
The act of establishing that a software work product meets the necessary criteria for admission into a repository.

Component
A collection of related work products that are intended to be used as a consistent set of information.

Component Class
A set of software components sharing common attributes (e.g., stacks, queues, sequences, etc.)

Component Part
An essential element of a component (e.g.,design specifications, a formal document outline, minutes from a project review meeting, etc.).

2.2 Software Repositories

2.2.1 Repository System

2.2.1.1 Objectives

Consistent with program objectives (institutionalizing the reuse of software parts), the STARS repository/distribution system should provide the capabilities necessary to successfully establish and grow a resource of software work products for use in software development. It should provide a generally available means for the recovery and reuse of software work products.

Major capabilities to be achieved through the combined effects of hardware, software, and procedures include:

- Supply of software work products in reusable form.
- Management of a growing population of software work products, applicable to many projects.
- Distribution of shared software work products.
- Use of shared software work products in custom development.

Supply of new software work products to the repository involves their capture in a form that can be used by others. The supply process changes their status from single-use to shared use.

Management of the repository establishes and preserves the integrity of items entrusted to it. This addresses repository ownership issues such as entry and exit criteria, configuration control of changes, and maintenance.

Distribution includes repository organization to facilitate or control access, ordering, and logistics.

Use of repository contents involves search for (and access of) suitable items, and their integration into product development.

2.2.1.2 Constraints

These objectives should be achieved within the following constraints:

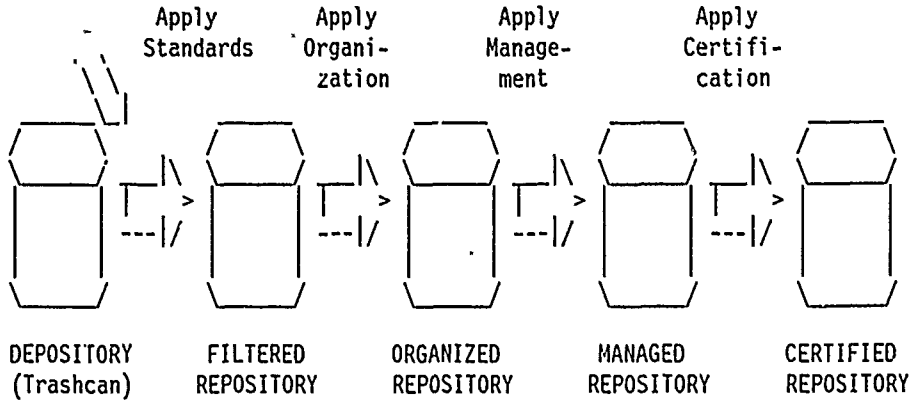
- Minimize the cost to the user of supply and use.
- Minimize the cost of maintenance to the owner.
- Protect the legal and data rights of both owners and users.

Minimizing the cost of supply and use is important to the success of the repository system. Economic barriers to the addition of new items could deter population growth; barriers to their use could prevent realization of the benefits of reuse. One cost consideration is the amount of effort necessary to convert a work product between project development standards to repository standards.

Minimizing the cost of ownership is important to the continued viability of the repository system. It includes costs of repository equipment, support organization, and maintenance of repository contents, and training.

Legal and data rights of all parties must be respected by the repository system. These issues include licensing, copyright protection, liability, security, and export controls.

2.2.2 Repository Classes



Spectrum of Repository Classes

Repositories of software work products can be graded according to how well their contents can be trusted. A progressive scale of repositories classed by confidence level is shown in the figure above.

Depository

Contents are deposited as-is, with minimal constraints. This repository class has the lowest level of confidence. It is mainly a storage facility. Characteristics of Depository include:

- Content is chronologically ordered
- Quality and usability of content are unpredictable
- The burden is on the user to find and evaluate potentially useful item
- Content is 'garbage in - garbage out'

Filtered Repository

Standards are applied to the form, contents and quality of repository submissions. This repository level establishes basic standards for the consistency and quality of its content, thereby reducing the effort required to evaluate items in the repository. Characteristics of a Filtered Repository include:

- "Junk" is eliminated:
  - redundant components
  - outdated versions
  - incomplete or non-working software
- Form and content are consistent
- This is the lowest repository level in which tools can process content

Organized Repository

Related contents are grouped together and organized for ease of access (e.g., catalog, taxonomy, or search attributes). Characteristics of an Organized Repository include:

- Tools are available for searching, ease of identification and access
- Related software work products can be grouped into components
- Relationships between components (such as dependency) can be expressed and automated
- Components can be grouped by component class

Managed Repository

Contents are managed through repository change controls, support agreements, established data rights, and management of commonality. Characteristics of a Managed Repository include:

- ownership is established for the repository
- maintenance responsibility for repository content is assigned
- content of the repository is controlled as a configuration baseline

- traceability can be established between a component-version and the products in which it is used through an "accounting record" which is automatically generated by processing data presented in Table 8 on page 32
- commonality of repository contents can be managed, so that new components are added to fill needs not covered by the existing set.

Certified Repository

Contents of the repository are validated by a certification process. Operations performed on its contents by the repository are also verified to ensure validity of the results. This class of repository has the highest level of confidence. Characteristics of a Certified Repository include:

- Results are highly trustable
- Results are suitable for use in critical applications
- Results are suitable for use in a manufacturing process.

Each level of this scale implies a progressive refinement of the repository system, including the level of tool functionality, the processes needed for repository operations, and content/information requirements for software work products managed by the repository. Therefore the set of standards and guidelines needed depends on the class of repository to be established.

2.2.3 Repository Structure

How the repository system is structured can have a significant effect on its usability and manageability. This section describes structural issues of organization within a repository. The geographic organization of the the repository system to meet the needs of distributed users is addressed in 2.2.4, "Repository Distribution" on page 10 . These structural issues will be used later to derive processes and information requirements.

The content of a repository can be organized into libraries that simplify its management and use. Common methods of organization are by:

- level of qualification attained
- scope of applicability
- access restrictions
- organizational structure

These methods can be used separately or in combination.

2.2.3.1 Qualification Levels

Repository content can be subdivided into a series of libraries that have increasingly rigorous qualifications for entry. This approach is similar to the repository classes defined above; specific qualification levels, as shown in the example in Figure 1 on page 7, should be selected based on the needs of repository users.

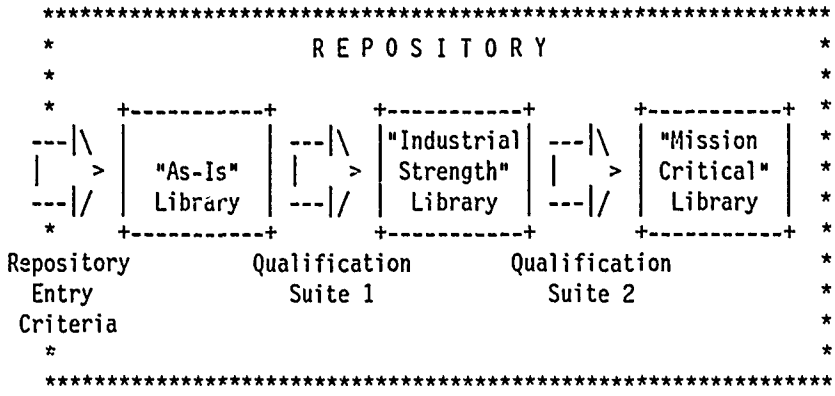


Figure 1. Sample Repository Organization by Qualification Level

2.2.3.2 Usage Hierarchy

Another method of organizing repository content into libraries is based on its scope of applicability. Libraries are organized into a hierarchy in which the highest library contains items of general use; libraries in lower tiers have increasing specialization resulting in narrower applicability. An example of this method of organization is shown in Figure 2 on page 8. In this method repository contents can be managed in specialized domain-specific libraries, permitting concentration of expertise.

2.2.3.3 Access Restriction

Another way to organize a repository is by the restrictions placed on use of repository contents.

In addition to encouraging wider use of items in the repository, an hierarchical method can be used to restrict their use. Access controls may apply due to several factors, including an item's ownership, export controls, or classification. A simplified example of this method of organization is shown in Figure 3 on page 9; the data rights categories shown are further discussed in 2.3.1, "Ownership" on page 12.

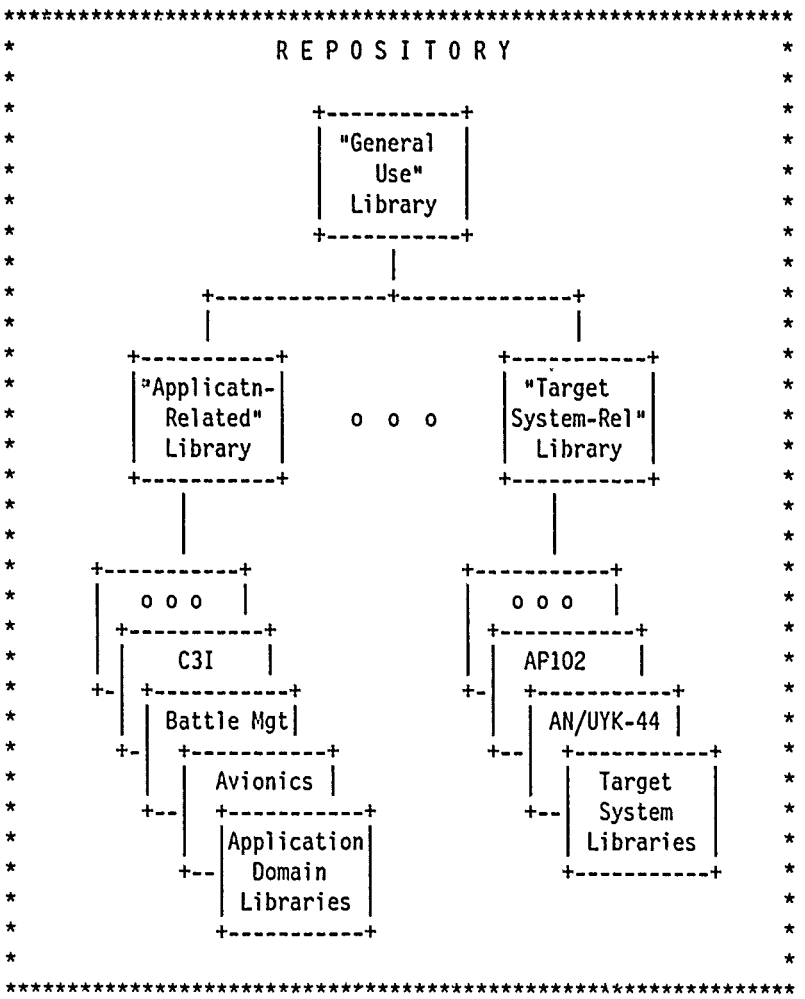


Figure 2. Sample Repository Organization by Applicability

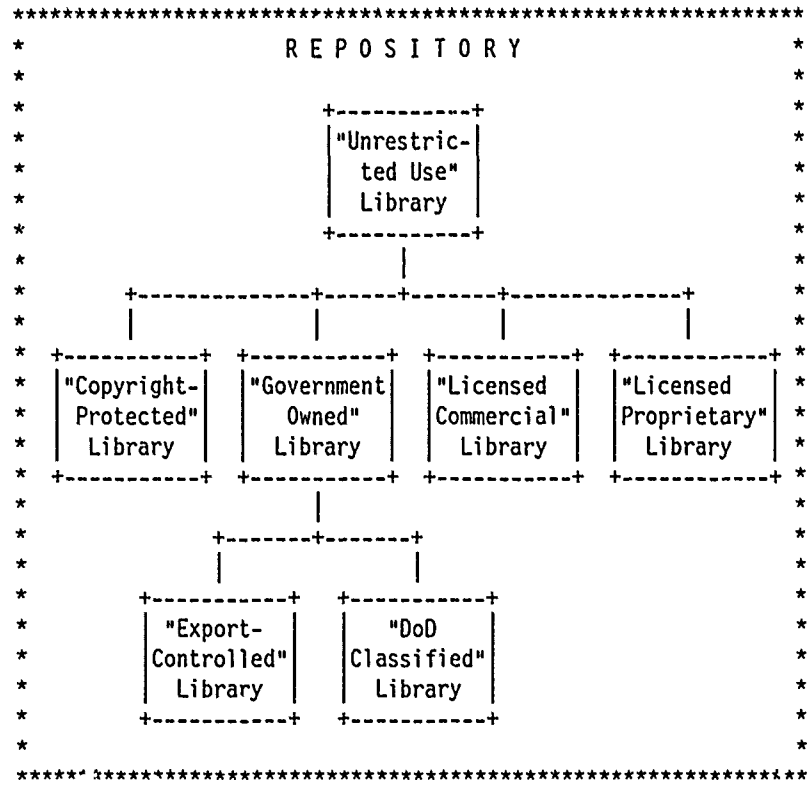


Figure 3. Sample Repository Organization by Access Restriction

An access-driven hierarchy is topped by a library containing items that can be used by anyone permitted to access the repository system. Lower levels isolate items by type of restriction, such as data rights or security classification. In this way, a user can access to only those libraries that he/she can legitimately use.

2.2.3.4 Organization Hierarchy

The hierarchical approach can also parallel the structure of the organization using the repository. In this method, management of each library can be tied to a corresponding organizational element.

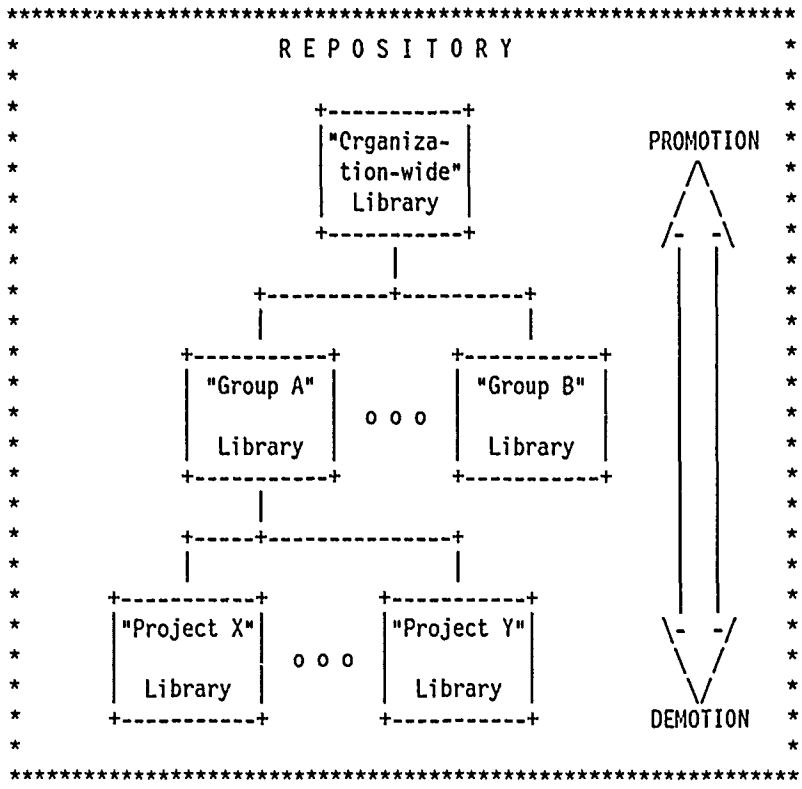


Figure 4. Sample Organization-based Repository Structure

An organization-wide library is at the top of this hierarchy, tiering down to individual project libraries. Promotion or demotion of items from one library to another can be used to reflect changing scope of use. This method, illustrated in Figure 4, provides a natural way to transition from reuse within a project to reuse across projects.

2.2.4 Repository Distribution

Wide-spread use of a repository system implies a means of access to the repository system by remote users. This section outlines the alternative approaches to this issue, and their effect on the set of processes needed for repository operation.

Primary means of addressing geographic distribution of repository users are:

- transfer of physical media
- network access
- replication of "shadow" repositories

Methods of access to the repository, distribution of items in the repository, and maintenance of repository items are summarized in Figure 5 on page 11. Each type of distribution is further described below.

2.2.4.1 Physical Media

Access to repository contents through physical media such as tapes, diskettes, or paper requires processes for requesting and for physically distributing repository 'parts lists' to prospective users. Distribution of repository contents through physical media requires processes for placing orders for repositories or items, inventory control, media production, and shipping. Maintenance of repository content involves central collection of updates and redistribution of changed content or parts lists.

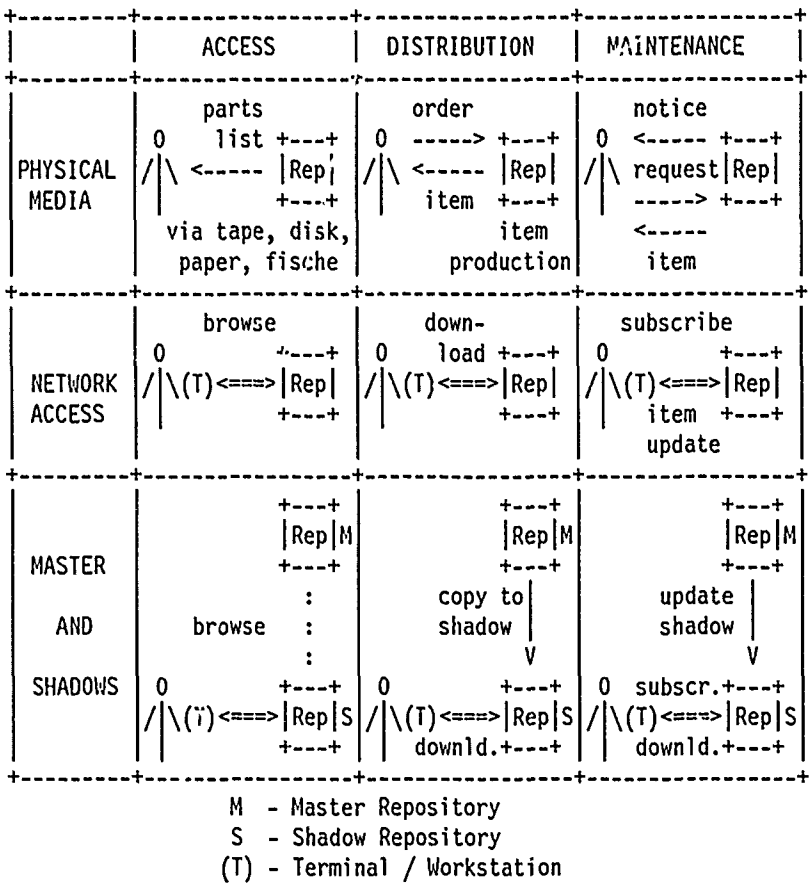


Figure 5. Summary of Repository Distribution Methods

2.2.4.2 Network Access

Access to repository content through wide area network links permits remote users to electronically search for, order, modify, or receive updates to items in the repository. Large requests requiring extensive network time may be better handled by shipment of physical media.

2.2.4.3 Replication

Local replication of a repository reduces end-user need for network transmission, and permits its use in electronically isolated environments such as secure systems. This is more important for libraries of items that at the low end of component scale, such as fragments or building blocks, so that a user can have have immediate access to them. However, redundancy creates the following maintenance issues which effect each repository process.

- avoiding divergence of content
- keeping all copies current

2.2.4.3.1 Master and Shadows: Divergence occurs when a local update is made to an item in a replicated repository. Designation of Master and Shadow copies are an effective device for avoiding divergence: one copy is identified as the 'master', in which all changes are made. Shadow copies are only updated from the master copy. This ensures that a single, authoritative source exists for repository content.

2.2.4.3.2 Shadow Update: Shadow copies can be kept current through periodic replacement, periodic update via physical media, or on-going transmission of changes from the master repository to all shadows over a network.

2.3 Shared-Use Software

Traditional software development produces a set of end-products that can be managed and controlled within the scope of a single project. This section addresses the issues of software which is shared across projects and organizations, thereby requiring a higher level of control.

The issues are divided into ownership (which includes legal and data rights, and finance) and operational (such as change control). While these considerations of shared-use software are not unique to a repository, they must be adequately addressed by it, through process guidelines, product standards, and repository system capabilities.

2.3.1 Ownership

2.3.1.1 Common Ownership Issues

Before an item can be entered into the repository, it is necessary to establish who owns it. A clear title is needed because ownership of an item brings certain rights, such as the ability to restrict and gain payment from its use. Ownership also includes responsibilities, such as financial liability for defective operation.

Among the responsibilities often included in ownership is maintenance and support of the item. Responsibility for maintenance needs to be established for items in the repository that are intended for long-term use. This may be carried out by the owner, or a separate agent associated with the repository.

2.3.1.2 Restrictions on Use

The owner of an item can place restrictions its use. They can set conditions on who can use the item, where it can be used, and how it can be used. Restrictions often take the form of license terms, or copyright restrictions for items which have been abandoned to the public domain.

2.3.1.2.1 Who Can Use It: The owner of an item can restrict WHO can use it. This begins with restriction to those who have agreed to the terms of a **license** permitting their use on an item. It may also include limitations on **disclosure** of the item, as in DoD classified or arms export controlled items. The concept may extend further to establish the subset of a users organization that is eligible to use item. For example, the right to use an item does not necessarily extend to subcontractors of a licensed user.

2.3.1.2.2 Where It Can be Used: The owner may also restrict WHERE an item can be used. Use may be licensed for use only at a specific **equipment** site or processor. Use may be restricted to a class of equipment, such as that which is provided as Government Furnished Equipment (GFE), or to equipment which is owned or operated by user.

The government also constrains **export** of software to other countries. Such transfers, where permitted, require a commercial export license. (Arms export controlled items, such as the Common Ada Missile Packages (CAMP), are subject to more severe restrictions against disclosure to foreign nationals. Since disclosure is main concern, regardless of location, this type of control is not primarily an export issue).

2.3.1.2.3 How It Can Be Used: The owner of an item can also restrict HOW it can be used. For example, embedded items like commercial products can be licensed with **restricted rights** that prohibit the user from subsequent application of that product in another system. Restrictions may also apply to **derivative products** such as translation of an item to other languages.

2.3.1.2.4 Packaging and Marking: Control of an item by its owner extends its repackaging by a user and inclusion in other products. Control over how such an item is marked is necessary to ensure that the owner's rights continue to be respected. Information requirements for distribution may be specified for source code, or for other documentation. These restrictions also apply to entry of an item into a repository.

2.3.1.3 Financial

2.3.1.3.1 Remuneration: Payment of usage fees to the owner is a common condition of licensed use of an item, and also can apply to public domain software protected by a copyright. Inclusion of these types of software in the repository requires capabilities to track usage, and procedures for transfer of funds from an item's user to its owner.

2.3.1.3.2 Liability: A significant issue to success of the repository system is the owner's financial liability for items entered into it. If the penalty for putting a defective item in the repository is too high, it can have a chilling effect on growth of the repository population. It is in both the government's and the owner's interest to limit the liability for an item in the repository.

2.3.1.4 Ownership Classes

There are four main classes of software ownership:

- Public Domain
- Government Owned
- Privately Owned, including Commercial and Proprietary subclasses.
- Mixed Ownership

2.3.1.4.1 Public Domain: In public domain software, the original owner has chosen to relinquish ownership by making the software available to the public without restriction. However, the owner may still protect it as intellectual property via copyright. User fees may also be requested.

This class of ownership is the least constrained, which creates a number of issues:

- How do you know the software has not been copyrighted? The absence of a copyright statement does prevent later exposure from someone coming 'out of the woodwork' to claim copyright. The issue is similar to title search for real estate.
- A copyright statement when included may not be sufficient to determine all the implications of copyright protection.
- Trustability of public domain software is varied. Determining whether it is of high enough quality to include in a product may be difficult.

2.3.1.4.2 Government Owned: Software which has been developed at least in part under government funding is generally classed as government-owned. There are issues for both the owner and the potential user.

For the government, a major ownership issue involves how to ensure continued maintenance and support for committed products. This is especially true when a change in contractors is made, resulting from a recompetition, or contractor default. In either case, enough information needs to be provided with the product to enable another group to assume maintenance responsibility for the software item. The repository should also be able to capture necessary support information.

A second issue is the potential for the government to be charged twice when an item that it originally paid for is included in another government product. This issue makes it necessary to provide visibility into what software has been reused in a product. It should also be noted that reuse is not free, a cost for evaluation, testing, and integration should be expected for reused software.

For the potential user, issues include:

- Determining what restrictions exist for use of an item, such as export controls.
- Use in of government-owned software in commercial products. Again, there is an implication that the government could pay twice if the resulting product were later sold to the government.

2.3.1.4.3 Commercial: Commercial software is privately owned software that is intended to be shared by many users through licensing. Issues for commercial software include:

- Product delivery may be restricted to object code only, limiting its usefulness where source code is needed (e.g., for portability).
- Use on government projects is a maintenance issue.
  - Cost of maintenance and support is shared over broader base of users.
  - Maintenance and support commitment from the vendor may not be adequate to support timely resolution of critical problems.
  - If a vendor should cease support of product, the government may need to ensure that it can assume maintenance.
- Termination provisions should also be considered from the perspective of shared software. Consider the effect on a global set of users if there is a license breach that nullifies their shared license to use a commercial product. How license breaches are addressed can have a significant effect on the risk to the user of loss of use.

2.3.1.4.4 Proprietary: Privately owned software that is not available to general users, but can be made available to specific user under special restrictions. Shared use of proprietary software must balance conflicting objectives of broader use and limited disclosure of proprietary information. Among the issues raised are:

- Use on government projects
- Cost recovery/usage fee
- Protection of proprietary software.

2.3.1.4.5 Mixed Ownership: Mixed government and private ownership is emerging as a way to balance the cost and risks of development against the government's need for the latest technology. Present DoD software policy reserves ownership to the government in all cases of mixed funding. Some government agencies, such as NASA, have a more liberal policy that encourages private commercialization of work done under contract.

2.3.2 Operations

Operation of a repository containing shared software centers on establishing and preserving the value of repository contents to its users. This encompasses entry and exit controls, quality assurance, and change control. Operation is also a shared responsibility in which repository users need to take an active role.

2.3.2.1 Entry and Exit Control

New additions to the repository should be in a form that facilitates sharing, and should meet minimum quality standards before entering the repository. And items currently in the repository should be removed to an archive when they are no longer useful. Criteria for entry into and exit from the repository need to be established, and implemented by procedures and tools, as shown in Figure 6 on page 15.

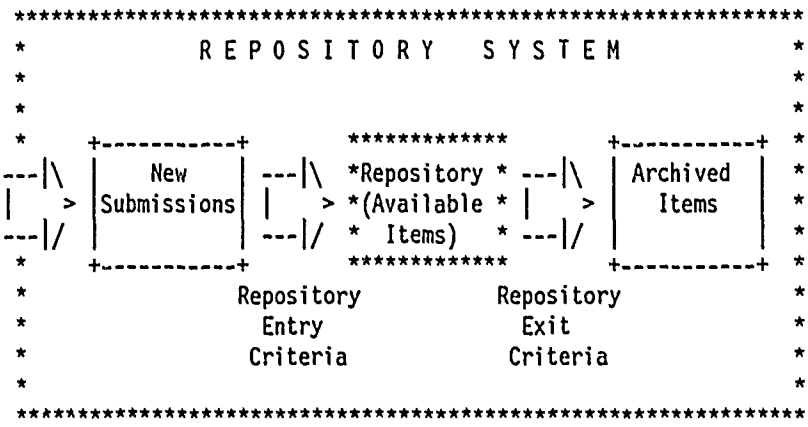


Figure 6. Repository Entry and Exit Control

2.3.2.2 Assurance

2.3.2.2.1 Quality Assurance: Quality Assurance must play a vital role in a repository system. Its task is to ensure that defined quality standards can be relied upon by repository users, thus providing a multiplicative pay-back in using projects.

2.3.2.2.2 Certification: A value-added service could be centrally provided for certification of items in the repository on request. It would perform standard types of tests and/or review the test and defect history of an item, to certify that it has attained a standard level of quality.

An example of this service is portability evaluation to a standard set of of Ada compilers and target systems. Items which have passed this test could be marked with the equivalent of a 'U/L Approved' label, and used in confidence within this scope of portability.

2.3.2.2.3 Decontamination: Assurance in a software repository system may also need to take on the tasks of prevention and detection of 'virus' programs, to prevent their spread into user systems. p.Prevention is based on traceability of repository entries, and on standards of construction. Unlike a bulletin board, or other informal exchange mechanism, the repository should require controlled registration of each item by its supplier. Traceability should be sufficient to identify the original source of an infective agent.

Detection of viral programs should be done by use of inspections and tests. Inspections, either manual or automated, look for characteristics of known types of viruses. This process can be aided by establishing standards for software construction that eliminate many of the techniques used to hide the program's purpose. Testing in an isolated machine can supplement inspection.

2.3.2.3 Change Control

A software item which is in shared use by several projects cannot be arbitrarily modified or replaced due to those dependencies. Alternative approaches applicable to shared software are backward compatibility for changes and use of independent repository and product baselines.

2.3.2.3.1 Backward Compatibility: Items in the repository which are in use should be subject to restriction on the changes that can be made so that existing users will not be impacted. A replaced version of an item should perform all functions of the previous version, and be of at least as high quality. Where functional alterations are necessary, a new item should be created with traceability from the old.

2.3.2.3.2 Baseline Control: The concept of a controlled repository configuration baseline with separate baselines for each using product provides a means to protect the interests of all users. Simply stated, the repository contains several versions of each item, only one of which is designated to be in the current

baseline set. Item users can work with either the current baseline or a back version, depending on their place in the development process. For example, if a product is in the field, replacement of an item with a new version may undesirable. Each using product needs to keep its own baseline of repository item versions in use, so that the project can control what changes to adopt, and when to adopt them.

2.3.2.4 Shared Responsibilities

Operation of a repository system is a shared responsibility that includes repository users. Each user has a responsibility to feed back the results of their experience with repository items. This can take many forms, including peer evaluation, defects identified, results of additional testing and portability evaluations, as well as 'how-to' and 'when-to' guidance. In this way, the repository becomes more than a service: it becomes the integrating element for community experience. Great synergism is possible by leveraging the experience of all participants.

2.4 Repository Contents

A primary goal of a repository system is to reduce redundant efforts in all of the life cycle phases of computer software development. To that end designers of repository systems should allow for and encourage (via ease of entry) the inclusion of all such repository contents. In addition the repository system design should include a convenient and consistent method for contents distribution to achieve the utmost in repository reusability.

The following paragraphs describe the rich characteristics of potential repository contents.

2.4.1 Range of Software Work Products

The phrase 'software work product' is not intended to restrict repository contents solely to computer software (ie. source code, object modules). There are a number of 'software work products' which contribute to the generation of a computer software product. Some of these enhancing, non-software work products include:

- Formal agreements between the customer and contractor
- System Requirements Review Comments
- Code Review Reports
- Configuration Management Plans
- Guidelines and Standards for software development

2.4.1.1 Static vs. Dynamic

The contents of a repository will be either static or dynamic. Static items are not altered once they have been entered into the repository (e.g., project presentations, minutes or reports of project reviews, test plans and procedures, etc.). Dynamic items would be updated as a project evolves (e.g., code being developed, operational procedures, schedules, etc.).

2.4.1.2 Transient vs Lasting Value

Besides having a static or dynamic quality, repository contents may also be categorized as having either transient or lasting value. Contents of transient values are ones which would be of little if any value after a set time period has passed. At that point the contents should be purged from the repository. Conversely, contents considered to have lasting value are those whose removal from the system must be carefully coordinated so as not to jeopardize the integrity of related products.

2.4.2 Software Components

Two broad characteristics which might describe the software components in a repository are size or scale and the level of abstraction. Examples of these characteristics are presented below.

fragments
concise sets of specialized code (e.g., a routine to process a stack).

packages
collections of related procedures, functions, and or data structures that represent a complete or skeleton solution for a definable portion of an application (e.g., a CASE template).

building blocks
encapsulated collection of routines that can provide complete function that is reusable in its "current" state

subsystems
integrated sets of lower level software components that perform specific subsystem-level functions (e.g., a communication subsystem)

systems
integrated sets of lower level software components and application unique software that performs specific system-level functions (e.g., an avionics system)

2.4.3 Software Component Parts

Some of the levels of abstraction that could apply to repository contents include:

- requirements specification
- high-level design
- detailed design
- source code
- object code

2.5 Reusability

The concept of reusability is a driving factor on the repository system. This section addresses reusability as it relates to the repository.

2.5.1 Reuse Classes

Software reusability is actually a complex of concepts, some of which have been in use for many years, and others which are now entering the state of the practice. These concepts can be divided into classes which address broad strategies for reuse.

The most fundamental division is between recovery (retrospective reuse) and planned (prospective) reuse. These classes are distinguished by **when** the decision to reuse is made. Both classes are in contrast to custom development, which starts each effort from scratch.

2.5.1.1 Recovery

Use of previously developed software as the starting point for new development is as old as the second square root routine. It is a distinct process from custom development because it must consider the suitability of existing software in the light of new product requirements. This evaluation process is not part of traditional methodologies.

Major recovery strategies are based on the type of change to be accommodated:

- rehost
- retarget, and
- salvage.

2.5.1.1.1 Rehost: Rehosting is modification of existing software to fit a new hardware or system software base. It focuses on revision of internal interfaces to fit the new environment, with minimal change in function. Rehosting may less expensive than custom development when the extent of modification is low. However, like all recovery strategies, it results in two products to maintain. The cost of rehosting must be borne again if a third system base is needed.

2.5.1.1.2 Retarget: Retargeting is the modification of existing software to fit a specific use or installation. It focuses on external interfaces to users and physical configurations of equipment. The underlying function is unchanged, but is altered in details. An example of retargeting is conversion of an avionics system between Air Force and Navy versions of an aircraft.

2.5.1.1.3 Salvage: The most basic form of recovery is salvage: extraction of potentially useful software from an existing system, modification to fit new a new use, and re-establishment of its integrity in its new environment. It is a strategy based on integration of elements from many sources.

2.5.1.2 Planned Reuse

Planned, or prospective, reuse makes the decision that an item is to be reused before it is developed. It is thus possible to build in features that enable the item to be more easily reused. Planned reuse increases the value of software by expanding its applicability in useful ways.

Major strategies of planned reuse parallel and improve upon those of recovery:

- Port
- Tailor, and
- Assemble from pre-built parts.

Relationships between these classes are shown in Figure 7 on page 19.

2.5.1.2.1 Port: Portability is the ability to easily move a software item from one hardware and system software environment to another. It is established by design considerations that isolate machine-dependent functions and utilize standard, virtual interfaces. Unlike rehosting, porting results in a single product to maintain, and is paid for once, in design of the product.

2.5.1.2.2 Tailor: Tailoring is a pre-planned modification of a product using a controlled customization interface that does not entail direct modification of source code. It enables a single software system to adapt to the needs specific installations, users, or missions. Tailorability is a designed in attribute based on a range of potential use. It also results in one base product to maintain.

2.5.1.2.3 Assemble: Assembly of a system from pre-built parts is a general strategy in which reusable software items are designed for integration, and products are defined at a component level. This strategy raises the level of software engineering to the level of reusable unit used.

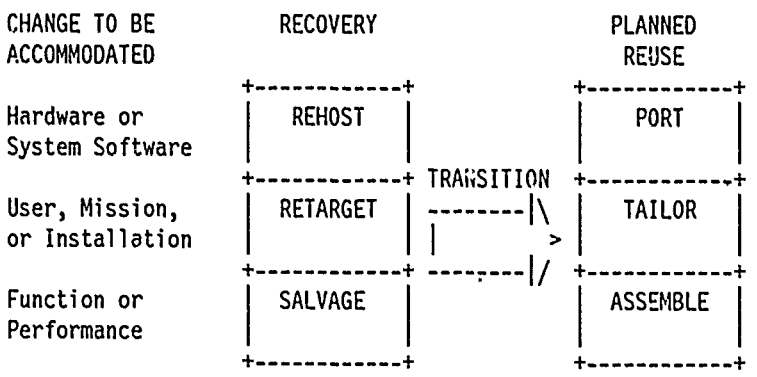


Figure 7. Relationship Between Reuse Classes and Strategies

2.5.2 Transferable Value

The value of reusable software goes beyond the source code. Software reuse is a transfer process involving transfer of:

- the product
- understanding of the product
- confidence that it will work well
- data rights to product use.

In order to communicate these things, they need to be collected along with the software product and retained in the repository. Sample items that contribute to each area are listed below.

- Product
  - requirements
  - designs
  - Ada code specifications
  - Ada code bodies
  - compile and link scripts
- Understanding
  - instructions for reuse (e.g., method of invocation, tailoring, etc).
  - user documentation (e.g., user guides, operations manuals, installation procedures, etc.)
- Confidence
  - test procedures and results
  - usage and defect history (this information should relate to the host or the target environmen)
- Data Rights

  Although frequently included in prologues, current deliverables do not adequately covers data rights transfer. At a minimum, this should be specified for:
  - Ada code specifications
  - Ada code bodies (which may have different rights from the corresponding specs).

3. Repository Process Guidelines

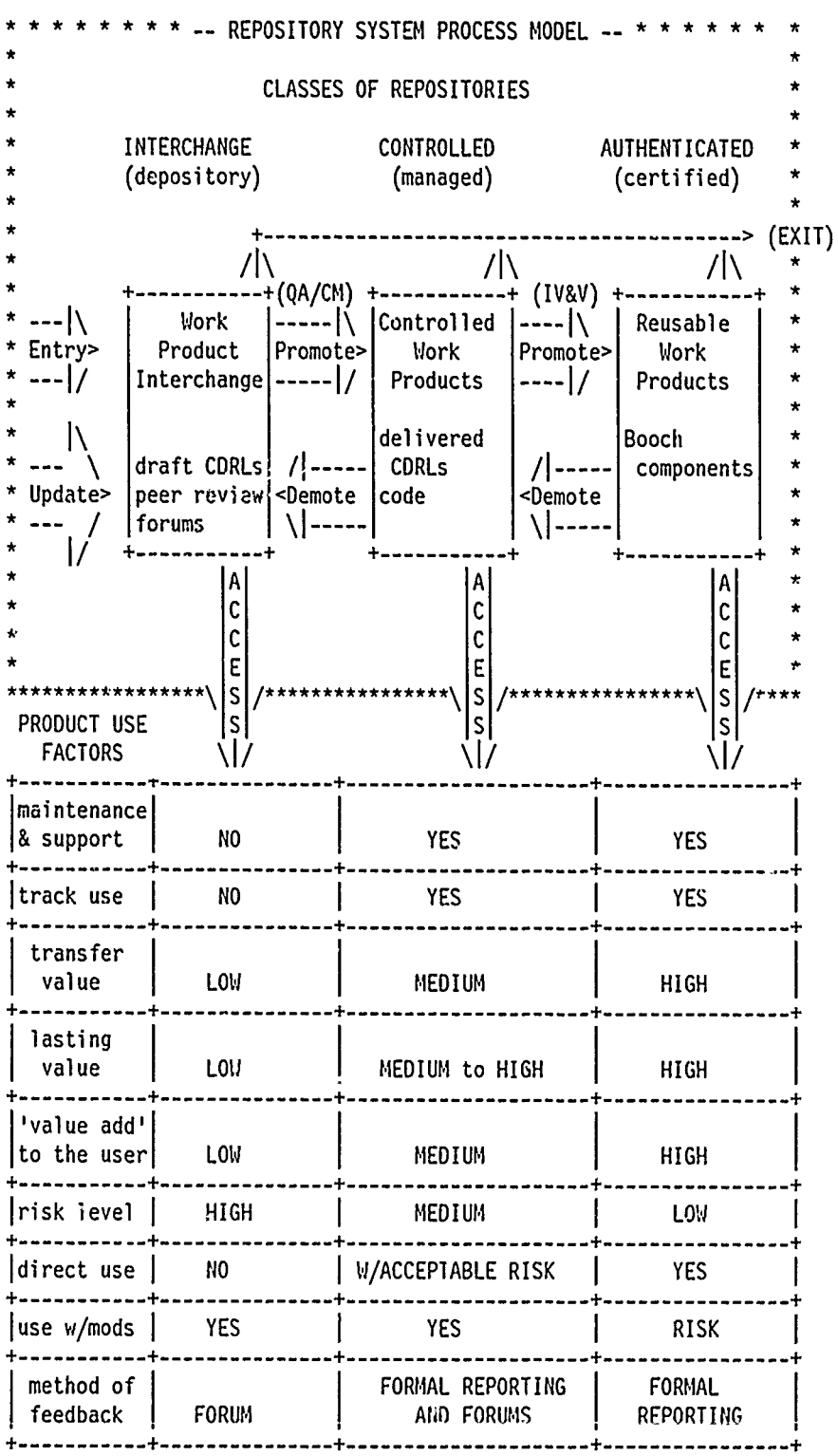
This section describes the guidelines (or criteria) which must be satisfied by the repository processes.

3.1 Processes

Six (6) of the more common repository processes are listed below.

1. entry (only into the "interchange" repository)
2. update (only in the "interchange" repository)
3. promotion
   a. from 'interchange' to 'controlled' repository using Quality Assurance and Configuration Management (QA/CM) procedures
   b. from 'controlled' to 'authenticated' repository using Independent Verification & Validation (IV&V) procedures)
4. demotion (from the 'controlled' and 'authenticated' repositories)
5. exit (from the any of the repositories)
6. access (product use)

The flow of these processes is illustrated in Figure 8 on page 21 .



PRODUCT USE FACTORS			
maintenance & support	NO	YES	YES
track use	NO	YES	YES
transfer value	LOW	MEDIUM	HIGH
lasting value	LOW	MEDIUM to HIGH	HIGH
'value add' to the user	LOW	MEDIUM	HIGH
risk level	HIGH	MEDIUM	LOW
direct use	NO	W/ACCEPTABLE RISK	YES
use w/mods	YES	YES	RISK
method of feedback	FORUM	FORMAL REPORTING AND FORUMS	FORMAL REPORTING

Figure 8. Repository System Process Flow

3.2 Process Criteria

A mapping of the process criteria to the process model shown in Figure 8 is presented in Table 1 on page 22 .

Table 1. Repository Process Criteria			
PROCESS(s) and associated criteria	**Class of Repository**		
	Interchange	**Controlled**	**Authenticated**
ENTRY,PROMOTE/UPDATE			
contributor organization (a)	X		
type of component (b)	X		
ACCESS/USE PRODUCT			
data rights (c)	X	X	X
Catalog entry in machine-readable form	X		
description (b)	X	X	X
restrictions (d)		X	X
format of component (c)	X	X	X
how to get the component (c)	X	X	X
classification (e)			X
usage fees (c)			X
third-party verification (b)			X
maintenance (a & f)			X
dependents (g)			X
DEMOTE/EXIT			
support discontinued	X	X	X
quality below standards		X	X
no users in past 'n' months	X	X	X
contributor request	X		
functionally obsolete	X	X	X

For details on criteria subscripted by a '(a)' refer to Table 2 on page 25 .

For details on criteria subscripted by a '(b)' refer to Table 3 on page 25 .

For details on criteria subscripted by a '(c)' refer to Table 4 on page 27 .

For details on criteria subscripted by a '(d)' refer to Table 5 on page 28 .

For details on criteria subscripted by a '(e)' refer to Table 6 on page 29 .

For details on criteria subscripted by a '(f)' refer to Table 7 on page 31 .

For details on criteria subscripted by a '(g)' refer to Table 8 on page 32 .

3.2.1 Use of repository content

Typical repository usage processes are listed below.

- search and access
- browse and print
- retrieval and distribution

3.2.1.1 Search and Access

The access to the components can be defined in any number of ways (e.g., ownership of a part, how the component is classified, etc.). The user community needs to know the access parameters of the repository contents so that they (the users) can request access authority to the contents which would be of greatest interest to them. Whenever access authority is granted to a user a two way communication link should also be established to facilitate feedback between the user and the repository owner/manager.

A fast and efficient access and search scheme must be available for a repository to remain viable. The design for this process should promote an implementation to group "like" components or component parts to maximize the efficiency of the browse process.

3.2.1.2 Browse and Print

The browse process allows the user to examine in greater detail portions of the repository contents located during the search process. While it may be desirable for users to have unlimited browse authority for items found during the access/search process, some portions of the repository may require restricted browse access because of data rights issues or other sensitive areas associated with the repository contents.

The ability to print repository contents available to the user in browse mode is a highly desirable feature for any repository system.

3.2.1.3 Retrieval and Distribution

The retrieval process allows users to obtain physical copies of selected repository contents. The speed of the access and search process will be dependent on the user's interface to the repository. The "appropriate"

The "appropriate" media (e.g., tapes, floppy disks, laser disks, etc.) for distributing repository contents will be dependent on a number of variables (e.g., amount of data involved, how soon the user needs the data, etc.).

4. Software Work Product Guidelines

The following sections will recommend software work product standards, guidelines and required deliverables for reuse repositories.

4.1 Information Collection Requirements

Information collection requirements, that is instructions for reuse, should be provided by the submitter along with other component description and general information as shown in Table 3 on page 25 and Table 4 on page 27 .

4.2 Work Products Form and Content

4.2.1 General Standards

The following standards are applicable to all types of data considered for inclusion into a repository. However, the extent to which these standards are applied is dependent on the class of the repository being established or managed (as described earlier in Section 2.2.2).

- Contributor Information
- Component Description
- General Instructions
- Restrictions
- Taxonomic Information
- Component Historical Data
- Component Relationship Data

Details of these standards are presented in the following subsections.

4.2.1.1 Contributor Information

The following information shall be provided by each contributor to the repository.

- Contributor
- Mailing Address
- Point-of-Contact
- Phone Number
- Sub-ID (Submitter or Contributor ID)

These items are defined in Table 2 on page 25 .

Table 2. Contributor Information Item Descriptions					
Item	**Definition**	**Source**	**Generatable**	**Searchable**	**Dynamic or Constant**
Contributor	Name of contributing, individual, company, organization or department	Sub-ID	N	Y	Dynamic
Mailing Address	Company Name, Street Address, City, State, Zip Code	Sub-ID	N	N	Dynamic
Point-of-Contact	Contributor or other individual affiliated with the contributor with working knowledge of the component or component part	Sub-ID	N	N	Dynamic
Phone Number	Phone number of point-of-contact or alternate technical lead.	Sub-ID	N	Y	Dynamic
Sub-ID	A code synonymous with the contributor, suitable as search-key for 'source'	Sub-ID	N	Y	Dynamic

4.2.1.2 Component Description

The following descriptive information shall be provided for each component or component part.

- Component (or Part) Name
- Component (or Part) State
- Type of Component
- Description
- Purpose
- Assumptions
- General Interface(s)
- Document References
- Version or Revision Number
- Third Party Verification
- Retention Period
- Sub-ID (Submitter or Contributor ID)

These items are defined in Table 3 .

Table 3 (Page 1 of 2). Component Description Item Descriptions					
Item	**Definition**	**Source**	**Generatable**	**Searchable**	**Dynamic or Constant**
Component (or part) Name	The name of the reusable component or part.	Sub-ID	N	Y	Constant
Component (or part) State	What is the state (new, repaired, derived) of the component or part?	Sub-ID	N	Y	Constant
Type of Component	Type of reusable component (e.g.,documentation, reports, source code, plans, meeting minutes, etc.)	Sub-ID	N	Y	Constant

Table 3 (Page 2 of 2). Component Description Item Descriptions					
Item	**Definition**	**Source**	**Generatable**	**Searchable**	**Dynamic or Constant**
Description	A general description of the component; could include description if inputs and outputs and limits (if any) on these.	Sub-ID	N	N	Dynamic
Purpose	A brief explanation of the components purpose or function. inputs and outputs and limits (if any) on these.	Sub-ID	N	N	Dynamic
Assumptions	A list of the assumptions about the behavior of the object (e.g., inputs are ordered, or type conversion is not needed for some inputs)	Sub-ID	N	N	Dynamic
General Interface(s)	An explanation if all formal parameters, along with a sample instantiation. This data required only for general components or parts.	Sub-ID	N	N	Dynamic
Document References	List referenced documents that are not stored as component parts.	Sub-ID	N	N	Dynamic
Version or Revision Number	If a component part is being described, indicate the current version or revision number.	Sub-ID	Y	N	Dynamic
Third Party Verification	For executable parts or components, indicate the third party that verified the quality of the item submitted.	Sub-ID	N	N	Dynamic
Retention Period	For 'transient value' parts or components, provide a retention period (e.g., 90-days).	Sub-ID	N	N	Dynamic
Sub-ID	A code synonymous with the contributor, suitable as search-key for 'source'	Sub-ID	N	Y	Constant

4.2.1.3 General Instructions

The following general instructions shall be provided for each component or component part.

- How to get the component
- Usage Fees
- Warranties
- Required Set-up
- Type of media
- Format
- Data Rights
- Disclaimers
- Exception or Abnormal Conditions

These items are defined in Table 4 on page 27 .

Table 4. General Instructions Item Descriptions					
Item	**Definition**	**Source**	**Generatable**	**Searchable**	**Dynamic or Constant**
How to get the component	Indicate procedure for obtaining a copy of the component or part	Sub-ID	N	N	Dynamic
Usage Fees	Specify if a fee is associated with the component or part.	Sub-ID	N	Y	Dynamic
Warranties	Identify any warranties that are applicable to the component or part	Sub-ID	N	Y	Dynamic
Required Set-up	Specify any pre-utilization set-up (e.g., calling special routines, unique initialization routines, installation dependent constants, etc.).	Sub-ID	N	N	Dynamic
Type of Media	Type of media where component or part is available (paper, magnetic, electronic, optical, etc.).	Sub-ID	N	N	Dynamic
Format	Format in which the component or part is stored (e.g., ASCII, EBCDIC)	Sub-ID	N	N	Dynamic
Data Rights	Copyright release or other proprietary release information	Sub-ID	N	N	Dynamic
Disclaimers	Indicate any warnings and or limitations associated with the part or component.	Sub-ID	N	N	Dynamic
Exception or Abnormal Conditions	Briefly explain each exception or abnormal condition. If there is a large number of these they should be documented in the source code.	Sub-ID	N	N	Dynamic

4.2.1.4 Restrictions

The following information on restrictions should be provided for each component or component part.

- Export Restrictions
- Security Restrictions
- Distribution Restrictions
- Environment-Related Restrictions
- Compiler-dependent Restrictions
- Implementation Restrictions
- Portability Restrictions

These items are defined in Table 5 on page 28 .

Table 5. Restrictions					
Item	**Definition**	**Source**	**Generatable**	**Searchable**	**Dynamic or Constant**
Export Restrictions	For example Export Administration regulations or International Traffic in Arms restrictions	Sub-ID	N	N	Dynamic
Security Restrictions	Self-explanatory	Sub-ID	N	N	Dynamic
Distribution Restrictions	Self-explanatory	Sub-ID	N	N	Dynamic
Environment-Related Restrictions	Self-explanatory	Sub-ID	N	N	Dynamic
Compiler-dependent Restrictions	Restrictions based on use of specific compilers (e.g.,ADA, COBOL, PL/1, etc.).	Sub-ID	N	N	Dynamic
Implementation Restrictions	Limitations that the implementation places on input.	Sub-ID	N	N	Dynamic
Portability Restrictions	Implementation dependencies, particularly on interrupts, operating system calls, representations, or specifications. Identify any compiler dependencies; if the part or component was written to take advantage of nonstandard packages supplied with a compiler, show how the code must be changed when the part or component is ported to a compiler without these nonstandard packages.	Sub-ID	N	N	Dynamic

4.2.1.5 Taxonomic Information

It is proposed that the following set of taxonomic information be stored for every data element entered into the repository.

- Element type
- Function(s)
- Object(s)
- Medium
- Application
- Functional Area
- Setting
- Element Structure
- Bounding Data
- Garbage Collection Data
- Iterator Information
- Concurrency Information
- Host Environment
- Target Environment

- Resources Required
- System Dependent Features
- Performance Characteristics
- Submitting Organization
- Language(s) Used
- Confidence Data
- Reusability Attributes

These items are defined in Table 6 .

Table 6 (Page 1 of 2). Taxonomic Item Descriptions

Item	Definition	Source	Generatable	Searchable	Dynamic or Constant
Element Type	Type of reusable element (e.g., software, documentation, plans, etc.).	Sub-ID	N	Y	Dynamic
Function(s)	The Function(s) performed by the part or component (e.g.,add, compare, create, delete, etc.).	Sub-ID	N	Y	Dynamic
Object(s)	The object(s) manipulated or implemented by the part or component (e.g.,characters, lines, digits).	Sub-ID	N	Y	Dynamic
Medium	The location where the action by the part or component takes place (e.g.,array, key-board, list, file, etc.).	Sub-ID	N	Y	Dynamic
Application	The application or system the component or part was designed for (e.g.,editor, DBMS, scheduler, etc.).	Sub-ID	N	Y	Dynamic
Functional Area	Application dependent activities which the component or part was designed for (e.g.,analysis, DB management,etc.).	Sub-ID	N	Y	Dynamic
Setting	The environment where the component or part is used (e.g.,air traffic control, point-of-entry sales, catalog sales etc.).	Sub-ID	N	Y	Dynamic
Element Structure	One example might be monolithic.	Sub-ID	N	Y	Dynamic
Bounding Data	Indicate if the size of the object implemented by the part or component is constant	Sub-ID	N	Y	Dynamic
Garbage Collection Data	Describe the garbage collection 'capabilities' of the component or part.	Sub-ID	N	Y	Dynamic
Iterator Information	Indicate if an iterator is provided (this is for object-oriented parts or components).	Sub-ID	N	Y	Dynamic
Concurrency Information	Is the component or part simultaneously sharable by several tasks? How is mutual exclusion provided?	Sub-ID	N	Y	Dynamic

Table 6 (Page 2 of 2). Taxonomic Item Descriptions					
Item	**Definition**	**Source**	**Generatable**	**Searchable**	**Dynamic or Constant**
Host Environment	Which computer and operating system were used to develop the component or part?	Sub-ID	N	Y	Dynamic
Target Environment	For which computer and operating system was the component or part developed?	Sub-ID	N	Y	Dynamic
Resources Required	Provide a list of all required resources; including time and space requirements.	Sub-ID	N	Y	Dynamic
System Dependent Features	Provide a list of all system dependencies; particularly those on interrupts, representation specifications or operating system calls.	Sub-ID	N	Y	Dynamic
Performance Characteristics	Discuss performance characteristics and issues such as general efficiency, tasking structure, and any performance trade-offs.	Sub-ID	N	N	Dynamic
Submitting Organization	The name and or Sub-ID if the submitting organization.	Sub-ID	N	Y	Dynamic
Language(s) Used	List the language(s) used in the development of the part or component.	Sub-ID	N	Y	Dynamic
Confidence Data	The confidence or trust level of the part or component.	Sub-ID	N	Y	Dynamic
Reusablity Attributes	Reusablity attributes which include domain applicability, error tolerance, documentation adequacy, problem complexity, clarity, etc.	Sub-ID	N	Y	Dynamic

4.2.1.6 Component Histroical Data

The following information reflecting 1) the current state of the component, 2) the change history, 3) known problems and open issues.

- Reason for Development
- Development Standards
- Application
- Submission Date
- Acceptance Date
- Version Number
- Current Status
- Author of Problem or Change
- Title of Problem
- Description of Problem
- Date Problem Reported
- Current Status of Problem
- Title of Change
- Description of Change

- Date Change Requested
- Current Status of Change
- Date Change Accepted
- Elements Affected
- Revision Number

These items are defined in Table 7 .

Table 7 (Page 1 of 2). Component Historical Data Item Descriptions

Item	Definition	Source	Generatable	Searchable	Dynamic or Constant
Reason for Development	State why the part or component was developed.	Sub-ID	N	N	Constant
Development Standards	Specify the development standards used while developing the part or component.	Sub-ID	N	N	Constant
Application	Briefly describe the application for which the part or component was originally designed.	Sub-ID	N	N	Constant
Submission Date	Indicate when the part or component was submitted for consideration for inclusion into the repository.	Sub-ID	Y	Y	Constant
Acceptance Date	Indicate when the part or component was initially incorporated into the repository.	Sub-ID	Y	Y	Constant
Version Number	Indicate the version number of the part or component when is was added to the repository.	Sub-ID	Y	N	Constant
Current Status	Indicate the current status of the part or component (e.g.,submitted, rejected, accepted, fixed, tested, verified, etc.).	Sub-ID	Y	Y	Dynamic
Author of Problem or Change	The individual identifying a problem or initiating a change.	Sub-ID	N	N	Constant
Title of Problem	Briefly describe the problem observed.	Sub-ID	N	N	Constant
Description of Problem	Describe the recommended changes; give rationale for changes.	Sub-ID	N	N	Constant
Date Problem Reported	The date when the request to revise the part or component was entered into the change control tracking facility.	Sub-ID	Y	Y	Constant
Current Status of Problem	Indicate the progress on problem resolution (e.g., analysis underway, probable 'fix' being tested, updated version submitted for consideration by change control board, etc.).	Sub-ID	Y	Y	Dynamic
Title of Change	Briefly describe why the part or component should be changed.	Sub-ID	N	N	Constant

Table 7 (Page 2 of 2). Component Historical Data Item Descriptions					
Item	**Definition**	**Source**	**Generatable**	**Searchable**	**Dynamic or Constant**
Description of Change	Describe change to the part or component and rationale for change	Sub-ID	N	N	Constant
Date Change Requested	The date when the request to update a part or component was entered into the change control tracking facility.	Sub-ID	Y	Y	Constant
Current Status of Change	Indicate the status of the requested change (e.g., analysis underway, probable implementation being tested, updated version submitted for consideration by change control board, etc.).	Sub-ID	Y	Y	Dynamic
Date Change Accepted	The date when the repository was updated to correct the problem or implement the change requested.	Sub-ID	Y	Y	Constant
Elements Affected	List all parts or components that are affected by this change.	Sub-ID	N	N	Constant
Revision Number	The revised version number resulting from the revision of this part or component.	Sub-ID	Y	N	Constant

4.2.1.7 Component Relationship Data

Component relationship tells of the relationships between parts and components in the repository. Such information might include

- Parent,
- Children,
- Siblings,
- Dependents,
- Dependent Upon,
- Called By,
- Calls.

These items are defined in Table 8.

Table 8 (Page 1 of 2). Component Relationship Data Item Descriptions					
Item	**Definition**	**Source**	**Generatable**	**Searchable**	**Dynamic or Constant**
Parent	The name and version (or revision) of the part or component from which the given part or component was derived.	Sub-ID	Y	Y	Dynamic

Table 8 (Page 2 of 2). Component Relationship Data Item Descriptions					
Item	**Definition**	**Source**	**Generatable**	**Searchable**	**Dynamic or Constant**
Children	The name(s) and version(s) (or revision(s)) of the part or component that were derived from the given part or component.	Sub-ID	Y	N	Dynamic
Siblings	The name(s) and version(s) (or revision(s)) of the sibling parts or components that were derived from the same parent as the given part or component.	Sub-ID	Y	N	Dynamic
Dependents	The name(s) of parts or components that depend on or with the given component.	Sub-ID	Y	N	Dynamic
Dependent Upon	The name(s) of part or component that the given part or component depends on or with.	Sub-ID	Y	N	Dynamic
Called By	The name(s) of parts or components that call the given part or component.	Sub-ID	Y	N	Dynamic
Calls	The name(s) of parts or components that are called by the given part or component.	Sub-ID	Y	N	Dynamic

4.2.2 Relationship to Development Phase

The time phasing, in which work products should be supplied to the repository, will be proposed with respect to the program objective of institutionalizing the reuse of software parts.

4.2.3 Relationships to Scale

Future updates to the STARS repository guidelines will identify the relationship between software work products and the scale of part represented, including:

- fragments
- packages
- building blocks
- subsystems
- systems

4.3 Detailed Work Product Descriptions

Specific guidelines for deliverable items from STARS technical tasks will be proposed in subsequent increments of the STARS program. Recommendations that consider repository system needs and the institutionalizing the reuse of software parts will be made for each CDRL type.

5. Supporting Tools

The following paragraphs will identify the requirements for support tools for:

- customization of repository installations
- differences in language requirements
- diverse software development methodologies
- enforcement of component standards (STARS task Q-9, CDRL No 0370)
- enforcement of repository standards. (STARS task Q-12, CDRL No 0460)

Repository support tools can be grouped as follows:

- general support
  - compilers
  - editors
  - listing utilities
  - text formatters
- reuse-specific support
  - domain analyzers ... process description items to determine the domain for the component (e.g., purpose, description, general interfaces, etc.)
  - dependency analyzers ... process taxonomic items to assess component dependencies (e.g., system dependent features, concurrency information)
  - standards checkers ... routines which process component specifications to insure that the component complies with all applicable standards
  - trust level analyzers ... routines which process component history data to evaluate the trust level of the component
  - measurement tools ... tools which measure the complexity or performance characteristics
  - certification tools ... routines which process restriction and history data to certify the component's usability

6. References

Document Number	Title
AFTL-TR-88-86	Common Ada Missile Packages (CAMP) - Phase 2 Final Report November 1988.
(Mccain85)	McCain,R., "A Software Development Methodology for Reusable Components", Hawaii International Conference on Systems Science, 1985
(Brisson87)	Brisson, L.M., Coglianese, L.H., Fritz, R.L., Kemerer, N.J., Luckey,P.H., "OWEGO SOFTWARE COMPONENT GUIDELINES", November 1987.
(Angier88)	Angier,R., "Experience With Large Scale Parts-Based Reuse on the Space Shuttle Program", Software Reusability Symposium National Institute for Software Quality and Productivity, April 14,1988
(STAR83)	"STAR (Space Transportation Automated Reconfiguration) System Definition and Data Flow", IBM FSD Houston, July 1983