Technical Report

CMU/SEI-88-TR-013
ESD-TR-88-014

Phase I
Testbed Description: Requirements and Selection Guidelines

Robert Holibaugh
J. M. Perry
L. A. Sun
September 1988

Technical Report
CMU/SEI-88-TR-013
ESD-TR-88-014
September 1988

Phase I
Testbed Description:
Requirements and Selection Guidelines

Robert Holibaugh
J. M. Perry
L. A. Sun

Application of Reusable Software Components Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Phase I Testbed Description: Requirements and Selection Guidelines

Abstract. The Application of Reusable Software Components Project has constructed a reuse testbed for conducting software engineering experiments in software reusability. The hardware and system software of the testbed will provide a distributed computing environment with file-server capability for the storage of reusable components and other artifacts of the development process. The testbed will support a variety of domain-independent and domain-dependent reusable components. The testbed will also support tools that foster reuse. This document contains the requirements and selection criteria for the testbed hardware, software, reusable resources, and an environment. For each of these four testbed resources, the requirements are grouped into five areas: support of experiments, maximization of experience and reusability, applicability to problem domains, acceleration of technology transition, and advancing the state of the practice in reuse.

1. Introduction

1.1. Background

The Application of Reusable Software Components Project at the Software Engineering Institute (SEI) is constructing a reuse testbed in which to conduct experiments. A reuse testbed is a computer system for conducting software engineering experiments in software reusability for the Reuse Project. The broad goals for this project include increasing experience with and understanding of software reusability, transition of reuse technologies between the SEI, the government, and the mission-critical computer resource (MCCR) industry (i.e., relevance to the MCCR community), and the advancement of the state of the practice in software reuse.

The purpose of these experiments is to:

- Quantify the risks and identify issues and problems when developing software from reusable components.
- Identify the information necessary to apply reusable components during systems and software development.

To investigate applying reusable components, the project must have software development facilities, reusable components, and experimental methods. To meet these first two needs, we will construct a testbed in which we will conduct experiments with affiliate partners. The requirements for the Reuse Project testbed are specified in this document. In summary, the requirements for the testbed address the support of experiments; the gaining of experience in reusability; technology transition between the SEI, government, and industry; reasonable

breadth of application to problem domains; and advancing the state of the practice in software reuse.

The testbed's hardware and system software will provide a distributed computing environment with file-server capability for the storage of reusable components and other artifacts of the development process. These requirements also address system and software test capabilities.

The software development environment of the testbed will support all phases of the development and have an integrated software engineering database for the storage and retrieval of reusable components and newly developed software engineering artifacts.

The testbed will support a variety of reusable components, domain-independent and domain-dependent. Example domains include data structure packages, missile navigation components and tools, avionic designs, user interface tools, and signal processing subsystems.

The testbed requirements are general, because reuse has a large intersection with environments, methods, and tools for software engineering. The guidelines for selecting the testbed hardware and software have been derived from the requirements. A testbed must be an open system into which one can integrate new tools and techniques as they become available. It should also be representative of the facilities available to MCCR contractors today.

1.2. Objective

The primary objective of the Reuse Project testbed is to provide the physical framework to acquire, use, and evaluate existing reusable components. The testbed will also support executing a set of carefully controlled reuse experiments on problems of realistic size and complexity and gaining experience and understanding of software reusability. The elements of the Reuse Testbed are hardware, software tools, a software development environment, a library for existing reuse parts, and a candidate development problem.

This paper will establish the guidelines for selecting the elements of the testbed that serve the above purposes. The selection guidelines are derived analytically from the requirements for the testbed. These requirements will be used to guide the selection of testbed elements, which may be heterogeneous. For example, one may select both Apollo and MicroVAX computers for the testbed. An Apollo network could be used to support requirements and design, and the MicroVAXes could be used for implementation and testing.

Definitions of terms used in this document are presented in Appendix A.

2. General Selection Guidelines

2.1. General Selection Methodology for the Testbed

The approach to selecting testbed elements is to expand on the requirements, establish guidelines for each requirement, and prioritize the requirements. Hardware, software, environment, and reusable component guidelines are used to evaluate candidates for addition to the testbed. The testbed may contain several equivalent elements, e.g., compilers. The availability of similar but different capabilities will support more and varied experiments than a single set of elements. This means that we may add tools or software to the testbed when we already have a tool or software that provides similar functionality. The guidelines will be applied to each element before it is added to the testbed. Thus, the guidelines will change over time as the needs of the project evolve.

To experiment with reusable components in a subsystem development, the testbed environment needs to contain a reasonably complete software development environment that supports all phases of the life cycle. Since the testbed will not remain static during the life of the project, the environment must be capable of evolving to support new requirements and additional experiments.

2.2. Emphasis on Reuse and Experiments

The selection guidelines will emphasize features of the hardware, the software, and the environment that satisfy the testbed requirements for the purpose of software reuse. Various tools will be considered for their ability to integrate with the environment, support reuse, and support the reuse experiments.

2.3. Selection Based on Subjective Judgment

Some features of the testbed cannot be evaluated objectively because there are no appropriate metrics or because objective evaluation would be extremely time-consuming or costly. Therefore, a certain amount of subjective professional judgment is required. These guidelines, consequently, make statements about features of the environment even if no reasonable measurement of that feature exists.

2.4. Selection Limitations

These guidelines deal with the testbed running a particular operating system on particular hardware. The problems of transportability or interoperability will only be addressed to a small degree. These guidelines stress basic software development areas that are important to experiments on reusability. The guidelines are concerned with the appropriateness of testbed elements in the general case. Some selections are more fully evaluated than other

selections. The depth of evaluation depends on the impact of the element being considered. There are no tests or measures to verify the correctness of each feature in the environment. The basic features of consideration are assumed to work to the extent that they have been verified or confirmed by other current SEI users or affiliates.

2.5. Basic Principles for Selection

The Reuse Project testbed is basically a set of computer facilities, integrated software tools, and procedures to:

1. support experiments in software development
2. maximize our experience in reusability
3. apply to problem domains
4. accelerate technology transition
5. advance the state of the practice in reuse

The key principles considered are listed below in priority order:

- Critical:
  - **Reliability**: Supporting software in the environment performs reliably. Users can be supported in a reliable manner on a given resource. In case of a system failure, the environment can be returned to a correct, consistent prior state.
  - **Usability**: The environment is easy to use. Further discussion is given in the section on user interface.
- High Priority:
  - **Ada-Based**: Since Ada will be used as the implementation language for the reuse experiments, the environment must support Ada development.
  - **Compatibility**: The environment has to be compatible with other environments used in the MCCR community to facilitate technology transition. The environment must be compatible with common operating systems used in the MCCR community. More details are included in the section on operating system compatibility.
- Average Priority:
  - **Extensibility**: New tools and new functions can be integrated. New reusable components can be stored in an integrated database.
  - **Adaptability**: It has portability. It can accommodate new methodologies, new languages, and new computers. An environment can be adapted to be domain-dependent or domain-independent.
  - **Completeness**: It supports all phases of the software development. The software tools should include analysis and design tools, editors, file managers, compilers, linkers, debuggers, etc.

The selection guidelines in the next chapter are derived from the five functional categories of requirements for the testbed listed above and from the above basic principles.

3. Selection Guidelines

3.1. Hardware Selection Guidelines for the Reuse Testbed

The hardware selection guidelines listed below are grouped with respect to the goal categories mentioned above. This grouping is helpful, because it provides some motivation for the guidelines, it reflects the manner in which the guidelines were derived, and it helped to identify guidelines to achieve completeness. Criteria are itemized below, prefixed by HG, and grouped by category. The guidelines are stated in qualitative form and are intended to provide information to project personnel so that they can formulate more quantitative specifications.

3.1.1. Support of Experiments

HG1 Distributed system - A distributed configuration of host and target machines is needed to support a distributed software engineering environment and to provide data storage for a large number of reusable components. The distributed environment provides the capability to integrate new and additional hardware as it becomes available. A distributed environment also provides the flexibility to change hardware and software when necessary. A distributed environment provides the potential for multiple experiments to be conducted simultaneously without the experiments interfering with each other.

HG2 Commonly available operating system - The hardware must have a commonly available operating system or environment in order that the experiments reflect MCCR practices. A common operating system will also be representative of the tools the MCCR community can acquire and install.

HG3 Networking and communications facilities - Communication capabilities are necessary for transporting with industry-acceptable performance source and object files between machines, for access to network servers, for utilization of existing systems at the SEI, and for distributed software development.

HG4 File server support - The testbed must have a file server for storing a potentially large number of reusable components and other artifacts of the development process. This file server can be one machine or a set of machines, or a virtual file system.

HG5 & HG6 Computing power and storage - The testbed configuration must provide, initially, computational and storage capabilities with industry-acceptable performance for two concurrent experiments each equivalent to a 7-person-year software development effort involving use of tools, libraries, and application software. Moreover, these capabilities must be expandable as necessary. In the future, it will have to support multiple, replicated, or blocked experiments, along with unobtrusive automatic collection of at least simple types of data.

HG7 Compatibility with systems/software test benches - The testbed hardware must

have potential, to be realized at some future time, for interfacing with a test bench for testing and debugging. Such a test bench may include simulators, console displays, debuggers and monitors, systems buses, data generators, and a target processor.

3.1.2. Maximization of Experience in Reusability

HG8 Run-selected software development environment - The testbed hardware must run the support software and software development (and maintenance) environment selected for use on the project.

3.1.3. Applicability to Problem Domains

HG9 Target processors - The testbed must have the potential, to be realized at some future time, for not only communicating with but for developing software for embedded processors such as 1750A.

3.1.4. Acceleration of Technology Transition

HG10 Representability - The hardware of the testbed must be commercially available and have sufficient representation in the MCCR community so as not to limit the extrapolation and impact of the conclusions of the software engineering experiments. For example, hardware such as Symbolics will not be available to the general software engineering community in the near future.

3.1.5. Advance the State of the Practice in Reuse

HG11 System/software engineering workstations - The hardware of the testbed must consist of state-of-the-art processors for which a quality Ada compiler is available. In addition, the testbed must utilize engineering workstations because many techniques for analysis and design exploit an interactive graphic man-machine interface.

3.2. Software Selection Guidelines for the Reuse Testbed

This section addresses software guidelines for the testbed as well as guidelines for the software itself. Software comprises support software, tools, and project utilities. Most of the software will be acquired. However, software may be developed if it is not available and if it is important for achieving the goals of the experiment. Since the software development environment is a major component of the testbed, the environment is addressed separately in another section. In fact, some of the software indicated below may be included as part of a software development environment. Selection guidelines are derived from and included with the testbed requirements. Guidelines are itemized below, prefixed by SG, and grouped by category.

3.2.1. Support of Experiments

SG1 Basic software tools - The following software tools are required:

- Language editor (language)
- Compiler
- Linker-loader
- Runtime system
- Communications software
- File-transfer software
- File manager
- Configuration manager

The following software is desirable:

- Word processing
- Design tools
- Requirements tools
- Security software
- Electronic mail

SG2 Data collection - Software must provide the capability to collect and support the analysis of data from the experiments. The collection mechanisms should not be intrusive and should use a minimum of the testbed's computer resources. The data collection mechanisms should work off of existing software and augment the collection with predefined forms or questionnaires. Once a software artifact is entered into the system, all changes must be recorded. The information collected with each change needs to be entered into a database for validation and later analysis. Furthermore, all relevant data for the change must also be collected and recorded, for example, the reason for the change, the type of change, the category of the change, other artifacts affected by the change.

SG3 Compatibility - The software must be compatible with the hardware, languages, operating systems, reusable components and environments utilized for experiments. The software should run on standard hardware and software systems. Software that is available on a wider variety of hardware, operating systems, and environments is preferred over software that can only be used in limited configurations.

SG4 Easy to use - Software must have a good user interface, be well documented, and reliable. The purpose of the software tools is to improve the productivity of software professionals who are developing systems.

The evaluation of a human interface must take into account the characteristics of the tools as well as the characteristics of the user employing the tools.

The following features need to be considered in an evaluation of the human interface:

1. Documentation
2. Simplicity, clarity of expression
3. Consistency and uniformity of commands
4. Ease of use, ease of learning
5. Appropriate use of input/output devices
6. Help facilities
7. Tutorial/training material
8. Error messages and diagnostic capabilities

3.2.2. Maximization of Experience in Reusability

SG5 Software engineering database - Software must include a database or library system for:

- storing a large set of heterogeneous components; these components could be code, test generators, designs (graphics, text, or structured items)
- classifying the components by life-cycle phase, type, domain, breadth-of-applicability, whether generative or passive, etc.
- locating a set of potential components based on the classification given above and the attribute (requirements) of the components
- relating components to each other as appropriate, to designs that they implement, to the appropriate testing procedures for the component, to the requirements for that component, and to the systems in which the component has been used
- retrieving reusable components from the database or library into the developers' workspace

SG6 Reusable tools - Software should include reusable tools, for example, analyzers that address reusability, template creators and manipulators, shells that embody a paradigm to locate, understand, and integrate components for reuse. Reuse tools may be classified into one of four support areas:

- locate
- understand
- modify
- integrate (e.g., composition, instantiation)

These areas are not mutually exclusive. For example, common Ada missile packages (CAMP) provide a tool that interacts with the user to select parameters for the instantiation of an Ada Kalman filter generic. This tool would be able to both modify and compose the reusable component with other components.

SG7 Lessons learned - Software must support the feedback of experience gained from the application of reusable resources so that experience will be captured. Software tools will be

included in the testbed if they can be used to demonstrate the applicability of reusable components, problems with reusable components, or the benefits of reusable components.

3.2.3. Applicability to Problem Domains

SG8 Relevance to MCCR domains - Software tools should be appropriate for MCCR problem domains and should include tools that support those domains. Software tools that are not designed for MCCR problem domains may be included if the concepts are applicable or the tools are generally useful. Some examples of useful tools include: source test analyzers, code generators, simulators, or prototyping tools.

3.2.4. Acceleration of Technology Transition

SG9 Restricted software - Software, or its underlying concepts, must be available for distribution. Software that is not available for distribution will be considered for the testbed if the concepts are valid and we can transition the concepts or principles to government or industry. Proprietary software will be considered for use in the testbed if it can be protected, necessary releases obtained, and the concepts transitioned.

3.2.5. Advance the State of the Practice in Reuse

SG10 Maturing technology - Software must advance the state of the practice of software reusability. Thus, requirement tools, design tools, and compiling systems that support automatic inclusion from and storage into a library are preferred over those that do not.

3.3. Reusable Resources Guidelines for the Reuse Testbed

The primary purpose of the project is directed at investigating software reusability. Thus, collections of reusable resources are central to the testbed. Guidelines for reusable resources are itemized below, prefixed by RG, and grouped by category.

3.3.1. Support of Experiments

RG1 Life-cycle support - The collections of reusable resources must include several types and have the potential of supporting several phases of the life cycle. Different types of reusable resources include requirements, goals and plans, designs, specifications, generics, schemas, transformations, standards, and tools. Moreover, these types must be applicable to the variety of software products, including designs, documents, codes, and tests that are produced during development.

RG2 Domain dependency - The testbed must contain both domain-dependent and domain-independent components, so that experiments can address the breadth of the domain-dependent parts or the power of the domain-independent parts. For example, the EVB parts are domain-independent; some of the CAMP components are domain-dependent.

RG3 Compatibility - The reusable components must be compatible with the software and environment used for the experiment(s). In particular, they must be compatible with Ada.

RG4 Adaptability - There must not be any strong restrictions on the use of the resources, e.g., any dependency on a non-common environment or methodology or operating system.

RG5 Reliability - The components must be reasonably reliable, given the state of the art in software reusability.

RG6 Facilitate experimental design - The application of the reusable components must involve a priori unanswered questions that will serve as the basis for experimental design and that will improve the chances for meaningful results.

3.3.2. Maximization of Experience in Reusability

RG7 Reusability - The components must have been designed for reuse or have attributes that make them reusable to a high degree.

RG8 Creators and users - It is desirable that the reusable components must come from several sources. Moreover, the users of the components in the experiment(s) must not be the ones who created the reusable components.

RG9 Ease of use - The reusable components must have adequate documentation and not require more than several days' training for their use.

RG10 Taxonomy - The reusable components must have an explicit taxonomy or fit into a given taxonomy.

RG11 Related components - Reusable components that have relationships actual or potential relationships with other reusable components are preferred over unrelated reusable components of small granularity.

RG12 Richness of a collection - The number of reusable components in the testbed must be sufficiently large to enable meaningful conclusions from the experiments. This guideline is based on the hypothesis that large collections are more likely to embody architectural and design concepts, methodologies for reuse, relational interactions among the components, or heuristics for reuse. A richer collection will allow for more flexibility in the design of the experiments.

RG13 Variety of types - Collections of reusable resources must include several types and granularity of components and have the potential for supporting several phases of the life cycle. Examples of different types of reusable components are generic requirements, specifications, architectures, designs, subsystems, procedures, and data structures. Further, in addition to the components per se, it is preferable that a collection has a paradigm associated with it for its use. For example, the reuse paradigm for a collection may be in the form of methods or tools for manipulating or combining the components.

3.3.3. Applicability to Problem Domains

RG14 Relevant to MCCR domains - Domain-dependent collections of reusable components must be relevant to MCCR problem domains for some of the experiments, such as the redevelopment experiment and, thus, be mappable to problem-domain taxonomies.

RG15 Domain-independent relevancy - Domain-independent components must have sufficient performance, including data precision, timing, space, and target properties appropriate for MCCR domains. The documentation for the components must reflect the constraints imposed by the specific domain.

RG16 Compatibility - The methodologies, environments, tools, operating systems, and languages of the reusable components must not invalidate interpretation results that are applicable to the MCCR community.

3.3.4. Acceleration of Technology Transition

RG17 Obstacles - Transition of reusable components from industry and government to the SEI must not require unreasonable time and cost. On the other hand, proprietary and confidentiality rights on the reusable components must not prevent the conclusions of the experiments from being disseminated to the MCCR community.

RG18 Maturity - The reusability technology embodied in a collection of reusable resources must be advanced enough and mature enough that it has the potential of improving productivity, and, if beneficial, could be put into practice now or in the near future.

3.3.5. Advance the State of the Practice in Reuse

RG19 New results - The reusable components must embody a technology that is sufficiently advanced and mature so as to increase the chances for new results that advance the state of the practice. The reusable components may embody a proven reusability technology that can be applied to another domain, a technology demonstrated on a small scale but not yet applied to large system development, or a new technology that is ready for transition to practice.

3.4. Environment Guidelines for the Reuse Testbed

The environment must provide the testbed with capabilities to do software development in MCCR problem domains, to support software development experiments, to gain experience in reuse, to advance the state of the practice in software reuse, and to support technology transition. Guidelines for selecting a suitable environment are itemized below, prefixed by EG, and grouped by category.

3.4.1. Support of Experiments

EG1 Multiple tools and methodologies - Tool integration is an important environment issue. Typically, a common database serves as the underlying element for the other components. A common user interface provides another level of integration. A stronger form of integration is one that is based on a common concept underlying the tools. Ideally, this is best achieved by imposing a particular methodology. Since a variety of reuse experiments will be performed over several years, an environment that supports multi-methodologies and does not attempt to impose a particular methodology is desirable.

EG2 Reuse tool and methodology testing - It must support the testing and evaluation of tools, methods and methodologies for reuse. Examples of reuse tools include analyzers for reusability, libraries of reusable components, template creators and manipulators. For example, as the libraries become available, evaluation and testing can be done for performance and accuracy.

EG3 Life-cycle development - The environment must support the entire software development life cycle. The software life cycle includes a requirements phase, design phase, implementation phase, test phase, installation and acceptance, operation and maintenance phase, and sometimes a retirement phase. Tools that directly support any of the phases must be acquired or developed if they are not available. The environment must provide the capability of integrating these tools.

EG4 Data collection - Once the goals of the data collection are clearly defined and the procedures to collect the data are specified, automatic or semiautomatic data collection support is possible. For example, data can be collected on the changes made to different versions of the development work. Therefore, various capabilities of the version control/configuration management system can be utilized to facilitate the data collection. A collection mechanism in the form of predefined reports or questionnaires can be easily incorporated into the environment. Experimental data include usage information, frequency of changes, types of errors, and man hours.

EG5 All aspects of development - The environment must support all the activities of technical development in terms of project management, configuration management, data management, documentation, review, quality assurance, and software development planning activities. Some basic capabilities that the environment must support are word processing, editing, and electronic mail. Additional important capabilities are:

1. **Data management**: This includes an integrated database with facilities for user authorization (security control measures), backup and recovery, etc., and tools or language for storage, retrieval, and modifications of software artifacts.
2. **Access control**: Users of the environment have different access rights to the programs in the database. Different versions of a program may be modifiable by selected groups for their specific purposes.
3. **Version control**: A large-scale project has many modules that undergo many improvements and modifications before they are finished. Bookkeeping facilities for version maintenance are necessary.

4. **Configuration management**: Tracking and recording facilities for maintaining the independence between independently constructed modules in large projects are necessary.
5. **Project management**: Capability for managing the scheduling needs, people, tasks, and software modules in large projects is essential.
6. **Project communication**: Good communication is essential for improving the productivity of software professionals. Facilities such as electronic mail, bulletin boards, and other means of communication should be made available for the project personnel.

EG6 Support experimental methodology - The environment must support an experimental methodology such as the one described by Weiss and Basili (1985) on experimentation in software engineering. In general, the environment supports experimental methodologies in the following aspects:

- comprehensible - easy to use and understandable
- repeatable - to ensure objectivity of the experiment. i.e., it yields the same or similar results when performed by different implementors
- extensible - additional user activities and accompanying experiments can be added easily; it should also accommodate different designs of an experiment in terms of multiple projects or team performers
- user-oriented - to focus on user activities so that suitable "patterns of use" of reusable components can be exploited and understood
- independent of experimental methods - ensure that the environment is not biased for or against any particular experimental method or approach
- analysis tools available - to support statistical data analysis or interpretation, for example, tools for regression or factor analysis are desirable

EG7 Database support - The environment needs to be a distributed software computing environment with access to a shared database. Internal communication channels (e.g., via Ethernet) as well as external communication channels (e.g., via DECnet) must be provided with a network environment. The database needs to be integrated with the environment and the operating system to ensure efficient access and usage. There should be enough data storage to host the tools, the support software, the reusable components, and all project-related software development artifacts.

In addition, the selection guidelines for the environment in relation to its database also include:

- User interface - appropriate type of query language that is easy to learn, and without detailed knowledge of the data structure
- Tool interface - uniform interface between the database and tools for retrieval and storage
- Capacity - extensible to aid integration of tools, allow addition of new tools and functionality, and new data items and relationships
- Functional capability - support storage, retrieval and modification of all project data through the life-cycle development

- Security - protect SEI and company proprietary information, and export control in a multi-user arrangement through user authorization or controlled access
- Integrity - provide automatic backup and restoration, a frequent "autosave" while users are performing editing activities, and an enforcement of various types of constraints on data changes

3.4.2. Maximization of Experience in Reusability

EG8 New tools and methodologies in reusability - At the current time, there are few tools or methods that directly support reuse. As tools, methods, and techniques become available, the environment must be able to integrate them. The environment must support new tools that help the developer locate, access, modify, and compose the reusable components into the system under development. Also, the environment must provide the capability to develop new tools for reusability.

EG9 Compatibility - When a testbed environment is built on top of an existing operating system, it is important to ensure that the interface is smooth. If the interface is not smooth, a duplication of effort and loss of performance and productivity will result. The environment must be compatible with the hardware, the operating system, reusable components, and software that are utilized for experiments.

3.4.3. Applicability to Problem Domains

EG10 Multiple MCCR problem domains - The environment must support software development for real-time embedded systems involving signal processing, navigation, and command, control, and communication.

EG11 Integrated tool sets - In a problem domain such as signal processing, where the reusability technology has matured, a set of integrated tools may have been developed for reusing components. The environment should support incorporation of such well-integrated tool sets.

EG12 Reuse taxonomy - To locate the reusable components during development, criteria for evaluating and classifying components must be acquired or defined. The environment needs to support multiple taxonomies applicable to various problem domains in which the experiments are being performed.

3.4.4. Acceleration of Technology Transition

EG13 MCCR representability - The environment should be commercially available or be representative of the MCCR community so that the experimental results are applicable. If the environment is very specialized, then the results will not be applicable to the DoD or the MCCR community, and the environment will not support the transition of technology.

3.4.5. Advance the State of the Practice in Reuse

EG14 Storage and retrieval of components - The environment should not be so restrictive that it rules out mechanisms that are very useful, for example, mechanisms for storing, retrieving, indexing, and browsing libraries of reusable components.

EG15 Reuse technology - Software reuse may involve many different problems because reuse may take many forms. Available data and experience in the reuse experiments will help to solve problems, to capitalize on the different forms and methods, and to contribute to their acceptance and practice. The environment must support investigations into these problems.

EG16 Problem-domain integration - The environment should provide support for integrating reusable software artifacts with those of the current development.

EG17 Promote reusable designs - Reusable components belong to one of two classes: building block components, such as the EVB parts, or architectural components, such as the embedded machine signal processing (EMSP) common operational signal processing (ECOS) data flows or GTE structured techniques for engineering projects (STEP) generic architecture. In the second class, there are relational interactions among the components that influence design and constitute reusable design concepts. Reusability experience can be obtained by explicitly formalizing and recognizing these design concepts. The environment must be open and unrestrictive to support this effort. Environments that have features of class structure, encapsulation, and modularity also support lower-level reusable designs.

EG18 Support enforcement of standards - To promote reusability, standardization is necessary. As standards for reuse in software development evolve, the environment should be capable of actively supporting and enforcing them.

References

Balzer, Robert. Evolution as a New Basis for Reusability, *ITT Workshop on Reusability in Programming*, Sep 1983.

Berard, Ed. Creating Reusable Ada Software, in *National Conference Proceedings: Software Reusability and Maintainability*, Sep 10-11, 1986.

Booch, Grady. *Software Components with Ada*, Benjamin/Cummings, 1987.

Cheatham, Thomas. Reusability Through Program Transformation, *IEEE Trans. on Software Eng.*, Vol SE-10, No. 5, Sep 1984, pp 589 - 594.

DoD-STD-2167.

Gargaro, A. and Pappas, Frank. Ada Reusability Study, Computer Science Corporation, Moorestown, NJ, Aug 1986.

McCain, Ron. Reusable Software Component Engineering, IBM, Houston, TX, Jul 1986.

McNicholls, Dan, Palmer, Constance, et al. Common Ada Missile Program, McDonnell Douglas, St. Louis, MO, Air Force Armament Laboratory, Eglin AFB, FL, AFATL-TR-85-93, May 1986.

Neighbors, James. The Draco Approach to Constructing Software from Reusable Components, *IEEE Trans. on Software Eng.*, Vol SE-10, No. 5, Sep 1984, pp 564 - 574.

Nise, Norman S. and Giffin, Chuck. The Design for Reusable Software Commonality, Rockwell International, Downey, CA, Mar 1984.

Presson, Ed, Tsai, J., Bowen, T., Post, J., and Schmidt, R. Software Interoperability and Reusability Guidebook for Software Quality Measurement, Boeing Aerospace Co., Seattle, WA, Rome Air Development Center (RADC), Griffiss AFB, NY, RADC-TR-83-174, July 1983.

Prieto-Diaz, Ruben. A Software Classification Scheme, GTE Computer Science Labs, Oct 1987.

Roder, John. Phoenix Architecture, *Sig Design Automation,* June 1978, pp 18-22.

St. Dennis, R. A Guidebook for Writing Reusable Source Code in Ada, Honeywell Computer Science Center, Golden Valley, MI, May 1986.

STARS Application Workshop, Naval Research Laboratory, San Diego, CA, Sep 1986.

Weiss, D. and Basili, V. Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory, *IEEE Trans. on Software Eng.*, Vol SE-11, No. 2, Feb 1985, pp 157 - 168.

Appendix A: Glossary

accessibility - The extent to which software facilitates selective use or maintenance of its components.

architecture - See Program architecture, system architecture.

architectural design - The process of defining a collection of hardware and software components and their interfaces to establish a framework for the development of a computer system.

automated design tool - A software tool that aids in the synthesis, analysis, modeling, or documentation of a software design. Examples include simulators, analytic aids, design representation processors, and documentation generators.

compatibility - The ability of two or more systems to exchange information. Compare with interoperability.

component - Logical part of a system or program.

integration - The process of combining software elements, hardware elements, or both into an overall system.

interoperability - The ability of two or more systems to exchange information and to mutually use the information that has been exchanged. Compare with compatibility.

level - The degree of subordination of an item in a hierarchical arrangement.

methodology (development methodology) - A systematic approach to the creation of software.

modularity - The extent to which software is composed of discrete components such that a change to one component has minimal impact on other components.

module - A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading, for example, the input to, or output from, an assembler, compiler, linkage editor, or executive routine.

portability - The ease with which software can be transferred from one computer system or environment to another.

program architecture - The structure and relationships among the components of a computer program. The program architecture may also include the program's interface with its operations environment.

reusability - The extent to which a component can be used in multiple applications.

reuse - The practice of applying the principles and tools of computer science and software engineering to systematically acquire software development experience, represent it, and use it to improve the development process and products.

software engineering - The systematic approach to the development, operation, maintenance, and retirement of software.

software library - A controlled collection of software and related documentation designed to aid in software development, (re)use, or maintenance. Types include software development library, master library, production library, program library, and software repository.

standards enforcer - A software tool that determines whether prescribed development standards have been followed. The standards may include module size, module structure, commenting conventions, use of certain statement forms, and documentation conventions.

system architecture - The structure and relationship among the components of a system. The system architecture may also include the system's interface with its operational environment.

system design - The process of defining the hardware and software architecture, components, modules, interfaces, and data for a system to satisfy specified system requirements.

target machine - The computer on which a program is intended to run.

testbed - A test environment containing the hardware, instrumentation tools, simulators, and other support software necessary for testing a system or system component.

Table of Contents