

AD-A258 255



Special Report
CMU/SEI-92-SR-4

2



Carnegie-Mellon University
Software Engineering Institute

DTIC
ELECTE
DEC 10 1992
S C D

A Reuse-Based Software Development Methodology

K. C. Kang
S. Cohen
R. Holibaugh
J. Perry
A. S. Peterson

May 1992

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

127100

92-31228



3488

92 12 09 056

The following statement of assurance is more than a statement required to comply with the federal law. This is a sincere statement by the university to assure that all people are included in the diversity which makes Carnegie Mellon an exciting place. Carnegie Mellon wishes to include people without regard to race, color, national origin, sex, handicap, religion, creed, ancestry, belief, age, veteran status or sexual orientation.

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admissions and employment on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders. In addition, Carnegie Mellon does not discriminate in admissions and employment on the basis of religion, creed, ancestry, belief, age, veteran status or sexual orientation in violation of any federal, state, or local laws or executive orders. Inquiries concerning application of this policy should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2054 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2054.

Special Report

CMU/SEI-92-SR-4

May 1992

A Reuse-Based Software Development Methodology



Kyo C. Kang

Sholom Cohen

Robert Holibaugh

James Perry

A. Spencer Peterson

Application of Revisable Software Components Project

**Approved for public release.
Distribution unlimited.**

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

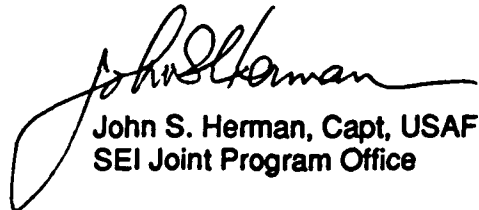
SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER



John S. Herman, Capt, USAF
SEI Joint Program Office

The Software Engineering Institute is sponsored by the U.S. Department of Defense.

This report was funded by the U.S. Department of Defense.

Copyright © 1992 by Carnegie Mellon University.

This document is available through the Defense Technical Information Center. DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145.

Copies of this document are also available through the National Technical Information Service. For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161.

Copies of this document are also available from Research Access, Inc., 3400 Forbes Avenue, Suite 302, Pittsburgh, PA 15213.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Table of Contents

1	Introduction	1
2	A Reuse Activity Model	3
3	System Requirements Analysis	5
4	Software Requirements Analysis	7
5	Design	9
6	Coding, Integration, and Testing	11
7	Expected Results	13
Appendix A Methodology Task List		15
A.1	System Requirements Analysis	15
A.2	Software Requirements Analysis	18
A.3	Preliminary Design	20
A.4	Detailed Design	22
A.5	Coding and CSU Testing	24
A.6	CSC Integration and Testing	24
A.7	System Integration and Testing	25
References		27

Accession For	
NTIS GR461	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist Special	
A-1	



A Reuse-Based Software Development Methodology

Abstract: Software has been reused in applications development ever since programming started. However, the reuse practices have mostly been ad hoc, and the potential benefits of reuse have never been fully realized. Most of the available software development methodologies do not explicitly identify reuse activities. The Application of Reusable Software Components Project of the Software Engineering Institute is developing a reuse-based software development methodology, and the current direction and the progress of the methodology work are discussed in this paper.

The methodology is based on the life cycle model in DoD-STD-2167A with refinement of each phase to identify reuse activities. The reuse activities that are common across the life cycle phases are identified as: 1) studying the problem and available solutions to the problem and developing a reuse plan or strategy, 2) identifying a solution structure for the problem following the reuse plan, 3) reconfiguring the solution structure to improve reuse at the next phase, 4) acquiring, instantiating, and/or modifying existing reusable components, 5) integrating the reused and any newly developed components into the products for the phase, and 6) evaluating the products. These activities are used as the base model for defining the specific activities at each phase of the life cycle.

This methodology focuses more on identification and application of reusable resources than on construction of reusable resources, and some enhancements in the construction aspect might be necessary to make it more complete.

This methodology has never been applied; it will be used in an application redevelopment experiment and then will be improved based on our experience.

1 Introduction

Software reuse is not a new development concept. For instance, we have been reusing software libraries for decades. However, reuse has mostly been limited to code and has generally been practiced in an ad hoc way. The impacts of reuse on software development and the potential benefits and risks of reusing other life cycle products (such as requirements and design) have not been fully understood.

The Application of Reusable Components Project of the Software Engineering Institute is conducting a reuse experiment through the redevelopment of an existing system. Through the experiment we want to gain experience with available reuse technology, measure and understand the impact of software reuse on products and the development process, understand what and how we can reuse at each phase of the life cycle, and collect and accumulate data for evaluating technologies in the future.

One of the experiment tasks is the creation of a reuse-based software development methodology. The purpose of the methodology task is to identify the development activities and to define the order in which the identified activities are to be carried out during the software development.

There are many reuse-based methodologies [1, 3, 4, 5]. However, most of these methodologies are based on specific methods or techniques and reusable components must be developed using the methods or techniques before they can be utilized. We are not interested in developing a new technology; we are interested in utilizing the available technologies. Therefore, we focus on how to identify the technologies appropriate for a given project and then integrate them into the development process and products.

We have chosen the DoD-STD-2167A [2] as the basis for our methodology and have followed the guidance of the "STARS Reusability Guidebook". The rationale behind our choice are:

1. It is general and can accommodate various development methods and techniques.
2. DoD mandates use of the standard in DoD contracts. Our methodology that is compatible with the standard will identify a reuse approach that DoD contractors might take advantage of. Through experiment, we might also identify problems that might be with the standard in promoting reuse.

A development methodology for a given project should identify:

1. The development activities,
2. The artifacts to be produced,
3. The available resources, and
4. A development plan relating the activities, artifacts, and development resources in terms of schedules and milestones.

The methodology in this paper is a general one. Therefore, a specific methodology may need to be created by instantiating the general methodology in this paper considering the level of experience with the application domain and the project constraints (e.g, available resources).

The methodology has not been applied, and is therefore not complete. By applying the methodology during the experiment, we want to understand what and how we can reuse. Based on our experience, we will improve the methodology. A detailed methodology task list is included in the Appendix.

2 A Reuse Activity Model

The methodology refines the life cycle model given in DoD-STD-2167A by identifying reuse activities applicable to each phase. (DoD-STD-2167A identifies the life cycle phases as system concepts development, system requirements analysis, software requirements analysis, preliminary design, detailed design, coding and unit testing, software component integration and testing, and system integration and testing.) In defining activities for each phase, a generic reuse activity model is used as a base model. The generic model is discussed in this section.

Reuse is an act of synthesizing a solution to a problem based on predefined solutions to sub-problems. The reuse activity is divided into six major steps performed at each phase in preparation for the next phase. These steps are:

1. Developing a reuse plan or strategy after studying the problem and available solutions to the problem,
2. Identifying a solution structure for the problem following the reuse plan or strategy,
3. Reconfiguring the solution structure to improve the possibility of using predefined components available at the next phase,
4. Acquiring, instantiating, and modifying predefined components,
5. Integrating the components into the products for this phase, and
6. Evaluating the products.

The major tasks under the first step are to understand the problem, build up the knowledge of the predefined solutions, and develop a plan or strategy of how to effectively apply the solutions to solve the problem. Alternative ways of structuring the problem are investigated, and rules and guidelines of applying the predefined solutions are developed.

The next step is to apply the knowledge to develop a solution structure that is best suited for the problem following the reuse plan or strategy developed at the previous phase. Various attributes (such as consistency, completeness, and understandability) of the product are evaluated during the development process.

Once a solution structure is identified based on the available predefined components at a given phase, the next step is to reconfigure the solution in order to optimize reuse both at the current phase and the next phase. Doing so requires identifying experts of the next phase activity who will review the proposed solution, identifying candidate components available at the next phase and evaluating the reusability of the candidate components. Based on the potential reuse at the next phase as well as at the current phase, an optimal solution structure is to be identified. We anticipate that the first three steps would be iterated a number of times. The major output from the first three steps is a solution structure and a reuse plan for the next phase. The fourth step includes tasks of making components identified in the solution structure ready for integration. These tasks include acquiring reusable components, modifying and/or instan-

tiating reusable components, and developing the components that cannot be acquired or for which modification is not economic. Finally, the completed components are integrated into the product(s) required for the phase and the integrated products are subjected to a formal review and evaluation before being released to the next phase. If any problem is found during the review, the activities are repeated starting at the step where the problem is introduced before the next phase activities begin.

The next sections show how this generic reuse model is applied to the life cycle phases.

3 System Requirements Analysis

At the system requirements analysis phase, the system requirements are allocated between hardware, software, and personnel. The major activities at this phase, as defined in DoD-STD-2167A, are:

1. Allocating the system requirements between hardware, software, and personnel, and determining whether the software requirements are consistent and complete,
2. Defining a preliminary set of engineering, interface, and qualification requirements for each CSCI, and
3. Developing a software development plan (SDP).

In allocating the system requirements between hardware and software, the current trend is to determine the software requirements first and then to determine the hardware requirements based on the resulting software. This approach is referred to as the software-first approach in [STARS87]. The motivation of the approach is to develop "common, reusable, and machine-independent software parts," and then to utilize those in systems development. Our methodology is based on the software-first approach.

To effectively utilize available resources (i.e., software artifacts, techniques, methods, tools, and human expertise) in the software development, the resources that are applicable for the development of a target system must be identified early in the life cycle, and the development must be planned with consideration of the applicable resources. The system requirements analysis phase is the phase of the life cycle where the functional boundary of software is defined and the development of software is planned. We believe that software reuse must start at this phase. Applicable resources must be identified and an overall strategy of utilizing the identified resources in the development must be developed and be incorporated into the SDP as a reuse plan.

A reuse analysis will be conducted at this phase to develop a resource reuse strategy. Resources available in a particular domain will be surveyed, and the applicability of the resources in the target system development will be assessed. (The assessment at this phase may be at a gross level.) The reuse analysis will focus on macro-level reuse, that is, on identifying similar (in terms of application domain, design, or run-time architecture) systems, especially those that are built for reuse. The assessment results will be used at this phase in determining the functional boundary between software and hardware, in identifying the software components (CSCIs) and their functional boundaries, and in developing a reuse strategy. In this context, the reuse analysis we plan to perform is different from the domain analysis in Neighbors [5]. This methodology focuses on identifying reusable resources in an application domain, whereas Neighbors' methodology focuses on identifying commonalities among the applications in a domain.

The reuse analysis must be performed by or with domain experts. (We have domain experts from industry.) Through the analysis, we want to identify application models, design models, any reusable artifacts (e.g., requirements, design, code, test cases) that may be used with each model, implementation techniques, run-time architectures, and any experts whom we may consult with for further analysis or more information. For each reusable artifact or technique, any alternatives, current applications, advantages and disadvantages, limitations, and experience will be identified. We will also identify any known or potential problems, such as any risks involved with new technology, any reliability, performance, or integration problems (such as development or run-time environment differences, interface compatibility, and standards or hardware problems), or any compatibility problems with project constraints (e.g., programming languages or development tools), and any suggestions to avoid or remedy the problems. Based on these findings and the analysis results, we will develop an overall reuse strategy by classifying the resources in terms of the life cycle phases at which they may be used and the life cycle products into which they may be incorporated. The information collected for each resource during the reuse analysis, including advantages, disadvantages, limitations, and problems, will also be included in the reuse plan to help make decisions at the appropriate development phase.

In addition to increasing reuse in the target system development, we also want to increase the reusability of the developed system for future projects by identifying the components which have reuse potential and developing them for easy adaptation to other applications. In order to do that we need to identify the potential applications of each component, analyze the similarities and differences between the applications, and then select an appropriate design based on the analysis results. A cost/benefit analysis, including the expected cost and schedule overhead, must also be performed to determine the possible impacts to the current system development relative to future systems. Those components with reuse potential and the suggested implementation techniques will also be included in the SDP as development guidelines.

The reuse plan developed at this phase is an overall plan. The plan is evaluated at the beginning of each phase, and, as appropriate, will be modified, adjusted, or refined.

4 Software Requirements Analysis

The major activities at the software requirements analysis phase are, as defined in DoD-STD-2167A:

1. Preparing the software requirements specification (SRS) and the interface requirements specifications (IRS) for each computer software configuration item (CSCI) identified at the previous phase (the system requirements analysis phase),
2. Test planning, in which qualification requirements are defined for each CSCI, and
3. Software specification review (SSR), in which the SRS and IRS are evaluated for their conformance to the predefined criteria.

Of these activities, the activity of preparing the SRS and IRS is refined in this section based on the generic reuse model defined in a previous section.

The major inputs to this phase are the system segment specification (SSS), the preliminary interface requirements specification (PIRS), and the preliminary software requirements specifications (PSRS). The first step to develop SRS and IRS is to study the input documents (SSS, PIRS, and PSRS) and understand the high-level requirements, and then to review the reusable requirements identified in the reuse plan, which is produced at the system requirements analysis phase. (We believe that system structures relevant to the system under development and the specifications of the objects, functions, interfaces, etcetera, defined for each structure are reusable components at this phase.)

Based on a reusable structure(s), a new structure for the target system is created, and reusable components are identified and reviewed. Domain experts are identified, and they review and evaluate the proposed structure and reusable software components. Based on their evaluation results and recommendations, the structure is modified if the modified structure results in an improved solution (for example, a solution that will result in higher productivity or quality). These activities (studying the SSS, PIRS, and PSRS; examining the reusable requirements; creating or modifying the structure; and evaluating the structure against reusable software) are repeated until an optimal structure is obtained. For example, the functional structure of an inventory control system might provide a good framework for developing the functional structure of a library system since both systems maintain information about the components (parts in the case of an inventory system, and books in the case of a library system), inventory status, back orders, customers, etcetera. The objects that are handled by the two systems are different, but the processing requirements can be quite similar.

Once a structure is identified, the components that come from outside the project must be acquired, the components that need to be modified or refined must be processed accordingly, and components that cannot be acquired or that are too expensive to modify must be defined. These pieces are combined to create the requirements documents (SRS and IRS).

After the requirements analysis, the next phase of the life cycle is the preliminary design phase. Based on the reuse activity model, some of the activities of the preliminary design phase are refined in the following section.

5 Design

Software design is performed in the preliminary design phase and the detailed design phase. High-level software components are identified for each CSCI during the preliminary design, and then the identified components are further detailed during the detailed design phase. We believe that the reuse activities at the detailed design phase are basically the same as those at the preliminary design phase but are performed at a finer granularity. Therefore, the discussion will be limited to the preliminary phase. Activities for both phases are identified in the Appendix.

The preliminary design phase is where high-level software components and relationships among the components are defined for the software requirements (SRS and IRS). The major activities of the phase are:

1. Developing a preliminary design for each computer software configuration item (CSCI) identified in the requirements documents (SRS and IRS), and allocating software and interface requirements to the computer software components (CSC) identified in the design.
2. Identifying and describing the formal qualification tests for each CSCI; establishing test requirements for conducting computer software component (CSC) integration and testing.
3. Conducting a preliminary design review (PDR), in which the SDD, IDD, STD, and the CSC test requirements are evaluated for their conformance to the predefined criteria.

Of these activities, the activity of developing a preliminary design is refined in this section.

The major inputs to the preliminary design activity are the requirements documents (SRS and IRS) and the software development plan. The first step in developing a preliminary design is to study and understand the requirements and the software development plan, and then develop a preliminary design reuse strategy (rules and guidelines) by refining and, if necessary, modifying the reuse plan in SDP. Some of the activities for developing a reuse strategy are:

1. Studying potential reuse by analyzing the "internal" (within the product; among the CSCIs) and "external" (with other products) commonalities. (The external commonalities are identified at the system analysis phase through a reuse analysis.),
2. Identifying the components that can be used for the product, and identifying any constraints (e.g., performance or hardware requirements) or design concerns (e.g., design alternatives or implementation techniques) that may affect the quality (e.g., reusability, portability, and maintainability) of the product, and
3. Performing a cost/benefit (e.g., technical difficulties, cost, and time) analysis and selecting the design structures and components that are cost-effective. (State any rules or guidelines for utilizing the structures and components in the development.)

Based on the reuse strategy, a design structure is defined for each CSCI. First, if there are candidate structures identified in the SDP, the design structures that satisfy the functional requirements are selected. Validity of each identified structure is determined by allocating the CSCI functions, memory and processing time requirements, and the interface requirements to the components of each design structure. The best one (for example, the one that requires the least amount of resources to instantiate, modify, operate, and/or maintain) is selected among the candidate structures, and, if necessary, is refined and/or modified following the reuse strategy.

The next step is to integrate the identified structures and components by defining the global data (e.g., structure, value range, accuracy, access, etcetera) and the control flow among the components within each CSCI. Once an initial preliminary design is obtained, the design is evaluated against the reusable components that are available at the next phase and, if necessary, modification is made to the design to increase reuse at the next phase.

Any CSCs and Units that are candidates for use at the detailed design phase are identified, any design concerns, rules or testing considerations that must be looked into or followed are recorded, and any potential benefits or problems (technical, cost, or schedule) that have been found during the analysis are described in the detailed design reuse plan.

The components (CSCs and Units) identified at the previous step are acquired and, if necessary, modification or instantiation is made to meet the allocated requirements. If the component is new, it is described by specifying inputs, outputs, local data, interrupts, timing, sequencing, processing algorithm, error handling, etcetera.

Once a preliminary design is obtained, it is evaluated against the reuse strategy.

6 Coding, Integration, and Testing

The rest of the life-cycle phases identified in DoD-STD-2167A are:

1. Coding and unit testing,
2. CSC integration and testing,
3. CSCI testing, and
4. System integration and testing.

For these phases, the methodology follows DoD-STD-2167A. However, the reused components that were tested previously, have documented test results, and have not been changed since the last testing are not required to be tested in this methodology.

7 Expected Results

By applying the methodology under the controlled conditions of an experiment, we will evaluate the methodology to identify and quantify the benefits and risks of a reuse-based approach. We will also characterize the reusable components and the development process. More specifically, we want to:

1. Identify and classify reusable resources and make an assessment as to how effective reuse was,
2. Evaluate if reuse can be effectively done under the DoD-STD-2167A process model and come up with a recommendation, if appropriate, and
3. Refine the reuse-based methodology and include guidelines for performing a reuse analysis and for developing and applying a reuse strategy.

In the development of this methodology, an emphasis was given in developing a general task model that can accommodate various methods, techniques, or application models. Therefore, depending on the specific methods or techniques selected for a project (e.g., automation-based techniques), some of the tasks may become irrelevant and be eliminated from the methodology. Also, when this methodology is applied to a domain repeatedly, some of the tasks (e.g., identifying reusable resources) might become unnecessary.

We plan to document our experience and the lessons learned as reuse guidelines. Our understanding will help to define the requirements for a high-productivity software engineering environment that is based on reuse. This new technology will mature through application, evaluation, and improvement.

Appendix A Methodology Task List

A Software Development Library (SDL), Software Development Files (SDFs), cost/schedule reports must be created and be kept up-to-date during the development and testing. Tasks of creating and maintaining the SDL, SDFs, and cost/schedule reports are not explicitly defined in the following task list but must be carried out as appropriate. Any subtasks of the tasks defined in the methodology or any corrective tasks for handling problems may be created as needed during the development or testing. The tasks for developing any documents (e.g., Computer Resources Integrated Support Document, Computer Systems Operator's Manual, Software User's Manual, Software Programmer's Manual, Firmware Support Manual) that are not specified in this methodology but are required by the Contract Data Requirements List (CDRL) should be added to the methodology.

A.1 System Requirements Analysis

1. Review the preliminary System/Segment Specification document (DI-CMAN-80008A).
2. Perform a reuse analysis.
 - a. Identify relevant subdomains (application domain, technology domain, computer science domain) of the application, and identify experts in each subdomain (Note: subdomain will be referred to as domain).
 - i. Identify experts of the application domain; consult with the experts to identify the technologies (e.g., communications, navigation) involved with the application and then identify experts in each technology domain; if necessary, consult with the technology domain experts to identify the relevant computer science domain (e.g., database, AI).
 - b. Perform domain analysis.
 - i. Identify a generic application architecture (or a framework) (a generic architecture may be defined in terms of a collections of functions, features, objects, etcetera.). The architecture serves as a model in decomposing a "problem" into subproblems, categorizing previous "solutions" to the subproblems, and synthesizing a solution using the existing solutions.
 - c. Define a framework of the development and test environment.
 - d. Perform resource analysis.
 - i. Survey available resources for the components identified in the generic architecture.
 - a. Identify software artifacts such as software components, application systems, requirements, design, development plan, test plan, test description, and test data.

- b. Identify methods and tools (e.g., development tools, test tools or environments, project management tools).
- ii. Assess the applicability and reusability of the identified resources.
 - a. Identify any project constraints on development environment, including languages, tools, methodologies, and hardware.
 - b. Identify application models, design models, reusable artifacts, implementation techniques (paradigms), run-time architectures, and experts.
 - c. Identify alternatives, current applications, advantages and disadvantages, limitations, and user experiences.
 - d. Identify any constraints or requirements in using the resources (e.g., programming language, hardware, software).
 - e. Understand the data rights and the support policy (error correction, enhancement, and modification), and evaluate documentation and any evidence of certification.
 - f. Identify any known or potential problems (any risks involved with technology, any reliability, performance, or integration problems) and any suggestions to avoid or remedy the problems.
 - g. Identify any relationships between the resources (e.g., if two components require different hardware, selection would be exclusive of each other) and determine how selection of a resource affects utilization of other resources.
 - h. Identify the development approaches (e.g., object-oriented, functional decomposition) that are best suited for each reusable resources.
 - i. Assess the reusability of the identified resources and determine the applicability of the resources to the application; evaluate and determine if they perform as documented.
 - j. Identify groups of resources, each of which can form a consistent development environment (including tools, methods, and artifacts).
- 3. Identify the Computer Software Configuration Items (CSCIs) that will maximize reuse (while minimizing the risk), and then allocate the system requirements to the CSCIs.

4. Determine hardware configuration items (HWICs) and manual operations considering the CSCIs identified at the previous step; document the allocation of system requirements in System/Segment Specification (DI-CMAN-80008A) and System/Segment Design Document (SSDD) (DI-CMAN-XXXXX, DI-ECRS-8XX1).
5. Define a preliminary set of engineering requirements for each CSCI and document these requirements in a preliminary Software Requirements Specification (SRS) (DI-MCCR-80025A, DI-ECRS-8XX7).
6. Determine a preliminary set of interface requirements for each CSCI and document these requirements in a preliminary Interface Requirements Specification (IRS) (DI-MCCR-80026A, DI-ECRS-8XX8).
7. Define a preliminary set of qualification requirements for each CSCI and document these requirements in the SRS.
8. Develop a Software Development Plan (including an overall reuse strategy) (DI-MCCR-80030A, DI-ECSR-8XX3).
 - a. Develop a reuse strategy.
 - i. Classify the applicable resources for each CSCI in terms of the generic architecture, life-cycle phases, and the life-cycle products.
 - ii. Describe the relationships (mapping, structural, utilization, application) amongst the resources (in a form of traceability matrix).
 - iii. Include the information such as requirements solved by reusable resources, advantages, disadvantages, limitations, problems, or any suggestions that will help make decisions at each development phase.
 - iv. Describe any constraints in using (or imposed by) the resources; for example, selection of one component might affect selection of the others.
 - v. Describe the data rights, support policy, and the certification plans; obtain contracting agency approval.
 - b. Develop a software configuration management plan (DI-ECRS-8XX4).
 - c. Develop a quality evaluation plan (DI-MCCR-XXXXX, DI-ECRS-8XX5).
 - d. Develop schedule, milestones, and development procedures.
9. Evaluate the SDP, SSDD, preliminary SRS, and the preliminary IRS.
10. Place documents (SSS, SDP, SSDD, preliminary SRS, and preliminary IRS) under configuration control.

A.2 Software Requirements Analysis

1. Study and understand the system requirements (the preliminary SRS, IRS, and SSDD).
2. Develop a requirements reuse plan based on the reuse strategy.
 - a. Study the reuse strategy in SDP.
 - b. Review domain/resource analysis results.
 - i. Identify the domains relevant to the application system.
 - ii. For each subdomain within the application domain, identify the candidate reusable components (CSCIs).
 - iii. Analyze the components for validity, compatibility, modularity, modifiability, etcetera.
 - iv. Study potential reuse by analyzing the "internal" (within the product; among the CSCIs) commonalities.
 - c. Identify design concerns that affect reusability and maintainability.
 - d. Perform cost/benefit analysis and develop a reuse plan.
3. CSCI requirements definition.
 - a. Based on the reuse plan, define or identify candidate target structures (rules, components, and relationships), and integrate the candidate structures.
 - i. Identify structures based on the reuse plan.
 - ii. Determine the validity of the identified structures.
 - a. Analyze the functional, performance, interface characteristics for validity, consistency, and completeness.
 - iii. Define and integrate structures.
 - a. Describe the top-level data and control flow between CSCIs.
 - b. Identify and describe the global data (e.g., structure, value range, accuracy, access, etcetera.).
 - b. Select, acquire, and modify structures.
 - i. Examine and identify candidate reusable top level Computer Software Components (CSC) relevant to the CSCI functions.
 - ii. Determine selection criteria (e.g., increase commonality) other than maximizing reusable CSCs.
 - iii. Evaluate and select structures (rules and relationships) from the candidate structures (identified above at iii.a).
 - iv. Modify structures and associated artifacts (e.g., global data definition), if needed.

- v. Identify the structures reusable in the future for other applications; the identified structures are incorporated into the generic architecture.
 - vi. Update the preliminary design reuse strategy in SDP.
 - a. Identify top-level Computer Software Components (CSCs) and critical lower level CSCs that may be reused.
 - b. Describe any design concerns, rules or testing considerations that must be looked into or followed at the next phase.
 - c. Identify potential benefits or problems.
 - c. Specify components (CSCIs); acquire/select and modify components.
 - i. Perform a cost/benefit analysis with respect to reuse.
 - ii. Acquire/select components.
 - iii. Modify/instantiate components.
 - iv. Specify new components.
 - a. Describe each CSCI by specifying inputs, outputs, local data, interrupts, timing, sequencing, processing algorithm, error handling, etcetera.
 - v. Identify the components reusable in the future; update the reusable resources knowledge base.
 - d. Integrate components and produce SRSs and IRS.
- 4. Test planning.
 - a. Define a complete set of qualification (correctness, reliability, etc.) requirements for each CSCI; document these requirements in SRSs; for reusable CSCIs, use the qualification requirements defined for the CSCIs, if applicable.
- 5. Evaluate the SRSs and IRS using the evaluation criteria in DoD-STD-2167A; produce a summary of the evaluation results.
- 6. Perform Software Specification Review(s) (SSRs).
- 7. Place the SRSs and IRS under the configuration control.

A.3 Preliminary Design

1. Study and understand the software requirements.
2. Develop a (preliminary) design reuse plan (rules and guidelines) based on the reuse strategy. (Note: A design reuse plan covering both the preliminary and detailed design phases may be developed here.)
 - a. Study potential reuse by analyzing the "internal" (within the product, among the CSCIs) and "external" (with other products) commonalities.
 - b. Identify any constraints (e.g., performance requirements, hardware requirements, etc.) that may affect reusability and maintainability.
 - c. Identify design concerns that affect reusability and maintainability.
 - d. Perform cost/benefit (e.g., technical difficulties, cost, time, etcetera) analysis and develop reuse strategies.
3. CSCI top-level design.
 - a. Based on the reuse plan, define or identify candidate structures (rules, high-level Computer Software Components (CSCs), and relationships), and integrate the candidate structures.
 - i. Identify structures based on the requirements.
 - ii. Determine validity of the identified structures.
 - a. Allocate CSCI functions and interfaces to CSCs.
 - b. Allocate memory and processing time to CSCs as applicable.
 - iii. Define and integrate structures.
 - a. Describe the top-level data and control flow within each CSCI.
 - b. Identify and describe the global data (e.g., structure, value range, accuracy, access, etcetera).
 - b. Select, acquire, and modify structures.
 - i. Examine and identify candidate detailed design reusable components relevant to the candidate high-level structures.
 - ii. Determine selection criteria (e.g., increase commonality) other than maximizing detailed design reusable components.
 - iii. Evaluate and select candidate structures (rules and relationships).
 - iv. Modify structures, if needed.
 - v. Identify the potential reusable artifacts (especially designs at domain level) in the future; update reusable resources knowledge base.

- vi. Update the detailed design reuse strategy in the SDP.
 - a. Identify low level CSCs and Computer Software Units (CSUs) that may be reused.
 - b. Describe any design concerns, rules or testing considerations that must be looked into or followed at the next phase.
 - c. Identify potential benefits or problems (technical, cost, schedule).
 - c. Specify components (CSCs); acquire/select and modify components.
 - i. Perform a cost/benefit analysis with respect to reuse.
 - ii. Acquire/select components.
 - iii. Modify/instantiate components.
 - iv. Specify new components.
 - a. Describe each CSC by specifying inputs, outputs, local data, interrupts, timing, sequencing, processing algorithm, error handling, etcetera.
 - v. Identify the components reusable in the future; update the reusable resources knowledge base.
 - d. Integrate components and produce preliminary Software Design Documents (SDDs) (DI-MCCR-80012A, DI-ECRS-8X10) and Interface Design Document (IDD) (DI-MCCR-80027A, DI-ECRS-8X12).
4. Test planning.
- a. Establish test requirements for conducting CSC integration and testing and record the requirements in the CSC SDFs; identify different classes (timing test, erroneous input/recovery test, maximum capacity test, interface test, security test, regression test) of CSC tests.
 - b. For formal CSCI testing, identify the test requirements; test organization, responsibilities, and schedule information; different classes of formal tests; data recording, reduction, and analysis requirements; and the purpose of each formal test planned.
 - c. Identify all the resources (facilities, personnel, hardware, software) required for informal and formal testing.
 - d. Identify reusable testing artifacts for large grained reuse.
 - e. Produce a Software Test Plan (STP) (DI-MCCR-80014A, DI-ECRS-8X16) and CSC test requirements documents.
5. Evaluate preliminary SDDs, preliminary IDD, STP, and the CSC test requirements using the evaluation criteria in DoD-STD-2167A; produce a summary of the evaluation results.

6. Perform Preliminary Design Reviews (PDRs).
7. Place the documents (SDDs, IDD, STP, CSC test requirements) under the configuration control.

A.4 Detailed Design

1. Study and understand the preliminary design.
2. Complete a detailed design reuse plan (rules and guidelines) based on the detailed design reuse strategy in SDP.
 - a. Study potential reuse by analyzing the "internal" (within the product; among the CSCs and CSUs) and "external" (with other products) commonalities.
 - b. Identify design concerns that affect reusability and maintainability.
 - c. Perform a cost/benefit analysis and develop reuse strategies.
3. CSC design.
 - a. Based on the reuse plan, define or identify candidate structures (rules, low-level CSCs and CSUs, and relationships among them), and integrate the candidate structures.
 - i. Identify structures for each CSC (refine CSCs into low level CSCs and CSUs).
 - ii. Determine validity of the identified structures.
 - a. Allocate CSC functions and interfaces to the refined CSCs (low level CSCs and CSUs).
 - b. Allocate memory and processing time to CSCs.
 - iii. Define and integrate structures.
 - a. Describe the data and control flow within each CSC.
 - b. Identify and describe the global data (e.g., structure, value range, accuracy, access, etcetera).
 - b. Select, acquire, and modify structures.
 - i. Examine and identify candidate reusable CSU components (code, libraries) relevant to the candidate low-level structures.
 - ii. Determine selection criteria (e.g., increase commonality) other than maximizing CSUs (libraries, code).
 - iii. Evaluate and select candidate structures (rules and relationships).
 - iv. Modify structures (and associated algorithms), if needed.
 - v. Identify structures (designs) reusable in the future; update the reusable resources knowledge base

- vi. Update the code reuse strategy in SDP.
 - c. Specify components; acquire/select and modify components.
 - i. Perform a cost/benefit analysis with respect to reuse.
 - ii. Acquire/select components.
 - iii. Modify/instantiate components.
 - iv. Specify new components.
 - a. Identify global data definitions within each CSC.
 - b. Identify inputs (global data, direct I/O, parameters, shared memory, etc.), input data attributes (source, format, unit of measure, limits or value ranges, accuracy and precision, frequency of input arrival, legality checks for erroneous information), local data definitions, process control requirements (source, purpose, and priority of signal/interrupt; required response time; minimum, maximum, and probable frequency of signal/interrupt), processing (control, algorithms, special control features, error handling, data conversion, communication interface, etc.), utilization of other elements, limitations, and outputs of all CSCs.
 - c. Identify inputs, local data definitions, process control requirements, processing algorithms, special control features, protection, error handling, utilization of other elements, limitations, and outputs for all CSUs.
 - d. Data base design including DBMS overview, database structure, database file design, and database references.
 - v. Identify the components reusable in the future; update the reusable resources knowledge base.
 - d. Integrate CSCs and produce SDDs and an IDD.
4. Test planning.
- a. Identify the requirements, responsibilities, and schedule for testing all CSUs; record this information in SDFs.
 - b. Describe test cases for each CSU in terms of inputs, expected results, and evaluation criteria; record this information in CSU SDFs.
 - c. Establish test responsibilities and schedule for conducting CSC integration and testing; record this information in CSC SDFs.
 - d. Describe the test cases for each CSC integration and testing in terms of inputs, expected results, and evaluation criteria; record this information in CSC SDFs.

- e. Describe the test cases for each formal CSCI test, in terms of initialization requirements, input data, expected intermediate results, expected output data, criteria for evaluating results, and any assumptions and constraints; produce Software Test Description (STD) for each CSCI (DI-MCCR-80015A, DI-ECRS-8X17).
 - f. Identify test artifacts for reusable components.
5. Evaluate the SDDs, IDD, CSU test requirements and test cases, CSC test cases, SDFs, and STD using the evaluation criteria in DoD-STD-2167A; produce a summary of the evaluation results.
 6. Perform Critical Design Review(s) (CDRs).
 7. Place the SDDs, IDD, and STDs under the configuration control.

A.5 Coding and CSU Testing

Note: CSUs that were tested previously, have documented test results, and have not been changed since the last test need not be tested again.

1. Review SDP and identify the resources (code, test procedures) that can be applied at this phase.
2. Review the STP and SDFs; understand the overall test requirements, responsibilities, and the schedule.
3. Develop test procedures for conducting each CSU test; record the procedures in the corresponding SDFs.
4. Code and test each CSU; record the test results in the corresponding SDFs.
5. Make all necessary revisions to the design documentation and code; perform retesting and update the SDFs.
6. Develop test procedures for conducting each CSC test; record the procedures in the SDFs.
7. Evaluate the products (the source code, CSU test procedures and test results, CSC test procedures, SDFs) using the evaluation criteria in DoD-STD-2167A.
8. Place the source code and updated design documents (if any) under the configuration control.
9. Identify potential reusable code, tests, and designs.

A.6 CSC Integration and Testing

Note: CSCs that were tested previously, have documented test results, and have not been changed since the last test need not be tested again.

1. Review the STP, STD, and SDFs; understand the overall test requirements, responsibilities, and the schedule.

2. For each CSCI, perform the FQT activities in accordance with the procedures documented in the STD; record the test results in the Software Test Report(s) (STRs) (DI-MCCR-80017A, DI-ECRS-8X19).
3. Make all necessary revisions to the design documentation, code, and IDD; conduct all necessary retesting, and update STDs and SDFs.
4. For each CSCI, prepare a Software Product Specification (DI-MCCR-80029A) and the source code for delivery.
5. Evaluate the STRs, the updated source code and design documentation using the evaluation criteria in DoD-STD-2167A.
6. Identify the exact version of each CSCI to be delivered; document this information in a Version Description Document (VDD) (DI-MCCR-80013A, DI-ECRS-8X15) for each CSCI.
7. Have the products authenticated (Function Configuration Audit and Physical Configuration Audit) by the contracting agency; baseline SPSs.
8. Identify test artifacts for reusable CSCIs.

A.7 System Integration and Testing

1. Support the development and documentation of system integration and test plans, test cases, and test procedures.
2. Support system integration and testing activities.
3. Support post test analysis and reporting of system integration and test results.
4. Perform evaluation of the updated source code and design documentation using the evaluation criteria in DoD-STD-2167A.

References

- [1] Bassett, P. G. "Frame-based Software Engineering." *IEEE Software* (July 1987).
- [2] *Defense System Software Development*, Military Standard DOD-STD-2167A, October 27, 1987.
- [3] Goguen, J.A. "Reusing and Interconnecting Software Components." *IEEE Computer* (February 1986).
- [4] Lenz, M., Schmid, H.A., Wolf, P.F. "Software Reuse through Building Blocks." *IEEE Software* (July 1987).
- [5] Neighbors, J. M. "The Draco Approach to Constructing Software from Reusable Components." *IEEE Transactions on Software Engineering* (September 1984): 564-574.
- [6] "STARS Reusability Guidebook V4.0." *STARS Application Workshop*, NRL, San Diego, CA (September 1986).

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS None		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-92-SR-4			5. MONITORING ORGANIZATION REPORT NUMBER(S) 		
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute		6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office		
6c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213			7b. ADDRESS (City, State and ZIP Code) ESD/AVS Hanscom Air Force Base, MA 01731		
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office		8b. OFFICE SYMBOL (if applicable) ESD/AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003		
8c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213			10. SOURCE OF FUNDING NOS.		
			PROGRAM ELEMENT NO 63756E	PROJECT NO. N/A	TASK NO N/A
			WORK UNIT NO. N/A		
11. TITLE (Include Security Classification) A Reuse-Based Software Development Methodology					
12. PERSONAL AUTHOR(S) Kyo C. Kang, Sholom Cohen, Robert Holibaugh, James Perry, and A. Spencer Peterson					
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM TO		14. DATE OF REPORT (Yr., Mo., Day) May 1992	
15. PAGE COUNT 30 pp.					
16. SUPPLEMENTARY NOTATION 					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number) method reuse software development		
FIELD	GROUP	SUB. GR.			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Software has been reused in applications development ever since programming started. However, the reuse practices have mostly been ad hoc, and the potential benefits of reuse have never been fully realized. Most of the available software development methodologies do not explicitly identify reuse activities. The Application of Reusable Software Components Project of the Software Engineering Institute is developing a reuse-based software development methodology, and the current direction and the progress of the methodology work are discussed in this paper. The methodology is based on the life cycle model in DoD-STD-2167A with refinement of each phase to identify reuse activities. The reuse activities that are common across the life cycle phases are identified as: 1) studying the problem and available solutions to the problem and developing a reuse plan or strategy, 2) identifying a (please turn over)					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED SAME AS RPTDTC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution		
22a. NAME OF RESPONSIBLE INDIVIDUAL John S. Herman, Capt, USAF			22b. TELEPHONE NUMBER (Include Area Code) (412) 268-7631		22c. OFFICE SYMBOL ESD/AVS (SEI)

solution structure for the problem following the reuse plan, 3) reconfiguring the solution structure to improve reuse at the next phase, 4) acquiring, instantiating, and/or modifying existing reusable components, 5) integrating the reused and any newly developed components into the products for the phase, and 6) evaluating the products. These activities are used as the base model for defining the specific activities at each phase of the life cycle.

this methodology focuses more on identification and application of reusable resources than on construction of reusable resources, and some enhancements in the construction aspect might be necessary to make it more complete.

This methodology has never been applied; it will be used in an application redevelopment experiment and then will be improved based on our experience.