1

TASK: UA48
CDRL: 04102
19 June 1992

Integrating Domain-Specific Reuse For System/Software Engineers Course Description

Informal Technical Data

STARS-AC-04102/001/00
19 June 1992

TASK: UA48
CDRL: 04102
19 June 1992

INFORMAL TECHNICAL REPORT

For The

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS)

Integrating Domain-Specific Reuse for System/Software Engineers Course Description - Preliminary

STARS-AC-04102/001/00
19 June 1992

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0009

Prepared for:

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

Prepared by:

EWA
under contract to
Paramax Systems Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

TASK: UA48
CDRL: 04102
19 June 1992

Data ID: STARS-AC-04102/001/00

TASK: UA48
CDRL: 4102
19 June 1992

INFORMAL TECHNICAL REPORT
Integrating Domain-Specific Reuse for System/Software
Engineers Course Description
Central Archive for Reusable Defense Software
(CARDS)

Principal Author(s):

Senior Software Designer ***Kammi Kai Hefner*** ***Date***

Software Engineer ***Kerrin Elizabeth Smith*** ***Date***

Approvals:

Task Manager ***Lorraine Martin*** ***Date***

(Signatures on File)

PREFACE

This is a preliminary document which will be updated for release in early 1993. Feedback will be solicited during this update period. The final edition will include enhancements to the course content and the development of accompanying course materials.

The scope of the course content will be reviewed. The course content focuses on designing software systems with software reuse. Specific consideration will be given to enhancing this course to address developing software systems for reuse. Based on the results of the scoping effort the duration of the course may be adjusted.

A list will be provided of required course materials. An example set of course materials will be developed using a selected subject domain. The materials will include required handouts, slides, and reference documentation for the corresponding domain-specific library.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE 19 June 1992	3. REPORT TYPE AND DATES COVERED Informal Technical Report

4. TITLE AND SUBTITLE Integrating Domain-Specific Reuse For System/ Software Engineers	5. FUNDING NUMBERS F19628-88-D-0031
6. AUTHOR(S) EWA Incorporates	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Paramax Corporation 12010 Sunrise Valley Drive Reston, VA 22091	8. PERFORMING ORGANIZATION REPORT NUMBER STARS-AC-04102/001/00
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Department of the Air Force Headquarters Electronic Systems Division Hanscom AFB, MA 01731-5000	10. SPONSORING/MONITORING AGENCY REPORT NUMBER 04102

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT Distribution "A"	12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This course provides system and software engineers an introduction to syster. development with domain-specific software reuse. A domain is a set of current ana future applications marked by a set of common capabilities and data. Domain-specific software reuse is the reuse of ideas, knowledge, artifacts, personnel and software components in an existing domain. The course introduces the methods necessary to integrate domain-specific software reuse concepts into current system and software development processes by emphasizing domain analysis, generic architecture development, specific architecture development, and system composition. This course is intended for use in both government and industry training. The course could be tailored for presentation at the university level.

14. SUBJECT TERMS Domain Specific Reuse	15. NUMBER OF PAGES 30
	16. PRICE CODE

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

TABLE OF CONTENTS

1. INTRODUCTION

1.1 PURPOSE

This course provides system and software engineers an introduction to system development with domain-specific software reuse. A domain is a set of current and future applications marked by a set of common capabilities and data. Domain-specific software reuse is the reuse of ideas, knowledge, artifacts, personnel and software components in an existing domain. The course introduces the methods necessary to integrate domain-specific software reuse concepts into current system and software development processes by emphasizing domain analysis, generic architecture development, specific architecture development, and system composition. This course is intended for use in both government and industry training. The course could be tailored for presentation at the university level.

1.2 RATIONALE

The DoD is moving towards a domain-specific structured approach to system and software development. Domain-specific software reuse has the potential to reduce system and software development costs, risks, and schedules while increasing reliability.

Education is the foundation for integrating new concepts into an existing process. To facilitate domain-specific software reuse, it is necessary to provide system and software engineers with an understanding of methods and techniques necessary to perform software reuse activities. This Integrating Domain-Specific Reuse for System/Software Engineers course provides the support for the integration of domain-specific reuse into the system and software development processes.

1.3 STUDENT CHARACTERIZATION

This course is intended for skilled system and software engineers.

1.3.1 System Engineers

System engineers, called systems analysts in some application domains, are concerned with the decomposition of systems, the allocation of software development responsibility for specific system components (to specialist developers or manufacturers), and with the subsequent composition of software and hardware system components to produce the final system [1].

System engineers begin with customer-defined goals and constraints and derive a representation of function, performance, interfaces, design constraints, and information structure that can be allocated to each of the generic system elements [23].

1.3.2 Software Engineers

Software engineers are concerned with how the individual software components are actually designed, developed, and integrated into the total system.

1.4 STUDENT PREREQUISITE KNOWLEDGE

Successful completion of this course requires that the student possess prior experience in participating in a complex software development project (e.g., 100K Lines of Code); knowledge of systems analysis and requirements; and knowledge of structured analysis design techniques and/or object-oriented analysis techniques to the depth presented in a one semester undergraduate advanced software engineering course.

It would be beneficial, but is not required, that the student be Ada literate and possess some knowledge of model-based engineering [2].

1.5 COURSE COMPLETION CRITERIA

The following products will be produced as proof of course participation:

1. A context model for a simple domain.
2. A domain model for a simple domain.
3. A specific system architecture from a generic architecture.
4. A system prototype using a domain-specific software reuse library.

1.6 COURSE OUTLINE SCHEDULE

This course will be presented over four consecutive eight-hour days.

Day One:	Lecture	-	Introduction to software reuse
Day Two:	Lecture	-	Domain analysis
	Activity	-	Context modeling Domain modeling
Day Three:	Lecture	-	Introduction to simple domain-specific library which corresponds to lab exercises
	Activity	-	Specific architecture development System composition
Day Four:	Activity	-	Continued from day three
	Lecture	-	How reuse can be incorporated into the system development processes used by the students
	Lecture	-	Course summary

2. ANNOTATED COURSE OUTLINE

2.1 DAY 1 - INTRODUCTION TO SOFTWARE REUSE CONCEPTS

2.1.1 Learning Objectives

Domain-specific software reuse is gaining recognition as an important part of the system and software development processes. It is necessary for the students to have an understanding of some of the factors surrounding software reuse.

The direct objective of this lecture is to demonstrate to system and software engineers the potentials of domain-specific software reuse.

Upon completion of this lecture the student will be able to:

1. Define software reuse.
2. List the most important incentives for integrating software reuse.
3. Explain the supply and demand issues associated with software development and the potential payoffs offered by software reuse.
4. Define software reuse artifacts (components, assets).
5. Differentiate between designing software systems with reuse and designing software systems for reuse.
6. Explain how software reuse artifacts fit into various software development life-cycle models.
7. Define domain and domain-specific software reuse.
8. Explain what domain-specific software reuse factors contribute to the success of a specific project.
9. List and discuss two domain-specific pioneering software reuse projects.
10. Explain how a library is used as a tool.
11. Define library mechanism.
12. List several software reuse library mechanisms.
13. Define classification, classification methods, and classification scheme.
14. Give at least two examples of domain-specific software reuse libraries.

The following references are recommended for developing this section [3, 4, 5, 6, 7].

2.1.2 Lecture Content Area

Software reuse is the reapplication of domain knowledge, development experience, design decisions, architectural structures, requirements, designs, code, and documentation from existing systems to an emerging system in an effort to reduce the costs associated with software development and maintenance [3]. Cost includes manpower (skills and availability) and risk to the schedule, software, and hardware budgets.

The students will be introduced to software reuse. This introduction will provide students with an understanding of why software reuse is gaining recognition as an important part of the software development process. The instructor will discuss where software reuse fits in the software development life-cycle. The students will be presented with early software reuse projects and learn what made some projects successful.

A. Incentives for Software Reuse

Students will gain an understanding of the supply and demand issues associated with software development and the potential payoffs offered by software reuse.

1. Lower Software Development Cost [3]

   a. The cost of software is increasing while the cost of hardware is decreasing.
   b. Reusing validated components reduces the cost of software development.
   c. The developer must consider the cost of locating, adapting, and integrating the validated component, versus developing it from scratch.

2. Shortage of Experienced Personnel [8,9]

   a. The demand for qualified software development personnel is steadily increasing.
   b. Fewer and fewer are pursuing a career in the software development fields.
   c. Productivity is increased by reducing all aspects of the development effort across the life-cycle.

3. System Reliability is Increased [6]

   a. Components which have been previously proven in working systems are more reliable than untested components.
   b. System testing time is reduced.

4. Overall Risk is Reduced [6]

   a. Greater uncertainty exists in the costs associated with developing a new component.
   b. There is an initial investment associated with reusing an existing component.
   c. There is a reduction in the overall uncertainty in the project cost estimation.

5. Software Development Time can be Reduced [6]

   a. Early completion is often more important than overall development costs.
   b. System development time should be reduced.

B. Reuse and the Software Life-cycle

Students will be introduced to the concepts of designing for reuse and designing with reuse. Students will become familiar with what reusable components are and how they can be incorporated into the software development life-cycle.

1. Design for Reuse

   Build into the design and implementation the flexibility, variations and adaptability to create a reusable component. The instructor will discuss software reverse-engineering, object-oriented design (information hiding and portability), and automated documentation. Constructs of languages such as Ada and C++, which facilitate the development of reusable components, will be presented.

2. Design with Reuse

   Incorporate available reusable components into system analysis and design. Discuss reuse-in-the-large versus reuse-in-the-small. Address reuse of modified and unmodified components.

   Activity: Present a brief example of an analysis in which a modified component was reused. Stress the potential gains and potential losses of incorporating the modified component into the system. Examine how the modified component did and did not facilitate reuse.

3. Entry-points of Life-cycle Artifacts

   It is important to stress that reusable components include all life-cycle artifacts such as architectures, models, requirements, design, code, executables, and test suites. The instructor will present several life-cycle models (e.g., waterfall, spiral), demonstrate where software reuse can be incorporated, and identify which reusable components are valuable within the various life-cycle phases.

C. Evolution of Domain-Specific Software Reuse

Domain is the set of current and future applications marked by a set of common capabilities and data (i.e., an area of activity or knowledge).

Students will be introduced to several pioneering domain-specific software reuse projects, what their goals were and what they accomplished. Students will learn why some early software reuse projects were successful and how their success was measured. The instructor will emphasize domain-specific software reuse as a means of achieving success in future software reuse projects.

1. Domain-specific Software Reuse

The instructor will introduce the students to the concept of domain analysis product reuse. The instructor needs to explain the basic concepts and advantages of domain-specific software reuse, what domain products are, how they are created and how they relate to the domain. This is just a top-level introduction; the detailed presentation follows.

2. Pioneering Software Reuse Projects

Common Ada Missile Packages (CAMP) project conducted by McDonnell Douglas under contract to the DOD STARS program:

The federally-funded Software Engineering Institute at Carnegie-Mellon University in Pittsburgh has conducted the Applications of Reusable Software Components (ARSC) project, which has experimented with the CAMP reusable parts by developing software for the Tomahawk missile system. The U.S. Army Communications Electronics Command (CECOM) Center for Software Engineering has also conducted research based on the CAMP parts. The CAMP project is an example of an early domain-specific software reuse library [10].

Magnavox's **AFATDS** project:

The project consisted of approximately 770,000 lines of Ada code. Approximately 100,000 of those lines were reused code. In the initial program planning stages, it was determined that a specific software reuse factor was required to meet the proposed schedule and budget. Object-oriented design techniques were used and incentives were offered to encourage software reuse [10].

Raytheon Missile Systems Division:

Successful software reuse experience in business applications reported up to 60% reuse which increased productivity by 50% [11].

NEC's Software Engineering Laboratory:

Successful experience using their reuse-based SEA/I environment reported a 6.7 fold productivity improvement and 2.8 fold quality improvement in three controlled experiments [11].

Fujitsu's Software Development for Electronic Switching Systems (SDESS):

Successful experience reporting significant improvement in software development measured in percent of current projects being on time. Seventy percent of 300 projects vs 20% before SDESS [11].

GTE DS Asset Management Program:

Successful experience reporting 14% reuse first year. Fifty percent expected by the end of 5th year [11].

D. Payoffs of Domain-specific Software Reuse

Activity: Present a comparison of building a system architecture from scratch to building the same system using domain-specific software reuse artifacts. The instructor will demonstrate the benefits of domain-specific software reuse.

The instructor will present an example which illustrates the development of a simple system architecture by using two sets of metrics. The first set of metrics is based on the development with no reuse. The second set of metrics is based on a development process which exploited domain-specific reuse. The instructor and students will analyze the areas in which cost savings occurred. Several models available for this activity include: Constructive Cost Model (COCOMO); the Software Analysis Cost Model (SWAN); SoftCost-Ada and Kanter's Factors.

E. Software Reuse Libraries

1. Library as a Reuse Tool

The instructor will introduce the reuse library as a tool, not as the reuse solution. Software reuse libraries are not a new concept. The instructor will discuss how most individuals maintain their own pool of reusable components and how compilers are equipped with their own set of reusable libraries. The instructor will present how the software reuse library concept is now being extended to include publicly accessible and centrally located libraries of life-cycle artifacts. This extension is essential if the full potential of software reuse is to be realized.

2. Reuse Library Representations

"Classification is grouping like things together. All members of a group, or class, produced by classification share at least one characteristic that members of other classes do not. Classification displays the relationships among things and classes of things. The result is a network or structure of relationships"[3]. Classification is addressed in reference [3], section 4.3, and in several papers contained in reference [7]. The instructor will develop a lecture that presents a brief overview of various methods (e.g., Dewey decimal, semantics, faceted, indexed, and keywords).

3. Search and Retrieval Methods

The instructor will have the students consider full text searching, facet searching, keyword searching, knowledge-based searching, and browsing.

Activity: The instructor may take a poll to determine the form of searching the students believe they would prefer. Benefits of selections for different environments should be discussed.

4. Available Libraries

The instructor will provide the students with an overview of several different types of libraries and the purpose each serves the software reuse community. The instructor will discuss the classification and search and retrieval methods used by each library. The instructor will have literature available for the students to browse through (or take) on the libraries presented. The instructor will provide a directory of all software reuse libraries with contacts and phone numbers. Some of the reuse libraries available include:

- a. CARDS
- b. ASSET
- c. RAPID
- d. DSRS
- e. AdaNET
- f. COSMIC
- g. ASR
- h. Others
  - (1) AdaSoft
  - (2) Booch
  - (3) GRACE
  - (4) Ada Math Advantage

2.2 DAY 2 - DOMAIN ANALYSIS

2.2.1 Learning Objectives

The teaching of domain analysis will be complete enough to instill an awareness of domain analysis techniques and their role in systematic software reuse. This section will also build a trust in the products produced by the domain analysis process.

Upon completion of this lecture the student will be able to:

1. Express why domain analysis is important.
2. Define domain modeling.
3. List three types of domain models.
4. Define modeling techniques.
5. Give an example of three different modeling techniques.
6. Distinguish between a functional model and a feature model.
7. Differentiate between a feature model and a object model.
8. Justify why composition rules are beneficial.
9. Define functional model.
10. Present the process of generating requirements for a generic architecture based on the functional model.
11. Explain what factors contribute to assessing whether or not a domain is a good candidate for domain analysis.
12. Narrate the importance of determining the boundaries (scoping) of the domain being analyzed.
13. Define context diagram.
14. Explain what areas should be addressed in developing a context diagram.
15. Give an example of the types of questions that should be considered when determining what is outside of a domain.
16. Define an operational model.
17. Explain what information is captured by domain modeling.
18. List and explain the products of a domain analysis process.
19. Describe how a generic architecture is derived.

The following references are recommended for developing this section [4, 7, 12, 13, 14, 15, 16, 17, 18, 19].

2.2.2 Lecture Content Area

The CARDS project is not prescribing one particular approach to domain analysis, but references [13, 18] provide a solid foundation for exposition and course activities. These papers have been used to develop this section of the course. It is assumed that the students have a working knowledge of structured analysis and object-oriented analysis techniques. The instructor will begin the lecture with a very brief overview of these concepts and methods before addressing domain analysis.

A. Domain Modeling Methodologies

The instructor will provide an overview of several domain modeling methodologies. Most domain modeling methodologies formulate similar models and viewpoints of the domain. These models can often include feature models, functional models, object models, and composition rules. Some of the common modeling techniques used are entity-relationship models, state diagrams, hierarchical models and other structured analysis and design techniques. The domain model will capture the common parts of the domain along with the differences. The data commonality and the control flow for the functionality is captured in a functional model. This functional model is then used to generate the requirements for a generic architecture and the reusable components that implement the architecture. The following references are recommended for developing this section [2, 5, 11, 13, 16, 17, 18, 19].

B. Domain Identification

Students will gain an understanding of how to assess whether or not a domain is a good candidate for domain analysis.

It is necessary to choose the right domains to analyze. A domain must be stable and well understood to be a good candidate for domain analysis. A discussion of how a domain is chosen will occur and an illustration will be provided by the instructor. Some questions to consider in making domain selections are:

1. Is the domain broad or narrow?
2. Is the domain mature and well understood?
3. Is the domain stable or changing continuously?
4. Is the domain strongly dependent on technology?
5. Is the domain based on well established principles, methods, and formalisms?

The instructor will discuss the importance of identifying the resources they can draw upon in meeting their goals. Who do they have to work with? What do they have to work with? How will they verify their models?

C. Defining the Domain and Domain Modeling

Students will learn the importance of determining the boundaries (scoping) of the domain being analyzed and receive hands-on experience developing two different types of models, an entity-relationship context diagram and a feature-oriented operational model [13].

In performing domain analysis, the boundaries of the domain must be properly scoped. This process places the domain relative to other domains and defines the external entities and data flows between the external entities and the domain.

The instructor will provide the name of the domain being modeled, a short description of the domain and the general user needs to be satisfied by applications in this domain. The instructor will provide the students with models of example systems in the domain.

Activity: The students will create a context diagram of the domain being analyzed and a feature-oriented operational model for the domain. These will not be independent exercises. The instructor will lecture, allow for discussion and then the students will perform exercises. This will continue throughout the lab. Students will work in groups on the exercises. The instructor, acting as the domain expert, will validate the models produced by the students.

1. Context Diagram

The students will draw a high-level block diagram showing the context of the domain and the relationships between the entities in the domain (e.g., an entity-relationship diagram). The instructor will discuss the context of the domain and the domain itself and then let each group develop their own model. A time limit will be set on this exercise. The instructor will collect the models developed and evaluate them at a later time. Models developed by the instructor will be handed out when the students' models are collected. The following types of questions will be provided to the students to guide them through this activity:

a. What is inside the domain?

(1) What are the classes of applications in this domain?
(2) What primary functions/objects/things are in the domain?
(3) What kinds of trade-offs exist in this domain?

b. What is outside the domain?

(1) What functions/objects/things in the domain are outside the scope of the sub-domain we are choosing to analyze?
(2) What are similar/related domains and sub-domains?
(3) How does this domain relate to other domains?

c. What is on the borders of the domain (input/output)?

(1) What are the inputs to the domain?
(2) What are the outputs from the domain?
(3) Where do inputs to the domain come from?
(4) Where do outputs from the domain go to?

2. Feature-Oriented Operational Model

The students will draw a feature-oriented operational model illustrating the end user's perspective of the capabilities of applications in the domain [13]. The instructor, acting as the customer, will discuss some of the perceived capabilities of applications in the domain. The instructor will set a time limit on this exercise. The instructor will collect the models developed and evaluate them at a later time. Models developed by the instructor will be handed out when the students' models are collected. The following types of questions will be provided to the students to guide them through this activity:

a. What capabilities does the user believe applications in the domain will have?

b. Are there capabilities not yet defined that may be required in the future?

c. What relationships exist between capabilities identified?

d. If one capability exists, which other capabilities must also exist?

e. If one capability exists, which other capabilities may not exist?

3. Domain Analysis Products

The instructor will provide the students with domain models, a dictionary and thesaurus of the domain-specific terminology, a "generic" requirement specification document, a system feature catalogue and a generic architecture. The instructor will review all of the domain analysis products with the students. The instructor will point out how the generic architecture emphasizes the "standard" concepts in the domain and how it represents many different applications in the domain. The generic architecture provides interface specifications among the components of the domain model.

The instructor will discuss how the generic architecture was developed and how general design and implementation constraints on the architecture were determined and applied to a domain model. In developing the generic architecture software constraints, hardware constraints, performance constraints and general design constraints (e.g., fault tolerance, security, and safety) are all considered. The students must understand the structure of the generic architecture in order to use it.

2.3 DAY 3 - INTEGRATION OF DOMAIN-SPECIFIC SOFTWARE REUSE

2.3.1 Learning Objectives

To achieve system composition using reusable components, the system and software engineers must understand how the various domain analysis products integrate into the system development process.

Upon completion of this lecture the student will be able to:

1. Define domain-specific software reuse library.
2. Explain how a domain-specific library can be used as a reuse tool.
3. Define prototype.
4. Explain how the domain-specific library can be used in defining and in clarifying the application requirements.
5. List three questions used to clarify the requirements.
6. Describe the three steps in instantiating the generic architecture.
7. Define and explain the process of system composition.
8. Explain integration testing of a simple system.

2.3.2 Lecture Content Area

This section will use instructor-provided domain models, system requirements and a domain-specific library to illustrate and support the development of a specific system architecture and a system prototype. This is an intensive hands-on part of the course.

A. Domain-specific Libraries

Use a domain-specific software reuse library as a tool to develop a specific implementation architecture and prototype a system. The instructor will review the library concepts presented earlier and will present a small domain-specific software reuse library which corresponds to the activity exercises. The library will include component browsing and knowledge-based component selection mechanisms. The instructor will provide the students with an easy to follow reference sheet for accessing and using the library. The instructor will review this in detail with the students.

B. Instantiation of the Generic Architecture

Activity: The student will define a specific application architecture. The instructor will provide the students with domain models, system requirements for the application to be developed, a generic architecture, and a domain-specific software reuse library. This activity will illustrate how the domain analysis products can define and clarify application requirements. The instructor will act as an expert in the domain being modeled and as the customer for the application being developed.

The initial set of requirements provided by the instructor will correspond to the generic architecture developed in Day One's activity. The students and the instructor will use the domain-specific software reuse library to further define and clarify the customer requirements. The students will use the browsing mechanism provided by the library to search for information.

1. Explore the Domain-specific Software Reuse Library

   a. What is the application being defined in the requirements?
   b. How will the software reuse library help shape the application design and decision making?
   c. Which components will be likely candidates for reuse?

2. Clarify the Requirements

   Using the software reuse library, the students and the customer (instructor) will clarify ambiguous requirements and isolate differences between the requirements and the generic architecture.

   a. Do the requirements correspond to the generic architecture?
   b. Are there features or functions specified in the requirements that are not represented in the generic architecture of the software reuse library?
   c. Are there features or functions in the generic architecture of the software reuse library that have been left out of the requirements that must be included?
   d. Have all constraints on the system been adequately specified in the requirements?
   e. Do the requirements include conflicts between features (mutual exclusion) which have not been clearly identified?
   f. Do the requirements include relationships between features which have not been clearly identified?
   g. Does development experience exist for a component or project which would suggest the requirements be modified in some way?

3. Revise the Requirements

   The students and the instructor will create a new set of requirements by mapping all of the issues identified in 2. onto the original requirements.

4. Instantiate the Generic Architecture

   a. For each set of alternative features or functions, choose those which correspond to the application requirements.
   b. For each optional feature or function, decide if it will be included in the applications architecture.
   c. What features do not exist in the generic architecture? (Note: These are the components that must be developed and do not exist in the software reuse library).

C. Component Composition

The software components contained within the software reuse library have been designed in a manner that enables them to be reused without detailed knowledge of the code itself. These components can be assembled to create a prototype.

Activity: The students will perform component composition using the software reuse library, the revised requirements and the application architecture defined in the previous section. The students will use the knowledge-based mechanisms provided by the library to identify components. The instructor will act as the customer and evaluate the prototypes created.

1. Collect Components

Using the knowledge-based mechanisms provided by the software reuse library, the students will specify the system requirements and collect components.

a. What are the systems requirements?
b. What are the constraints on the system?
c. Are composition rules [13, 20] recognized that were not identified earlier?

2. Record Issues, Trade-offs and Design Rationale

In cases where the student had to choose between several reusable components, the student will provide a rationale for why one was chosen over another.

a. What were the trade-offs?
b. Did this choice drive future decisions?
c. Did past decisions influence this choice?

3. Link Artifacts

The student will link the components collected from the software reuse library and present a brief demonstration of the prototype to the instructor. The instructor, acting as the customer, will evaluate the prototype and the associated component choice rationale. The instructor will discuss these with the students.

a. Are there problems associated with interfacing any two components taken from the library?
b. What input must be supplied?
c. How much of the system was prototyped?
d. Was it necessary to relax requirement constraints to create the prototype?

4. Integration Testing

The students will test the prototype using a simple set of test suites provided by the instructor.

a. Does the prototype function as required?
b. Does the prototype perform at required levels?
c. What parameters need to be modified to meet system performance requirements?

5. Prototype to Actual System

a. What components would need to be developed and integrated?
b. What changes to the prototype may the customer request?

6. Component Adaptation

a. What components would require adaptation?
b. Are the required changes small?
c. Will it be cost effective to adapt the component instead of developing it from scratch?

2.4 DAY 4 - HOW TO INCORPORATE SOFTWARE REUSE METHODS

2.4.1 Learning Objectives

Upon completion of this lecture the student will be able to:

1. Explain how reuse can be incorporated into their organization's software development processes.
2. List the barriers to reuse.

2.4.2 Lecture Content Area

A. Activity Continued

The students will be allowed several hours in the morning to complete the activity started in Day Three.

B. How Software Reuse Fits their Processes

The instructor will review how software reuse can be incorporated into several of the common software development life-cycle models (e.g., the waterfall model, the spiral model). The students will be asked where they believe software reuse can fit into their current software development processes. The instructor will discuss their processes with them and evaluate them for reuse. Does the student see the integration of software reuse as feasible? What barriers do they foresee? How can the obstacles be broken down?

C. Summary

The instructor will:

1. Summarize the important software reuse activities that have been introduced to the students.
2. Encourage discussion and clarify any issues raised by the students.
3. Stress the lessons learned during the students' hands-on activities.

This summary will reiterate all the important tasks and how they affect the student.

3. GLOSSARY OF TERMS

Analysis: the separation of any material or abstract entity into its constituent elements. This process has a method of studying the nature of something or of determining its essential features and their relations [28].

Application Domain: the knowledge and concepts that pertain to a particular computer application. Examples include battle management; avionic; communication, command, control and intelligence; and nuclear physics.

Architecture: a template that models the system which includes the user interface, input, system function and control, output, and maintenance and self test as elements. The architectural template allows the analyst to create a hierarchy of detail [23].

Architectural Domain: An application domain that is defined by a high level design. It includes all applications that share the common design as well as a set of reusable components that are intended for use in those applications. The architecture defines how the reusable components interact with each other so that they can be designed to work together.

ASR: Ada Software Repository.

Asset: a reusable entity [27].

ASSET: Asset Source for Software Engineering Technology.

CARDS: Central Archive for Reusable Defense Software.

Classification: the process of organizing assets according to a defined classification method [27].

Classification Methods: defined methods of classifying assets supported by a library mechanisms [27].

Classification Scheme: an organizational approach to utilization of classification methods [27].

Component: TBD

Component Composition: TBD

Context diagram: establishes the information boundary between the system being implemented and the environment in which the system is to operate [13].

Control Flow: shows the control information passed throughout the system and associated control processing [23].

COSMIC: TBD

Cost: includes manpower, skills and availability, risk to the schedule budgets and new software and hardware development budgets.

Data Flow: shows the data information passed throughout the system as it moves from input to output [23].

Domain: the set of current and future applications marked by a set of common capabilities and data. An area of activity or knowledge. Domains can be characterized as application, solution, horizontal, or vertical [27].

Domain analysis: the process of identifying objects and operations of a class of similar systems in a particular problem domain; pertains to operational software reuse libraries [27].

Domain-specific Reuse: reuse of ideas, knowledge, artifacts, personnel, and components in an existing domain.

DSRS: Defense Software Reuse System

Entity Relationship Diagram: a modeling tool used to portray the relationship that exist between data stores [19].

GRACE: TBD

Hierarchical Model: a top down view of a system [15].

Horizontal Domain: the knowledge and concepts that pertain to a particular functionality of a set of software components that can be utilized across more than one application domain. Examples include user interfaces, database systems, and statistics. Most *horizontal domains* can be organized as a set of equivalence classes where the distinguishing characteristics are software decomposition style (functional, object-oriented, data-oriented, control-oriented, declarative, etc.), conceptual underpinning, and/or required hardware. One example is subdividing user interfaces into ANSI terminal supporting versus bit-mapped, mouse input supporting.

Library Mechanism: an automated tool that supports classification, search and retrieval of assets [27].

Model engineering: developing abstract representations of what will eventually become a combination of computer hardware and computer software [19].

RAPID: Reusable Ada Packages for Information System Development

Rapid Prototyping: is a process that enables the developer to create a model of the system to be built [23]. The model itself may not evolve into the system.

Reuse-in-the-large: TBD

Reuse-in-the-small: TBD

Software Development: TBD

Software Engineer: software engineers are concerned with how the individual system components are actually designed, developed, and integrated into the total system.

Software Reuse: the process of incorporating into a software system any preexisting component generated at any stage of a system's development.

Solution Domain: the knowledge and concepts that pertain to a particular software solution for providing functionality desired in an application system. Examples includ graphical user interfaces, relational database systems, client-server architectures, and expert system shells.

State Transition Diagrams: a diagram that highlights the time-dependent behavior of the system [19].

System engineer: called a system analyst in some application domains, begins with customer-defined goals and constraints and derives a representation of function, performance, hardware and software interfaces, design constraints, and information structure that can be allocated to each of the generic system elements [23].

Vertical Domain: The knowledge and concepts that pertain to a particular application domain. Vertical domains can be organized as a hierarchy of subdomains that specialize the knowledge and concepts as one moves from the root to the leaves. Most application domains can be organized into a vertical domain hierarchy.

4. INSTRUCTOR

4.1 JOB DESCRIPTION

The instructor is responsible for conducting a tailored, implementation of the recommended course content through a lecture/activity type atmosphere.

This is accomplished by completing the following tasks:

1. Consulting the reference documents.
2. Updating the recommended outline of course content.
3. Preparing appropriate training materials.
4. Conducting the training session.
5. Setting up activities.
6. Ensuring that the atmosphere of the training session is informal and relaxing; intellectually stimulating; and learner-centered.
7. Integrating lecture, discussion, and small working groups into the training session.
8. Leading the course completion criteria workshop.
9. Providing an evaluation of the course completion criteria.

4.2 FORMAL EDUCATION

The instructor should hold at least a BS degree in software/system engineering (an MS degree is preferred) or a degree in computer science or a closely related field.

4.3 KNOWLEDGE OF INSTRUCTION

The instructor should have at least two years experience teaching short courses and should possess an understanding of the principles of learning; the methods of teaching; an ability to apply these principles and methods; and, excellent communication skills.

4.4 PRACTICAL TEACHING EXPERIENCE

The instructor should have at least two years of experience conducting short courses and should possess experience teaching at the collegiate level (at least two years); planning short courses; and, should possess experience addressing system engineer personnel.

4.5 KNOWLEDGE OF SUBJECT

The instructor should possess experience in the system and software engineering processes; knowledge and an understanding of the impact of integrating software reuse into these processes; formal training in the form of a workshop or seminar on domain analysis and domain-specific software reuse; and, work experience as system engineer or software engineer.

4.6 SKILL IN PERFORMANCE

The instructor should be able to demonstrate the ability to perform the tasks required in the practical application of the training course and have previous experience in conducting a domain analysis.

APPENDIX A References

[1] J. Tomayko (ed). Software Engineering Eduction, SEI Conference 1991, Springer-Verlag, October, 1991.

[2] J. Withey. Model-Based Engineering, SEI - Tutorial slides, March 1992.

[3] T. Biggerstaff and A. Perlis (eds). Software Reusability, Volume I, Concepts and Models, ACM Press, Frontier Series, 1989.

[4] T. Biggerstaff. Topics In Reuse and Design, IEEE - Video, 1991.

[5] Y. Matsumoto. "Some Experience in Promoting Reusable Software: Presentation in Higher Abstract Levels", IEEE Transactions on SOFTWARE Engineering, Vol. SE-10, NO. 5, September 1984, pp. 502-513.

[6] I. Sommerville, Software Engineering, Addison-Wesley, Third Edition, Publishing Company, 1989.

[7] W. Tracz. Tutorial: Software Reuse: Emerging Technology, IEEE Computer Society Press, 1990.

[8] N. DiTomaso and G. Farris. "Diversity in the High-Tech Workplace", IEEE Spectrum, June 1992, pgs. 21-32.

[9] T. Bell, (ed). " '90s employment: some bad news, but some good", IEEE Spectrum, December 1990, pgs. 32-43.

[10] J. Hooper and R. Chester. Software Reuse Guidelines, AIRMICS, December 13, 1989.

[11] W. Frakes, R. Prieto-Diaz and R. Arnold. Software Reuse, Domain Analysis, and Reengineering, A Three Day Seminar, Reston, Virginia, April 6-8, 1992.

[12] E. Awad. Systems Analysis and Design, Richard D. Irwin, Inc., 1985.

[13] S. Cohen. Modeling Software Reuse Technology: Feature Oriented Domain Analysis (FODA) - Tutorial slides, SEI, Carnegie Mellon University, May 1992.

[14] D. Gause and G. Weinberg. Exploring Requirements Quality Before Design, Dorset House Publishing, 1989.

[15] R. Glass. Building Quality Software, Prentice Hall, 1992.

[16] H. Gomaa, L. Kerschberg, C. Bosch, V. Sugumaran and I. Tavakoli. "A Prototype Software Engineering Environment For Domain Modeling and Reuse, Sixteenth Annual Software Engineering Workshop", NASA Goddard Software Engineering Laboratory, December, 1991.

[17] R. Prieto-Diaz and G. Arango (eds). Domain Analysis and Software Systems Modeling, IEEE Computer Society Press, May 1991.

[18] W. Tracz and L. Coglionese. Domain-Specific Software Architecture Engineering Process Guidelines (Working Document), ADAGE-IBM-92-02, Version 0.1, IBM Corporation, March 17, 1992.

[19] E. Yourdon. Modern Structured Analysis, Yourdon Press Computing Series, 1989.

[20] P. Feiler. Configuration Management Models in Commercial Environment, SEI-91-TR-7, SEI, Carnegie-Mellon University, March 1991.

[21] R. Fairley and P. Freeman. Issues in Software Engineering Education, Springer-Verlag, 1989.

[22] P. Freeman. Tutorial: Software Reusability, IEEE Computer Society Press, 1987.

[23] R. Pressman. Software Engineering A Practitioner's Approach, Second Edition, McGraw-Hill Book Company, 1987.

[24] W. Tracz. "Ada Reusability Efforts: A Survey of the State of the Practice", Proceedings of the Joint Ada Conference, Fifth National Conference on Ada Technology and Washington Ada Symposium, US Army communications-electronics Command, Fort Monmouth, New Jersey, pp. 35-44.

[25] V. Basili, G. Caldiera, and G. Cantone. "A Reference Architecture for the Component Factory", ACM Transactions on Software Engineering and Methodology, Vol1, No. 1, January 1992, pgs 53-80.

[26] R. Prieto-Diaz. "Implementing Faceted Classification for Software Reuse", Communications of the ACM, May 1991, Vol. 34, No. 5, pgs 89-97.

[27] Draft RIG_TC1 Glossary, Version 1.0, January 22, 1992.

[28] The Random House College Dictionary, Revised Edition, Random House, Inc., 1988.

APPENDIX B Implementation Questions and Answers[14]

1. What is a reasonable size for the activity projects?

Due to the limited time of the course, very small projects will be presented. The project should not be an example from the domain in which the students work. This will help to keep the students focused on the methods being presented and not on the example itself. The example will be small and yet large enough to expose the students to all the concepts and methods they will require to incorporate software reuse into their software development processes.

2. What is the role of the instructor in the course?

The instructor must provide project description, assistance in assignments, guidance during project executions, lectures on software reuse, software reuse libraries, domain analysis, generic architecture development, specific architecture development and system composition, and activity evaluations. Instructor intervention may be needed to direct students in the right direction during activity exercises. To minimize the risk of student failure during the activity exercises, it is necessary for the instructor to be very comfortable with the exercises being presented and their solutions. The instructor will be prepared for students to encounter problems. To ensure coordination between lecture and labs there will be just one instructor.

3. How are the activity exercises evaluated?

Upon completion of each lab, the instructor will review the work of the students. The labs will not be graded. The instructor will provide the students with comments, where appropriate, indicating the areas they need to give more attention to and those they did well. Upon returning the labs to the students, the instructor will also provide the students with a "solution" to the lab. The instructor will discuss the activity exercise with the students before moving on to the next activity to ensure that all students are relatively comfortable with the methods presented in the previous lab. Each activity will build upon the products of the previous labs.

4. How can the instructor ensure that all students get off to a fast start on the activity exercises?

Preliminary preparation for the activity exercises by the instructor is critical to the overall success of the course. All materials required by the students to complete the exercises must be prepared and thoroughly reviewed by the instructor. The instructor must be certain that all required equipment is present and operational. A fast start is essential if the students are to complete the activity assignments in the given timeframe. To minimize the time it takes for a student to get started on the assignment, the instructor will have all materials ready for the student on the first day of the course. The students will then have some time to consider the exercise before beginning it. The instructor will be prepared to give considerable help to all students in the critical first key steps of the activity exercises. Specifically, the instructor may choose to make some of the activities group exercises and some individual exercises. This will keep slower students from falling too far behind.

APPENDIX C Course Evaluation Form[14]

Please complete the following course evaluation form .

1. Name (Optional)

2. Organization

3. I perform the following tasks as part of my job responsibilities (indicate all that apply):

specify systems/software	design systems/software
code software	test systems/software
verify systems/software	control systems/software
use software	manage systems/software
other___________	

4. I will be able to apply the course material to my job duties:
   yes no

5. Before I attended this course, I was familiar with (Level of knowledge - novice to expert):

   Structured Analysis Techniques
   Model-based Engineering
   Domain Analysis
   Software Reuse Libraries
   Domain Specific Software Reuse Libraries
   Generic Architecture Development
   System Composition

6. The length of this class was:
   too long too short just right

7. The material covered was:
   too specific too general adequate

8. The domain analysis activity exercise was a valuable part of the course:
   yes no

9. The architecture development and prototyping activity was valuable:
   yes no

10. The examples helped clarify the concepts:
    yes no

11. The domain-specific software reuse library was easy to use:
    yes no

12. The course handouts were useful:
    yes no

13. I would recommend attendance to my colleagues:
    yes no

14. I have attended other software reuse training classes or seminars:
    yes no

15. I would be interested in attending other software reuse training seminars:
    yes no

16. I would be interested in receiving additional information on software reuse:
    yes no

17. I believe domain-specific software reuse is very valuable in the development of software systems:
    yes no

18. Additional Comments:

APPENDIX D Possible Activity Exercises

Following is a list of possible subject domain areas for activity exercises. In developing the activity exercises it is essential that the domain be small. The time-frame provided by this course description does not allow for detailed activity exercises.

1. Automatic Teller Machines (ATMs)
2. Banking Systems
3. Hospital Patient Monitoring Systems
4. Text Editors (Word Processing Systems)
5. E-mail Systems
6. Calculator Programs
7. Menu Manager Systems
8. Elevator Systems
9. Walkman Devices
10. Warehouse Systems
11. Flight Reservation Systems

APPENDIX E Course Equipment

This section will contain a listing of equipment to support activity exercises, such as overhead projectors and computer equipment.

TBD

APPENDIX F Course Materials

This section will contain an example set of handouts, slides and library reference documentation to support both the lecture and the activity exercises of a sample domain.

TBD