Publication 3057

ftware Reuse Issues

edings of a workshop held in
Melbourne, Florida
November 17-18, 1988

NASA

NASA Conference Publication 3057

Software Reuse Issues

Edited by
Susan J. Voigt
and Kathryn A. Smith
NASA Langley Research Center
Hampton, Virginia

Proceedings of a workshop sponsored by
NASA Langley Research Center,
Hampton, Virginia, and held in
Melbourne, Florida
November 17-18, 1988

NASA
National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Division
1989

PREFACE

The Workshop on NASA Research in Software Reuse was held November 17-18, 1988, in Melbourne, Florida. The workshop was sponsored by the Systems Architecture Branch, Information Systems Division, NASA Langley Research Center and hosted by Software Productivity Solutions, Inc., Indiatlantic, Florida. The workshop was held to permit NASA researchers in software reuse to share their plans and to learn about other current research activities of direct interest. Space Station Freedom Software Support Environment project personnel were also invited to attend to describe their plans to support software reuse in the Space Station Freedom Program.

Several presentations and demonstrations were given by Software Productivity Solutions, Inc., on their work related to Eli, the reusable software synthesis system under development with a NASA Small Business Innovative Research contract.

A list of issues was developed during the workshop discussion session, and a few recommendations were made related to experimentation and data collection within NASA software reuse applications.

Susan J. Voigt and Kathryn A. Smith
NASA Langley Research Center

PRECEDING PAGE BLANK NOT FILMED

CONTENTS

INTRODUCTION

NASA Langley Research Center sponsored a Workshop on NASA Research in Software Reuse on November 17-18, 1988, at the Quality Suites in Melbourne, Florida. The workshop was hosted by Software Productivity Solutions, Inc. (SPS) and led by Susan Voigt of NASA Langley. Participation was by invitation only and included representatives from four NASA centers and headquarters, eight NASA contractor companies, and three research institutes.

The primary purpose of the workshop was to share information and plans for software reuse research at the four NASA centers participating in the Office of Aeronautics and Space Technology (OAST) "NASA Initiative in Software Engineering." Other objectives were to identify areas for cooperative and collaborative research among the NASA centers; to provide NASA researchers an opportunity to learn about Eli, the reusable software synthesis system designed by SPS and being developed under a Small Business Innovative Research (Phase II) contract with Langley Research Center; and to expose Space Station Freedom Software Support Environment developers to NASA research activities in reuse.

In addition to NASA researchers from Goddard Space Flight Center, Johnson Space Center, Langley Research Center, and the Jet Propulsion Laboratory, a few representatives from the Space Station Freedom Program and outside research groups were invited to attend and to make presentations that would be of interest to the NASA reuse research community. A list of workshop participants is included in this document. This publication summarizes the presentations made and the issues discussed during the workshop.

SUMMARY OF WORKSHOP PRESENTATIONS

This section contains a brief summary as well as the viewgraphs of each of the presentations made during the workshop. The order follows that of the workshop agenda.

Talks on related topics which were grouped together include SPS developments, research activities at the participating NASA centers, plans for software reuse in the Space Station Freedom Program, and some information about AdaNET. Invited talks were also given by representatives of the Software Productivity Consortium, the Software Engineering Institute, and a NASA Johnson Space Center contractor, Inference Corporation.

The first four presentations were given by members of the Software Productivity Solutions, Inc. (SPS) staff. The company has several activities related to software reuse including both research and product development. Since SPS had recently completed a NASA Small Business Innovative Research contract that included a broad survey of technologies to support reuse and currently is developing several elements of a software development system incorporating reuse, they were asked to host this workshop and present their key findings to the NASA research community.

Four NASA centers are participating in the NASA Initiative in Software Engineering Program under the Office of Aeronautics and Space Technology computer science research program. The research activities in this initiative are focused on software engineering for reliable complex systems, and each of the Center programs has an element in software reuse with various approaches to reduce complexity or increase reliability. The main objective of this workshop was the exchange of information among researchers from the four centers.

In the NASA Space Station Freedom Program, the Software Support Environment (SSE) is a bold step toward software reuse. It is the set of tools and rules to be used by all developers of software for the Space Station Freedom. The intent is to reduce life cycle costs of software by avoiding duplication of similar tools needed by many contractors as well as to make the integration and maintenance of software systems much easier. As part of its toolset, the SSE will provide library support for application software reuse. Space Station software representatives were invited to participate in this workshop to present their plans and requirements to the NASA research community and to explore ways for research results to help the SSFP.

Software reuse projects are active at both the Software Productivity Consortium, a software engineering research organization supported by 14 aerospace companies, and the Department of Defense Software Engineering Institute at Carnegie Mellon University. The approaches being taken in these organizations and the issues they are addressing were of interest to the NASA research community.

The advanced software development workstation project, funded by NASA, is coupling reuse support tools with an advanced graphical workstation. Progress and future plans on this work were summarized for the workshop participants.

AdaNET is an electronic distribution network for software engineering information and parts exchange. It is partially supported by the NASA Office of Technology Utilization. Representatives from the AdaNET project were invited to participate in the workshop to describe their plans and activities and how these might relate to the NASA research program.

N90-14790

AUTOMATED REUSABLE COMPONENTS SYSTEM STUDY RESULTS

Kathy Gilroy
Software Productivity Solutions, Inc.

The Automated Reusable Components System (ARCS) was developed under a Phase I Small Business Innovative Research (SBIR) contract for the U.S. Army CECOM. The objectives of the ARCS program were (1) to investigate issues associated with automated reuse of software components, identify alternative approaches, and select promising technologies, and (2) to develop tools that support component classification and retrieval. The approach followed was to research emerging techniques and experimental applications associated with reusable software libraries, to investigate the more mature information retrieval technologies for applicability, and to investigate the applicability of specialized technologies to improve the effectiveness of a reusable component library. Various classification schemes and retrieval techniques were identified and evaluated for potential application in an automated library system for reusable components. Strategies for library organization and management, component submittal and storage, and component search and retrieval were developed. A prototype ARCS was built to demonstrate the feasibility of automating the reuse process. The prototype was created using a subset of the classification and retrieval techniques that were investigated. The demonstration system was exercised and evaluated using reusable Ada components selected from the public domain. A requirements specification for a production-quality ARCS was also developed.

AUTOMATED REUSABLE COMPONENTS SYSTEM (ARCS)

Objectives

o Investigate issues associated with automated reuse of software components, identify alternative approaches and select promising technologies
  - classification criteria
  - library organization
  - retrieval techniques

o Develop tools to support component classification and retrieval activities
  - develop demonstration system in Phase I
  - define the requirements for a production-quality system to be developed during Phase II

CLASSIFICATION AND RETRIEVAL RESEARCH

Approach

o Research emerging techniques and experimental applications associated with reusable software libraries

o Investigate the more mature information retrieval (IR) technologies for applicability to the reusable software problem

o Investigate the applicability of specialized technologies (e.g., expert systems, semantic networks, fuzzy logic) to improve the effectiveness of a reusable component library

REUSABLE COMPONENT LIBRARY SYSTEM ROLES

o Classification - the process of entering the component into the library

o Retrieval - the process of finding an applicable component to meet a perceived need

REUSABLE COMPONENT CLASSIFICATION

1. Understand the component

2. Certify the component

3. Classify the component based upon knowledge of the classification strategy

4. Insert the component into the library

REUSABLE COMPONENT RETRIEVAL

1. Access the component by stating the need in terms compatible with the classification system
2. Understand the component
3. Evaluate the component for applicability and acceptability
4. Adapt the component for the particular application
5. Integrate the component into the baseline system under development

Classification Issues

o Classification criteria - those attributes of components that can be used to classify, understand and evaluate them.

o Classification organization - the mechanism by which components are logically organized in the library according to the classification criteria.

Classification Criteria

o Must support both searching and discrimination

o May be static or dynamic

o Classification criteria may be composed of:

- Key words associated with its function, purpose or application area
- Text description
- Characteristics or metrics of interest
- Language or other structured description

Classification Organization

o Enumerative organizations

o Hierarchical taxonomies

o Faceted schemes

o Semantic nets

o Clustered organizations

Query Logic

o Deterministic logic that retrieves based upon exact matches according to a Boolean query.

o Probabilistic logic that estimates the probability of relevance of specific components in the library.

o Fuzzy logic that uses weighted or graded measures to assess whether a component meets user query criteria.

Query Enhancement

A modification or enhancement of an original query in order to expand or refine the retrieval

o Query Generalization, required when there are too few finds or when the finds that are retrieved are "near- misses".

o Query Specialization, required when user is confronted with too many finds, or when most of the components that are retrieved are non-relevant.

Query Enhancement Application

o Query enhancement experiments in IR have not demonstrated improvements in retrieval effectiveness, and, in fact, demonstrate degradation in many cases

o A compromise to automated query expansion is to calculate an ordering of finds and to use the ordering to present the "best fit" or most relevant to the user first

o Relevance feedback has shown the most promise

Classification and Retrieval Conclusions

No single scheme is best -- employ a number of technologies, adapting and borrowing from database, information retrieval and knowledge-engineering disciplines

o Classification criteria: All types - key words, text, characteristic-based or metrics and languages

o Library classification organization: Faceted, later enhanced with clustering

o Query logic: Deterministic, later enhanced with probabilistic or fuzzy logic

Flexibility is important in improving effectiveness!

ARCS Prototyping Objectives

o Experiment with a faceted classification approach and with supporting multiple classification schemes

o Evaluate candidate criteria for usefulness in retrieval, evaluation and understanding

o Prototype the user interface to improve usability of the production ARCS

o Demonstrate the applicability of an Entity-Relationship database approach

o Support the formulation of requirements for the production ARCS

o Determine areas where more research is needed

Demonstration ARCS Tool

o Implemented entirely in Ada

o Hosted on VAXstation running VMS

o Employed a number of existing components and subsystems

- WINNIE (windowing/menu system)
- SMARSTAR/Rdb (relational database management system)
- Ada Entity-Relationship Interface to database subsystem
- Numerous low level data management components

ARCS Operations

o View the catalog information stored about a specific component existing in the ARCS database.

o Add a catalog entry for a new component, and insert its source code and test cases into the database. This information is then controlled by the ARCS much like checking-in and checking-out information from a CM system.

o Update the catalog information for a specific component.

o Delete all information about an obsolete component from the database.

o Extract a specified component from the database. The sources, tests, and/or catalog information can be copied to a user-specified VMS directory.

o Select (find) components which match search constraints on the values of component characteristics. The components so selected may then be viewed or extracted, or the selection criteria may be modified to improve the results of a subsequent search.

ARCS User Interface

- Consistency, on-line help, shortcuts (data entry still a burden)
- Menus and forms for component attribute update and query
- Supported by windowing and menu organization subsystems
- Attribute-based queries supported by simple query language:

$$[\text{not}]\quad [\text{qualifier}]\quad \text{value}\quad \{\ \overset{\text{and}}{\text{or}}\ [\text{not}]\quad [\text{qualifier}]\ \text{value}\ \}$$

ARCS Database

o Metaschema subschema defines the "super-structure" for the ARCS database, representing the ER model itself.

o Component subschema defines the entities, relationships and attributes containing all of the catalog information about each reusable component.

o Classification subschema defines the entities, relationships and attributes comprising the means for classifying components in different ways.

Implementation Issues Raised and Evaluation Results

o Attribute/criteria selection

o Population of the classification subschema

o Deferred support for certain policies

o Performance Issues

o Data Entry Issues for Usability

CONCLUSIONS

o There are sufficient underlying database, IR and knowledge-based technologies on which to develop a production ARCS

o The Phase I research successfully derived a flexible, extensible faceted approach for ARCS and identified promising technologies for further investigation

o The Phase I demonstration system reinforced the validity of the overall approach, while pointing out areas for future investigation

o Additional work is needed to determine the specific classification criteria and classification schemes

o Additional experimentation is needed to address the tradeoffs associated with ease-of-use, performance, applicability and effectiveness

N90-14791

KNOWLEDGE-BASED REUSABLE SOFTWARE SYNTHESIS SYSTEM

Cammie Donaldson
Software Productivity Solutions, Inc.

The Eli system, a knowledge-based reusable software synthesis system, is being developed for NASA Langley under a Phase II SBIR contract. Named after Eli Whitney, the inventor of interchangeable parts, Eli assists engineers of large-scale software systems in reusing components while they are composing their software specifications or designs. Eli will identify reuse potential, search for components, select component variants, and synthesize components into the developer's specifications. The Eli project began as a Phase I SBIR to define a reusable software synthesis methodology that integrates reusability into the top-down development process and to develop an approach for an expert system to promote and accomplish reuse. The objectives of the Eli Phase II work are to integrate advanced technologies to automate the development of reusable components and the use of reusable components within the context of large system developments, to integrate with user development methodologies without significant changes in method or learning of special languages, and to make reuse the easiest operation to perform. Eli will try to address a number of reuse problems including developing software with reusable components, managing reusable components, identifying reusable components, and transitioning reuse technology. Eli is both a library facility for classifying, storing, and retrieving reusable components and a design environment that emphasizes, encourages, and supports reuse. Eli is being developed incrementally and will be released in a series of builds with progressively more functionality. A related issue, not being addressed by the Eli project, is how to implement reuse within an organization.

Outline of Presentation

- Eli Project Background
- Problems that Eli Will Solve
- Overview of Eli Build Plan
- Some Eli Operational Issues

Eli Project Background

- Phase I completed in Fall 1987, objectives were to:
  - Define reusable software synthesis methodology that integrates reusability into the top-down development process
  - Investigate formal languages for specifying reusable component interfaces, operations and requirements
  - Investigate knowledge and database representations for organizing and storing both components and knowledge of the application domain and development process
  - Develop approach for expert system to promote and accomplish reuse

Eli Project Background (Conc)

- Phase II started in July 1988; objectives are to:
  - Integrate advanced technologies to automate the development of reusable components and the use of reusable components within the context of large system developments
  - Integrate with user development methodologies without significant changes in method or learning of special languages
  - Make reuse the easiest operation to perform

Problems That Eli Will Solve

What Reuse Problems Must Eli Address?

- Developing software with reusable components
- Managing reusable components
- Identifying reusable components
- Transitioning reuse technology

What is Eli?

- Library facilities for classifying, storing and retrieving reusable components
- Design environment that emphasizes, encourages and supports reuse

User Roles

- Eli will support the following user roles:
  - Classifier
  - Searcher
  - Promoter
  - System Administrator

Key Qualities of Eli

- Adaptability
- Performance
- Ease of Use

• • *Make reuse the easiest operation to perform* • •

How Will Eli Solve Reuse Problems?

Identifying Reusable Components

- Flexible component classification facilities
- Flexible browsing and querying facilities

Managing Reusable Components

- Efficient storage and retrieval of large component inventories
- Open architecture to support integration with user environment
- Facilities for tracking and promoting reuse activities

Developing Software With Reusable Components

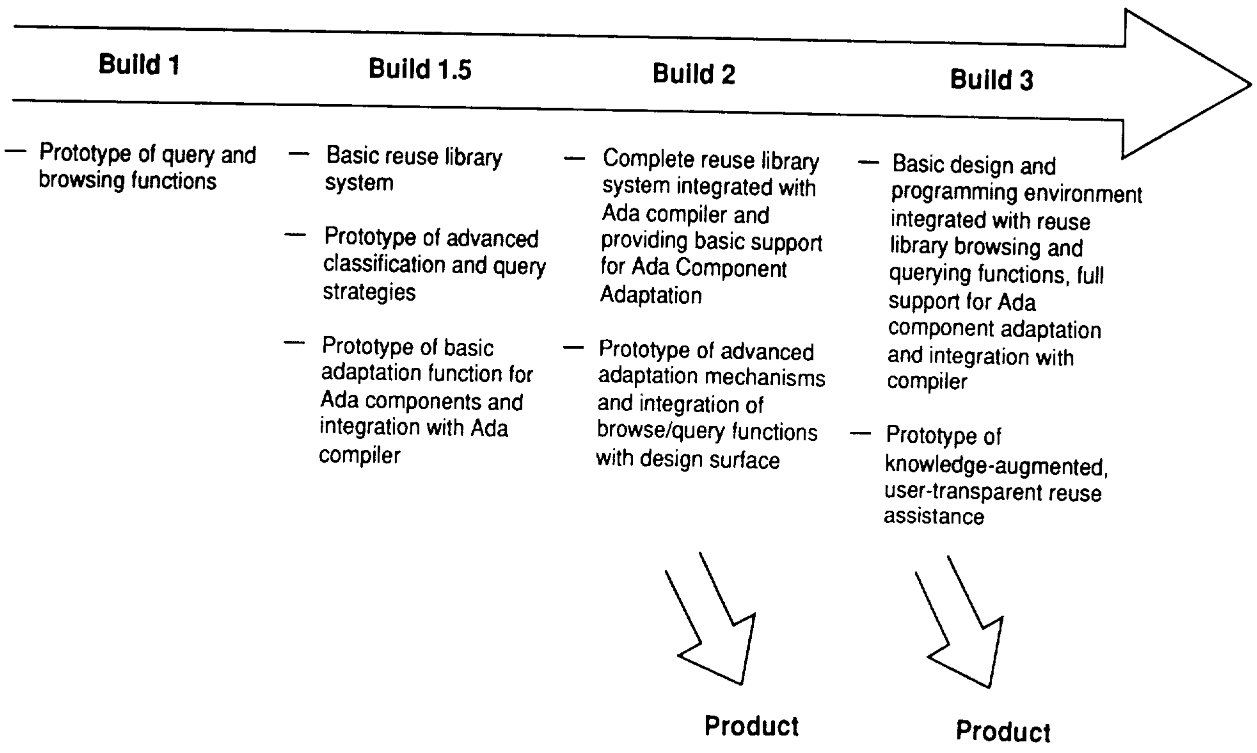
- Direct support for Ada components, including adaptation and integration
- Support for object-oriented design and programming
- Integration of design surface with library facilities

Transitioning Reuse Technology

- Support for defining new types of components, new component characteristics and new component relationships
- Loose and tight integration capabilities to transition existing tools and information

Overview of Eli Build Plan

Build Plan

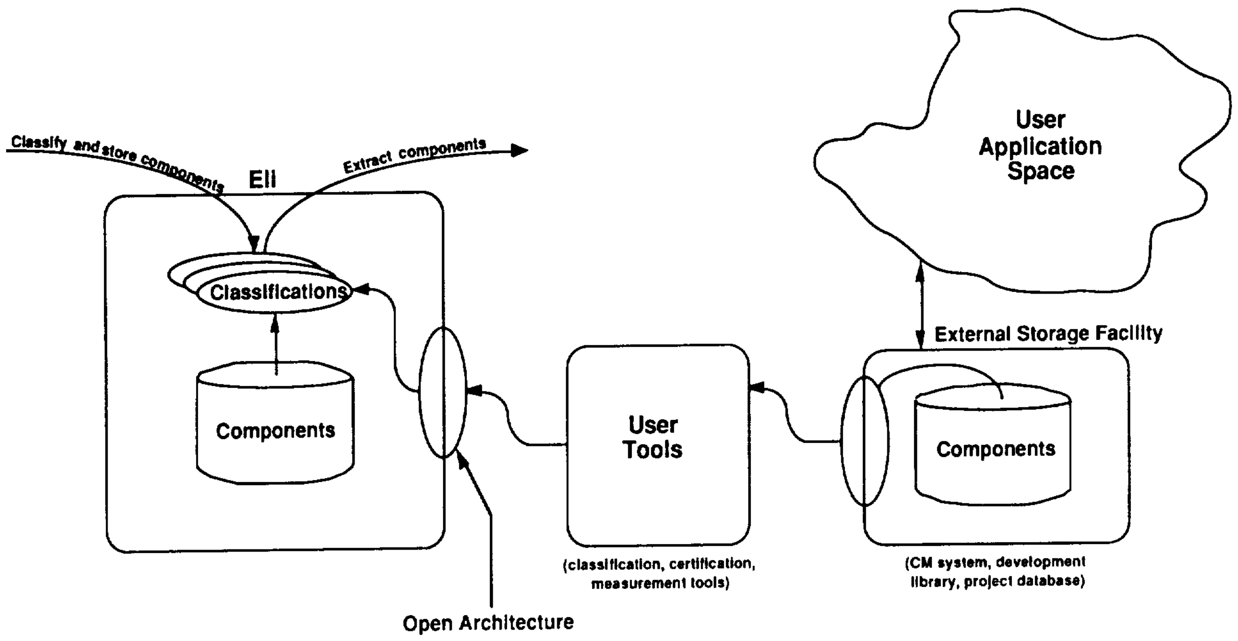


Build 1	Build 1.5	Build 2	Build 3
— Prototype of query and browsing functions	— Basic reuse library system — Prototype of advanced classification and query strategies — Prototype of basic adaptation function for Ada components and integration with Ada compiler	— Complete reuse library system integrated with Ada compiler and providing basic support for Ada Component Adaptation — Prototype of advanced adaptation mechanisms and integration of browse/query functions with design surface	— Basic design and programming environment integrated with reuse library browsing and querying functions, full support for Ada component adaptation and integration with compiler — Prototype of knowledge-augmented, user-transparent reuse assistance
		Product	Product

Build 1.5

- This build will provide basic library capabilities:
  - Creation and maintenance of libraries
  - Creation and maintenance of classification schemes for library components
  - Classification and storage of components
  - Browsing of libraries to find/identify components
  - Querying on libraries to find/identify components
  - Extraction of classification schemes, components and component information
  - Integration of component classification, storage, query and extraction functions through a program interface

Build 1.5/2


Classify and store components
Extract components
Eli
Classifications
Components
User Application Space
External Storage Facility
User Tools
(classification, certification, measurement tools)
Components
(CM system, development library, project database)
Open Architecture

Build 2

- This build will provide a complete, sophisticated library system:
  - Import/export of libraries and classification schemes
  - Enhanced manipulation of classification schemes and component classifications
  - Semi-automated derivation of Ada component characteristics
  - Classification support for Classic-Ada components
  - Clustering of components and support for "like this" querying
  - Enhanced and additional forms of interactive browsing and querying on component characteristics
  - Storage, retrieval and modification of query sessions, including batch submittal of queries and query sessions

Build 2 (Conc)

- Version control on libraries, classification schemes, components and component information
- Access control to libraries, classification schemes, components and component information
- Adaptation and integration of reusable Ada components with user application
- Collection and reporting on library and classification scheme usage, and component submittal and extraction
- Customization and tailoring capabilities

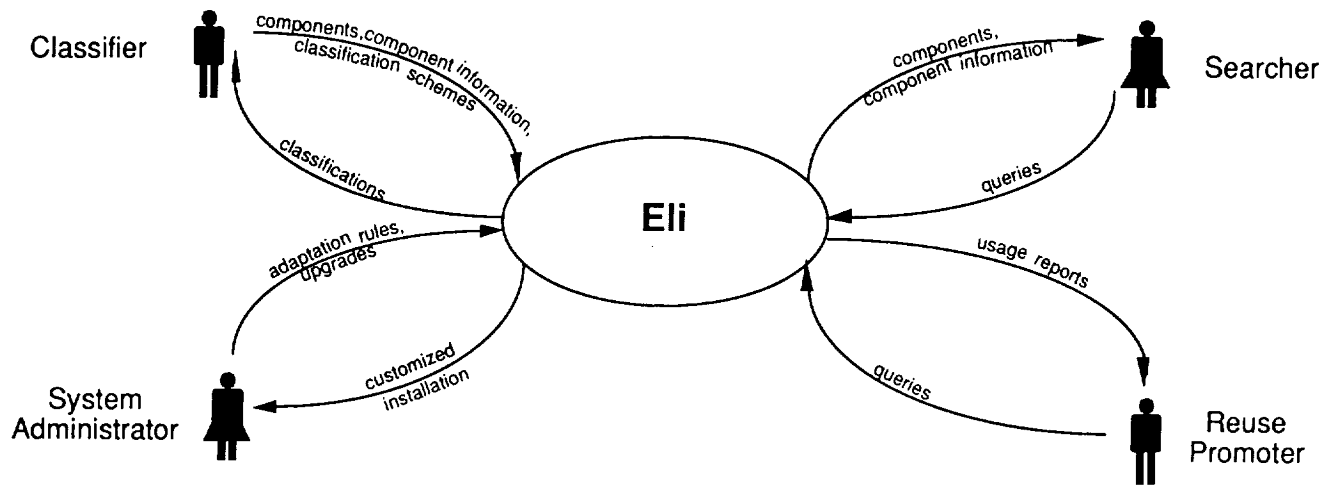
Build 3

This build will provide an object-oriented design surface with the following capabilities:

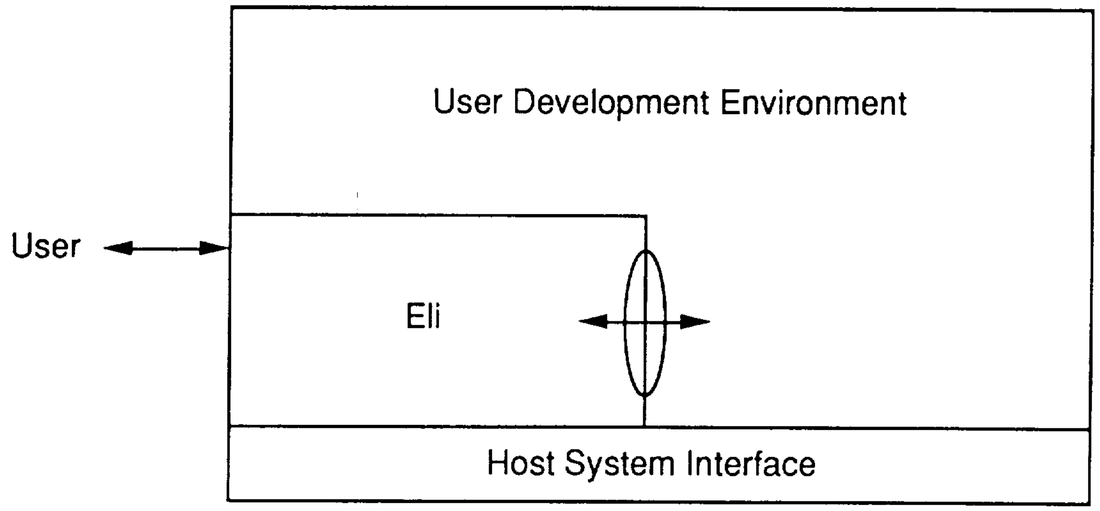
- Integration with Eli library facilities for design-time reuse assistance
- More automated derivation of component characteristics and classification of components
- Ordered assessments of components identified as result of queries
- Advanced support for Ada component adaptation and integration

Some Eli Operational Issues

User Roles



Eli "Black Box" View



Eli Interface Requirements

Interface Area	Principal Eli Focus
• Host Operating System	Transportability
• User's Development Environment Framework	Interoperability
• User's Development Environment Tools	Interoperability
• User's Development Environment Policies, Procedures and Methods	Adaptability

Eli Host Operating System Interfaces

Approach: Establish localized internal interfaces and utilize industry standards (e.g. Unix, XWindows, TCP/IP, Postscript) for transportability

- Device management
- Process Management
- File Management
- Communications

Eli Interfaces to User's Development Environment Framework

Approach: Support many levels of interaction including an open architecture - - procedural access to internal Eli facilities, published information schemas/structures, and an ASCII import/export interchange mechanism.

- Eli invocation
- Import of environment roles, access rights, procedures, etc.
- Configuration management of components
- Ada library manager
- Environment information management facilities
- Invocation of other environment tools/facilities

Eli Interfaces to User's Development Environment Tools

Approach: Provide open architecture - - procedural access and ASCII import/export facilities to allow users to exchange information with other tools

- Ada compilation system
- Documentation tools
- Other CASE (i.e. design surface) tools
- Other reuse systems (e.g. libraries, domain analysis tools)
- Project management tools

Eli Interfaces to User's Development Environment Policies, Procedures and Methods

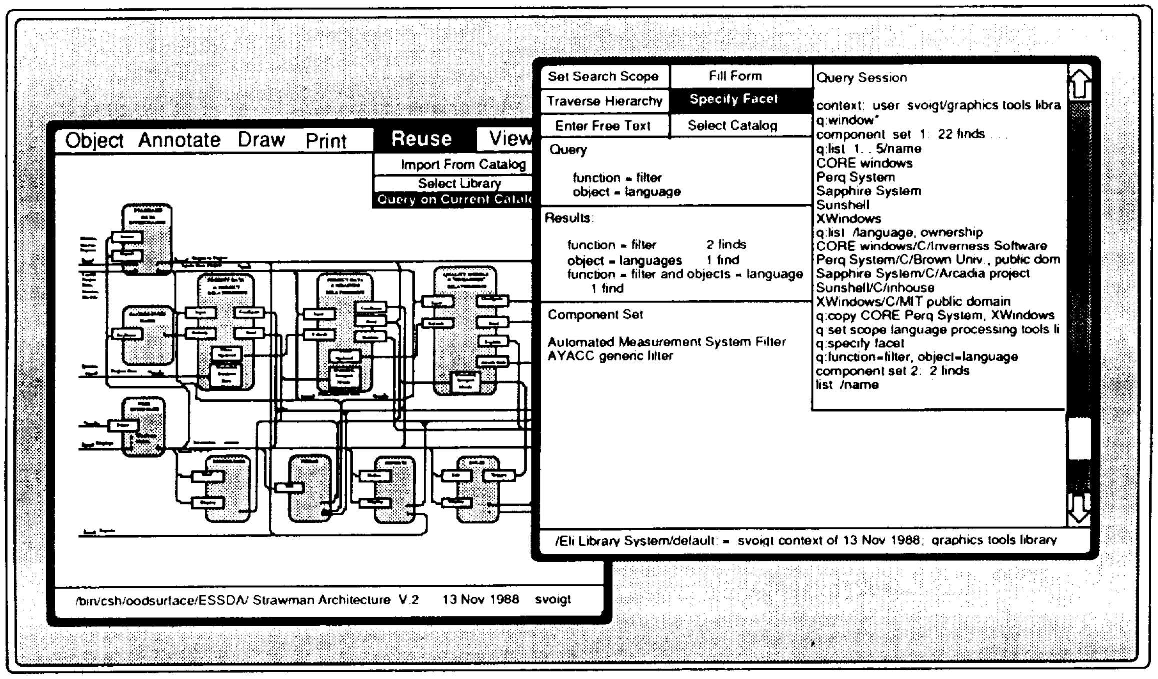
Approach: Make Eli facilities adaptable to accommodate a wide spectrum of usage

- User roles and access rights
- Usage scenarios/sequences/work flows
- Configuration management procedures
- Component certification procedures
- Custom component attributes/facets
- Custom classification schemes
- Site/library installations

Eli Distribution Options

	Classification Update (Library Control)	Component Classification & Storage	Library Access
Non-distributed library model	Local	Local	Local only
Interaction of remote, separately controlled libraries (e.g., interlibrary loan)	Local	Local	Local plus protocol or accessing remote libraries
Master/branch library (e.g. , bookmobile)	Local to master library	Local to master library	Accessible across affiliated branches
Partitioned library (e.g., library system)	Single point or negotiated	Partitioned	Accessible across library sites
Cooperating, distributed libraries	Distributed	Distributed	All libraries accessible transparently from any site

Library Interaction Through Design Surface



```
Query

    function = filter
    object = language

Results:

    function = filter          2 finds
    object = languages         1 find
    function = filter and objects = language
        1 find

Component Set

Automated Measurement System Filter
AYACC generic filter
```

```
Query Session

context: user svoigt/graphics tools libra
q:window*
component set 1: 22 finds . . .
q:list 1. . 5/name
CORE windows
Perq System
Sapphire System
Sunshell
XWindows
q:list /language, ownership
CORE windows/C/Inverness Software
Perq System/C/Brown Univ., public dom
Sapphire System/C/Arcadia project
Sunshell/C/inhouse
XWindows/C/MIT public domain
q:copy CORE Perq System, XWindows
q:set scope language processing tools li
q:specify facet
q:function=filter, object=language
component set 2: 2 finds
list /name
```

```
/Eli Library System/default: = svoigt context of 13 Nov 1988; graphics tools library
```

```
/bin/csh/oodsurface/ESSD/V Strawman Architecture V.2    13 Nov 1988    svoigt
```

ORIGINAL PAGE IS
OF POOR QUALITY

N90-14792

IMPACT OF DOMAIN ANALYSIS ON REUSE METHODS

Kathy Gilroy
Software Productivity Solutions, Inc.

SPS is performing a study for the US Army CECOM on the impact of domain analysis on reuse methods. Domain analysis is the first activity that should be performed in the development of reusable software. It identifies the commonalities between systems within a given problem domain (such as navigation systems or database management). In the software arena these commonalities are implemented as software components that can be reused by new systems within that application domain. The objectives of the study are to develop an approach that makes domain analysis practical and effective for the Army, to reinforce the importance of domain analysis for software reuse programs, and to summarize and coalesce domain analysis information into a single reference source. Existing methods and tools are being analyzed, critical issues identified, and key automation issues addressed. Based on these, a methodology and set of guidelines for domain analysis are being developed. Potential automated tools will be identified for each activity in the methodology.

Problem Statement

What is Domain Analysis?

- The first activity which should be performed in the development of reusable software
- Identifies commonalities between systems within a given problem domain
- Commonalities implemented as software components reused by new systems within that domain

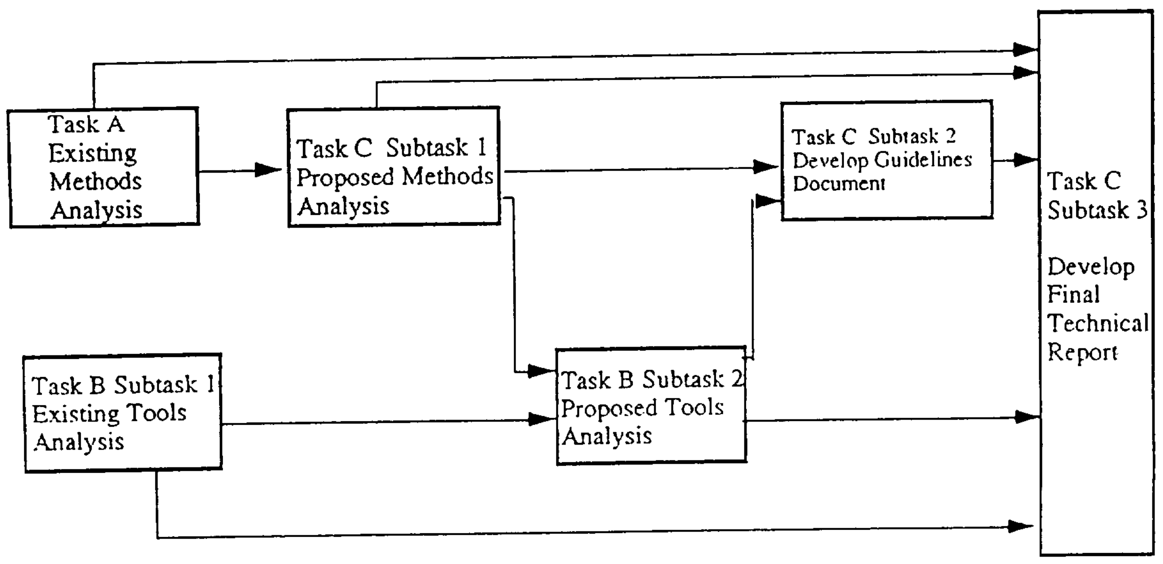
Little Data Available About Domain Analysis

- Few have been done to date
- Importance only recently identified
- Process is difficult and expensive
- Potential payoff not yet known
- "Bad" analysis decreases ability to reuse
- Well-defined methods non-existent
- "Goodness" criteria non-existent
- Recent ad hoc efforts provided little feedback
- No tools support entire analysis process

Objectives

- Develop approach that makes domain analysis practical and effective for Army
- Reinforce importance of domain analysis for software reuse program
- Summarize and coalesce domain analysis information into single reference source

Approach


Task A Existing Methods Analysis
Task C Subtask 1 Proposed Methods Analysis
Task C Subtask 2 Develop Guidelines Document
Task C Subtask 3 Develop Final Technical Report
Task B Subtask 1 Existing Tools Analysis
Task B Subtask 2 Proposed Tools Analysis

Task Interrelationships

Approach

Existing Methods Analysis

- Identify criteria for evaluating domain analysis approaches
- Survey existing methods for domain analysis and relate to criteria
- Develop description of desirable characteristics of a domain analysis
- Identify critical issues and assess risks involved

Identify Critical Issues

- Development methods
- Development languages
- Development tools
- Development personnel
- Application systems
- Domain analysis techniques
- Evaluation and validation
- Domain maintenance

Approach

Address Key Automation Issues

- Expert knowledge acquisition and use
- Domain analysis products standardization
- Data organization, storage and retrieval
- Reusable component library interfaces
- Integration with software development environments and reuse tools
- Feasibility of automation and/or tool maturity level

Proposed Methods Development

- Postulate alternative approaches to domain analysis for DoD Ada applications
- Select one or more approaches for further development
- Provide consistent, cohesive and complete description of proposed method; address the following:
  - strategies and paradigms for domain analysis
  - process model for domain analysis
  - methods for each identified activity
  - products of domain analysis activities
  - resources required for domain analysis activities

Approach

Process Model for Domain Analysis

- **Within context of three distinct but integrated processes:**
  - development of reusable components
  - reuse of components
  - development of application software

Proposed Tools Identification

- **For each activity in the proposed methodology:**
  - identify existing or potential automated tools
  - describe how they support the activity
  - assess the importance of automating the activity
  - assess the feasibility of automation or the maturity level of existing tools
  - make recommendations for acquisition or development

Approach

Develop a Set of Guidelines

- Use results of research and analysis of existing methods and tools
- Develop guidelines for conducting domain analysis and applying results during software development; document will contain:
  - recommendations for methodology and tools to perform domain analysis
  - relationship of domain analysis methodology and tools to overall development methodology and tools
  - recommendation for addressing critical issues and risks in using domain analysis
  - recommendations for future R&D in domain analysis

N90-14793

CLASSIC-ADA^{TM}

Lois Valley
Software Productivity Solutions, Inc.

The SPS product, Classic-Ada^{TM}, is a software tool that supports object-oriented Ada programming with powerful inheritance and dynamic binding. Object Oriented Design (OOD) is an easy, natural development paradigm, but it is not supported by Ada. Following the DOD Ada mandate, SPS developed Classic-Ada to provide a tool which supports OOD and implements code in Ada. It consists of a design language, a code generator and a toolset. As a design language, Classic-Ada supports the object-oriented principles of information hiding, data abstraction, dynamic binding, and inheritance. It also supports natural reuse and incremental development through inheritance, code factoring, and Ada, Classic-Ada, dynamic binding and static binding in the same program. Only nine new constructs were added to Ada to provide object-oriented design capabilities. The Classic-Ada code generator translates user application code into fully compliant, ready-to-run, standard Ada. The Classic-Ada toolset is fully supported by SPS and consists of an object generator, a builder, a dictionary manager, and a reporter. Demonstrations of Classic-Ada and the Classic-Ada Browser were given at the workshop.

Why *Classic-Ada*™?

- Ada Mandate
- Object-Oriented Design is an easy natural development paradigm
- Ada doesn't support the object-oriented paradigm
- SPS needed a tool that allowed us to think in OOD and implement in Ada
- *Classic-Ada* is our answer to that need

What is *Classic–Ada*™?

Classic–Ada is:

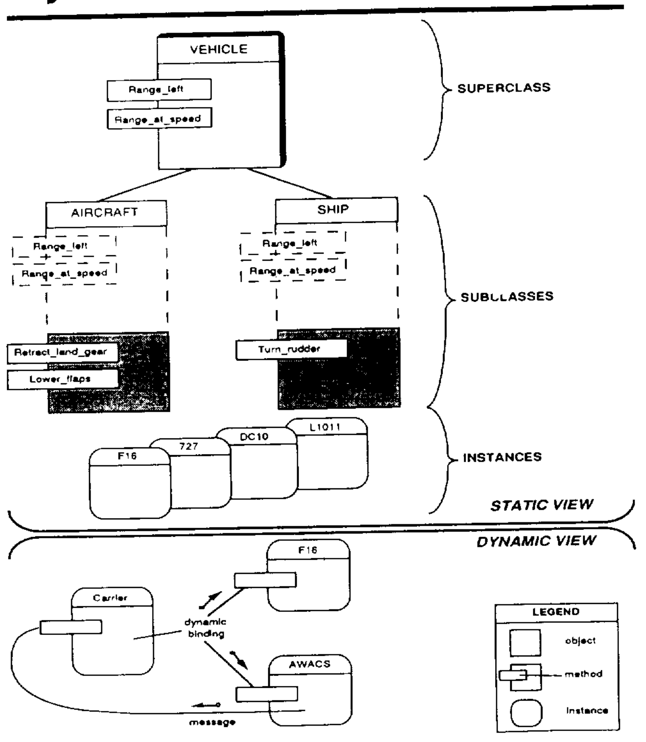
- A design language
- A code generator
- A toolset

Classic-Ada™ as a Design Language

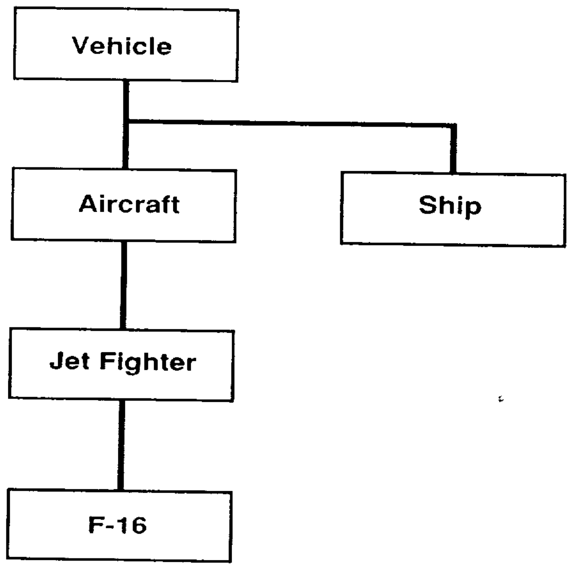
Supports object-oriented principles

- Information hiding - hiding the state of software components in variables visible only within the scope of that component
- Data abstraction - abstract data types defining an internal representation plus a set of operations used to access and manipulate it
- Dynamic binding - determining which operation is invoked for a specific abstract data type dynamically at runtime, depending on the object being manipulated
- Inheritance - enabling the easy creation of objects that are almost like other objects with just a few changes

Inheritance and Dynamic Binding



Inheritance Hierarchy



Reusability

- Inheritance enables the creation of objects that are *almost like* other objects with just a few changes
- Generalization promotes the constant migration to more general (and more reusable) objects
- Inheritance enhances *code factoring*, i.e. code to perform a particular task is found in only one place
- Dynamic binding increases flexibility by allowing the addition of new object classes without modifying or recompiling existing code
- Polymorphism, the ability for different classes to respond to the same message promotes interchangeable parts

Natural Reuse

"Object-oriented development integrates reuse into the development process so well that developers will find themselves developing reusable objects and reusing existing objects without even thinking about it."

Smaller, Cheaper Solutions

- Solve large problems by making the solutions smaller
  - Typically at least 1/4 the number of lines of code in an OOPL
  - Often as little as 1/10 or 1/20
- Productivity increases because effort per line of code is about the same as with procedural HOLs
- Manageability improves dramatically
  - Software system is easier to understand
  - There are less people to manage

    "Managers must reward designers for doing less - not more."

    -- Wilf LaLonde
- As the development converges, the lines of code will actually decrease as generalizations further optimize and compress the code
- You don't have to do *programming-in-the-large* to solve large programs if you make the large program small

Large OOP Experiences

<u>500,000 - 1,000,000 LOC</u> ► <u>25,000 - 100,000 LOC</u>
Procedural HOL → OOPL

- Operating systems
- Workstation / office automation environments
- CAD /CAE
- Telecommunications
- User interface / application frameworks
- Object-bases
- Simulations
- Manufacturing, operations and control systems
- Management information systems

Classic-Ada™ as a Design Language

Has added only nine constructs to Ada to provide this powerful capability. These constructs are:

- Class
- Superclass
- Instance
- Instantiate
- Method
- Destroy
- Send
- Self
- Super

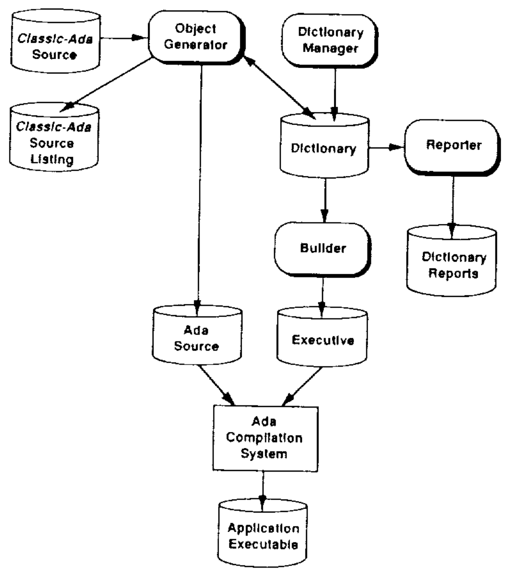
Classic-Ada™ as a Code Generator

- Generates user application code
- Generates application executive
- Generates fully compliant, ready to run, DoD standard Ada

Classic-Ada™ as a Toolset

- Is fully supported by SPS
- Consists of the following tools:
  - an object generator
  - a builder
  - a dictionary manager
  - a reporter

Classic-Ada™ Toolset



Classic-Ada™ as a Design Language

- Supports natural reuse through its inheritance capabilities
- Supports incremental development through inheritance
- Supports code factoring
- Supports Ada, *Classic-Ada*, Dynamic binding, and static binding in the same program
- Makes it easy to both generalize and specialize during development due to the way *Classic-Ada* is implemented
- Minimizes the need to compile large portions of code

N90-14794

PROTOTYPE SOFTWARE REUSE ENVIRONMENT AT GODDARD SPACE FLIGHT CENTER

Walt Truszkowski
NASA Goddard Space Flight Center

The Goddard Space Flight Center (GSFC) work is organized into four phases and includes participation by a contractor, CTA, Inc. The first phase was an automation study, which began with a comprehensive survey of software development automation technologies. Eight technical areas were analyzed for goals, current capabilities, and obstacles. The study documented current software development practice in GSFC Mission Operations and Data Systems Directorate, and presented short- and long-term recommendations that included focus on reuse and object-oriented development. The second phase, which has been completed, developed a prototype reuse environment with tools supporting object-oriented requirements analysis and design. This phase addressed the operational concept of software reuse, i.e., it attempted to understand how software can be reused. This environment has two semantic networks: object and key words, and includes automated search, interactive browsing and a graphical display of database contents. Phase 3 was a domain analysis of Payload Operations Control Center (POCC) software. The goal in this phase was to create an initial repository of reusable components and techniques. Seven existing Operations Control Centers at GSFC were studied, but the domain analysis proved to be very slow. A lesson learned from this was that senior people who understand the environment and the functionality of the area are needed to perform successful domain analyses. Four reuse paradigms were identified which are appropriate to different parts of a POCC. Phase 4 is the development of a prototype environment for rapid synthesis of POCC software. The four paradigms (or views) of software reuse will be prototyped and combined to provide support for POCC software development. These four paradigms are a dialog-based specification of high-level architecture, a very-high-level-language specification of the operational database, interface navigation/selection of reusable components, and graphical programming. Future work includes the design of a knowledge-based reuse environment.

Workshop on NASA Research in Software Reuse

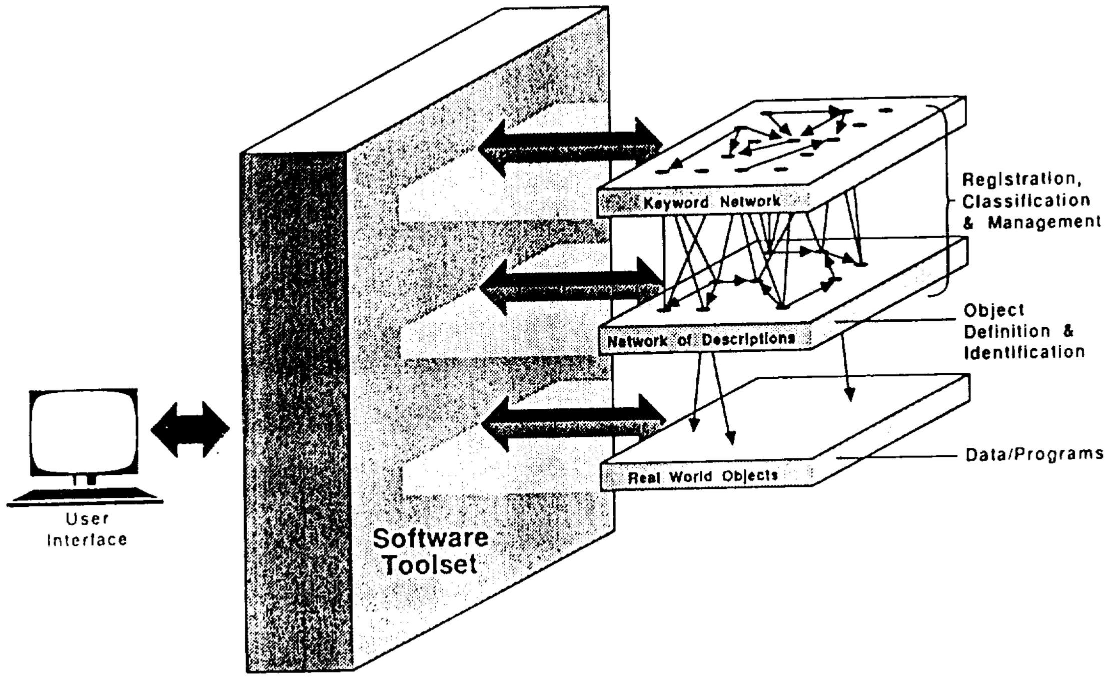
Phase 1 (FY '86): Automation Study

- Comprehensive survey of software development automation technologies
- Analyzed 8 technical areas: goals, current capabilities, obstacles
  - Semi-formal specification, formal specification, reuse, knowledge-based systems, prototyping, software metrics, performance analysis, work management
- Documented current S/W development practice in GSFC Mission Operations & Data Systems Directorate
  - Methods, tools, perceived strengths/weaknesses
- Short- and long-term recommendations
  - Focus on reuse and Object Oriented Development (OOD)
  - Revisit CHI and AI around 1990

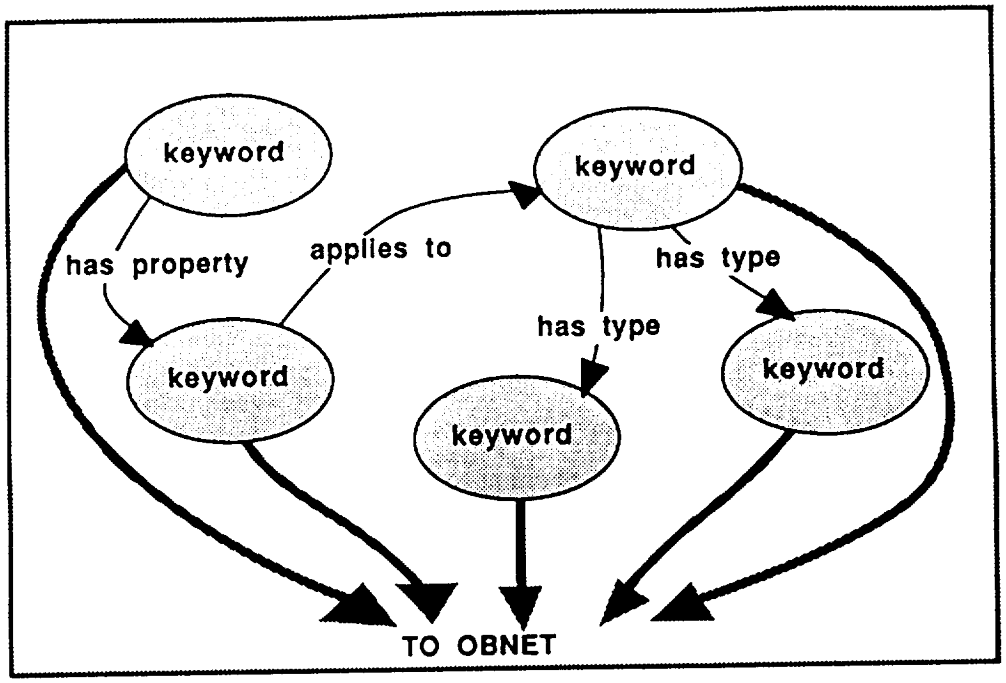
Phase 2 (FY '87): Prototype Reuse Environment

- Tools supporting object-oriented requirements analysis and design
  - Extended Goddard Object Oriented Design (GOOD) methodology to requirements analysis
  - Enhanced IDE Software Through Pictures environment
- Operational concept of software reuse
- Two semantic networks: objects and key words
  - Obnet: entity-relationship database of reusable components
  - Keynet: classification of reusable components
- Automated search and interactive browsing
- Graphical display of database contents

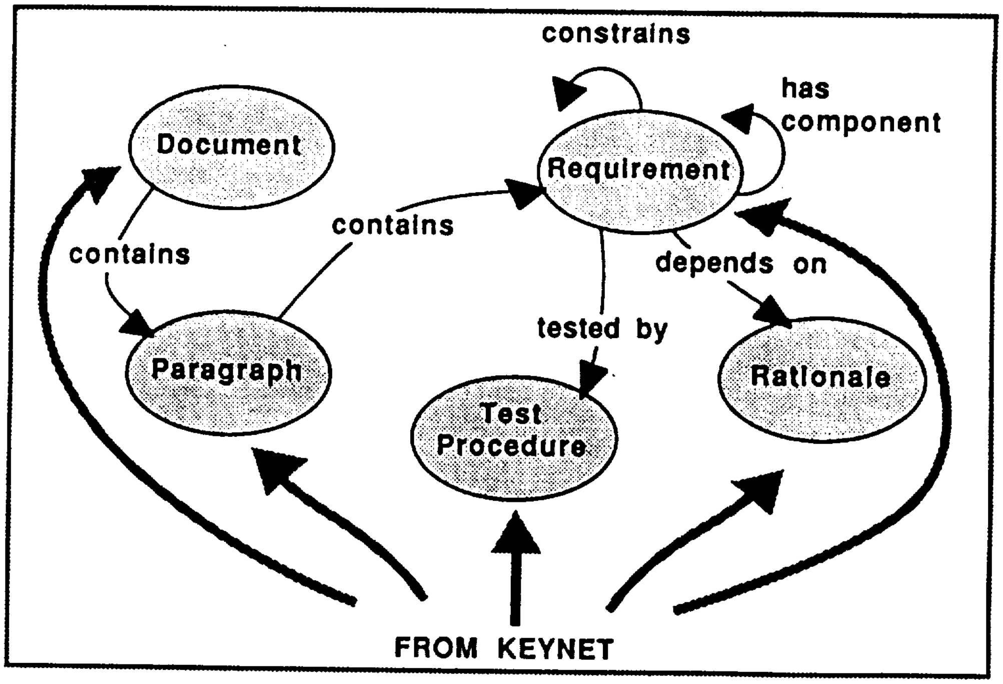
SEMANTX Architecture



The RMS KeyNet



RMS ObNet



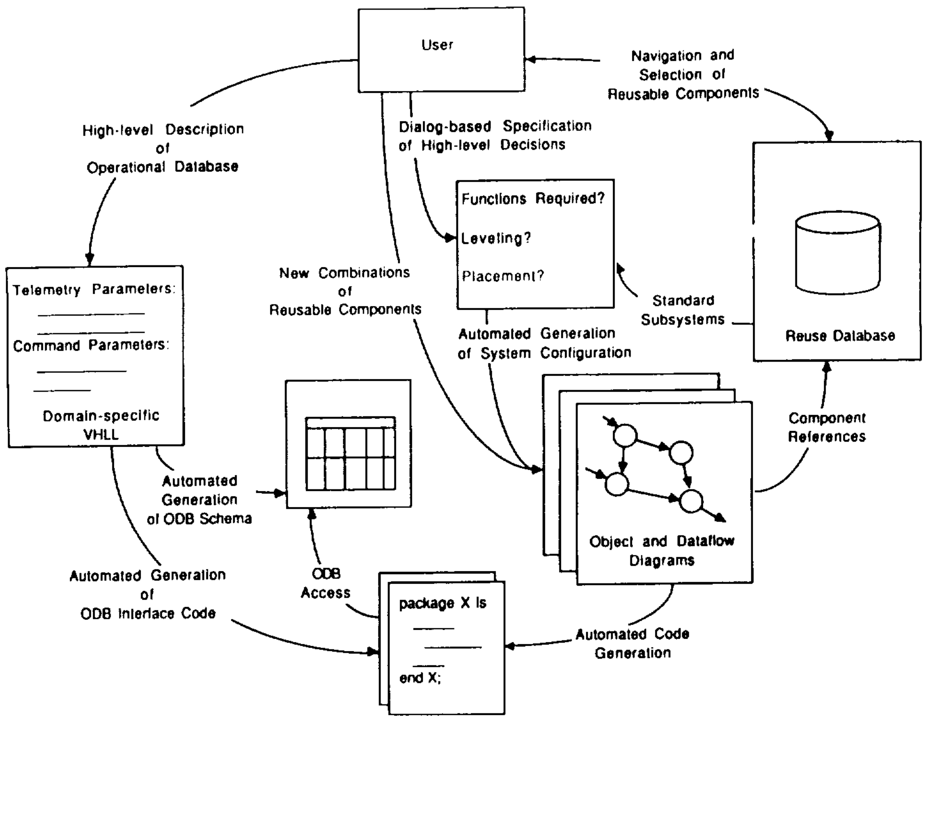
Phase 3 (FY `88): Domain Analysis of POCC Software

- Goal: Create an initial repository of reusable components and techniques
- Studied seven Multi-Satellite Operations Control Center (MSOCC) systems
  - Standard Software
  - Dynamic Explorer (DE)
  - International Sun/Earth Explorer (ISEE)
  - Earth-Radiation Budget Satellite (ERBS)
  - MSOCC Applications Executive (MAE)
  - Gamma Ray Observatory (GRO)
  - Cosmic Background Explorer (COBE)
- Determine typical POCC architecture and components
  - Classified variations
  - Identified obstacles to reuse
- Identified 4 reuse paradigms appropriate to different parts of a POCC system

Phase 4 (FY '89): Prototype Environment for Rapid Synthesis of POCC Software

- Dialog-based specification of high-level architecture
- Very-high-level language specification of Operational Database
  - Automated generation of database interface procedures
- Interface navigation/selection of reusable components
- Graphical programming
  - Specify new combinations of reusable components
  - Automated code generation from Object and Functional Diagrams

Four Automation Techniques Combine to Support POCC Software Development


User
Navigation and Selection of Reusable Components
High-level Description of Operational Database
Dialog-based Specification of High-level Decisions
Functions Required?
Leveling?
Placement?
Standard Subsystems
Reuse Database
Telemetry Parameters:
Command Parameters:
Domain-specific VHLL
New Combinations of Reusable Components
Automated Generation of System Configuration
Component References
Automated Generation of ODB Schema
Object and Dataflow Diagrams
Automated Generation of ODB Interface Code
ODB Access
package X is
end X;
Automated Code Generation

AI Revisited: Design of Knowledge-based Reuse Environment (FY `89)

- Survey recent efforts
  - Determine available technologies
  - Develop a knowledge-based reuse concept

- Focus on essential areas not yet explored
  - Capturing developer rationales
  - Learning from errors (e.g., misused components)

N90-14795

JPL REUSE PROGRAM

James W. Brown
Jet Propulsion Laboratory

The goal of the JPL reuse activity is to develop a quantitative understanding of the factors which encourage or inhibit software reuse, and of productivity improvements achievable through reuse. The primary activity is the measurement of parameters relevant to reuse in the environment of actual projects. The program has three objectives: (1) to develop a model to allow assessment of competing reuse techniques, (2) to extend reuse from the unit to the subsystem level, and (3) to expand from specific applications to a broader application domain. Application domains, which apply to all interplanetary projects, include Mission Operations, Science Information Systems, Flight Software, and Simulations. The program is targeting all phases and activities of the life cycle and a full range of software products. The approach will be both experimental (observe, hypothesize and evaluate) and constructive (introduce new tools and techniques). The primary target projects are Deep Space Network activities - the Ground Facilities facility upgrade, the Network Operations Control Center upgrade, and the Signal Processing Center. This is the first group of closely related projects being done in Ada at JPL. A "reuse base" will be developed initially by classifying potentially reusable components from one project; it will be used and expanded with additional projects.

GOAL

GOAL: Improve the software development process by application of advanced reuse technology

OBJECTIVES:

- Develop a model to allow assessment of competing techniques for reuse
- Improve leverage by extending reuse from the unit to the subsystem level
- Expand from specific applications to broader application domain

SCOPE - APPLICATION DOMAINS

- Mission Operations (planning, commanding, navigation, tracking and data acquisition for unmanned spacecraft)
- Science Information Systems (data management, level conversion, visualization, analysis and modeling)
- Flight Software (autonomous spacecraft operation, instrument software)
- Simulations (spacecraft operations, physical processes, command/control problems)

SCOPE - FULL LIFE CYCLE - ALL PHASES AND ACTIVITIES

Investigate impact of reuse on:

- Requirements Analysis
- Design
- Implementation
- Integration
- Test
- Maintenance
- Inter-project relationships

SCOPE - FULL RANGE OF SOFTWARE PRODUCTS

- Plans and Procedures (e.g. software management plans, configuration management plans, integration and test plans)
- Requirements and Constraints
- Designs
- Code (e.g. 3GL, 4GL, execution procedures, data tables)
- Test cases and test data
- Development tools and environments
- Run-time data (e.g. file labels, digital maps)

APPROACH

CHARACTERISTICS

- Experimental [observe (exploratory, descriptive), hypothesize, evaluate]
- Constructive (introduce new tools and techniques rather than survey natural selection)

STEPS

- Identify currently available reuse technologies
- Select a model that seems likely to improve software development
- Construct a "reuse base" based on the model
- Observe (measure) utilization patterns by actual development projects
- Revise and refine model based on observations
- Recommend tools and techniques for effective reuse

TARGET PROJECTS

Deep Space Network

- Ground Communications Facility Upgrade (GCF)
- Network Operations Control Center Upgrade (NOCC)
- Signal Processing Center (SPC)

Others (TBD)

BACKGROUND

Products available from previous work:

- Theoretical model
- Technology assessment
- Metrics evaluation
- Behavioral design for reuse base

SCHEDULE

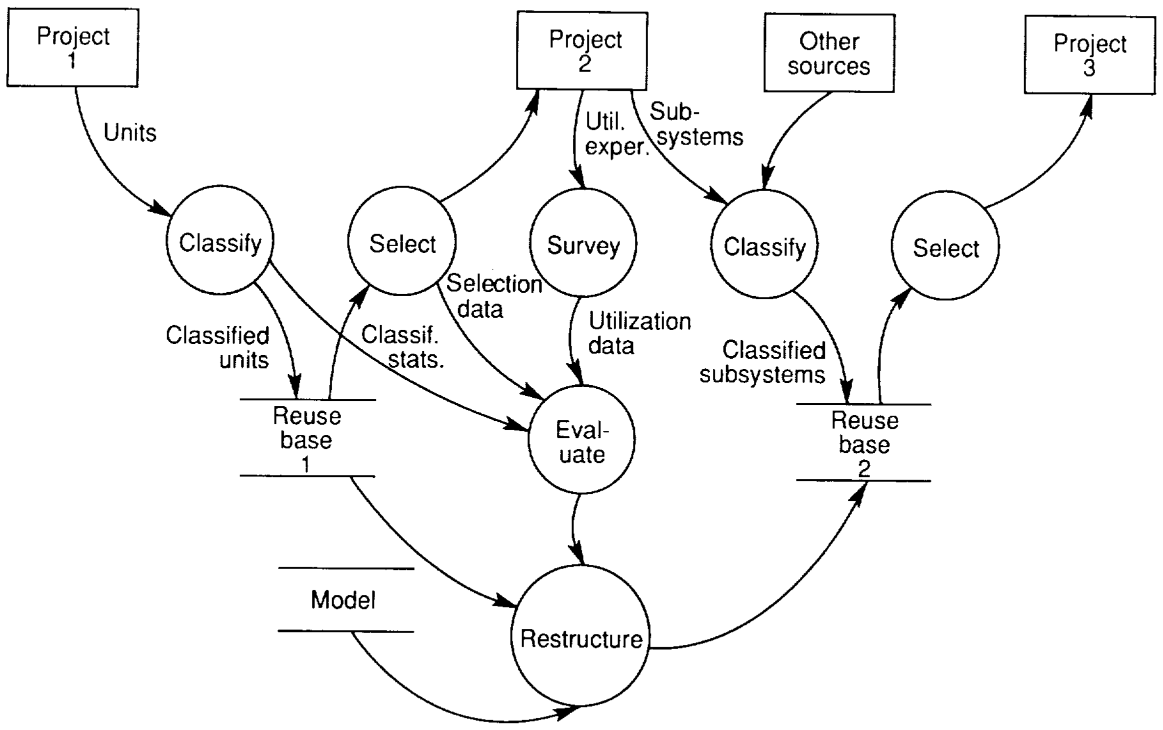
Phase 1 (FY89)

- Determine user needs (GCF); acquire and analyze components
- Design reuse base
- Implement reuse base
- Analyze reuse and report

Phase 2 (FY90-91)

- Determine user needs (NOCC, SPC)
- Identify new suppliers
- Modify reuse base design
- Implement modifications
- Monitor reuse; add users; add suppliers; adapt reuse base

JPL REUSE PROGRAM



N90-14796

JOHNSON SPACE CENTER SOFTWARE REUSE ACTIVITY

Steve Gorman
NASA Johnson Space Center

There is a strong operational interest in reuse and commonality at the Johnson Space Center (JSC). Although commonality and reuse were not emphasized in the Space Shuttle Orbiter Project, it is a major goal for Space Station Freedom and the Software Support Environment (SSE). Research activities at JSC are generally conducted through the Software Engineering Research Center (SERC) of the University of Houston at Clear Lake. The Life Cycle Model developed by SERC includes reuse at each phase, but reuse is not a principal theme. The SSE is a significant entry point for new reuse technology, and the SERC can provide consultation and possible prototypes. SERC is seen as an interface to other NISE reuse researchers. The AdaNET is managed at JSC through the University of Houston at Clear Lake for the NASA Office of Technology Utilization. It may also be a "gateway" for reuse research.

JSC MANAGED PROJECTS

- ORBITER PROJECT - COMMONALITY & REUSE NOT EMPHASIZED
- SPACE STATION - WORK PACKAGE 2 & SOFTWARE SUPPORT ENVIRONMENT (SSE)
  - COMMONALITY & REUSE A MAJOR GOAL
  - ADA FOR OPERATIONAL SOFTWARE
  - MODELS & SIMULATIONS
  - SAME SSE ACROSS THE PROGRAM
- AdaNET
  - MANAGED AT JSC (THROUGH UHCL) FOR OFFICE OF TECHNOLOGY TRANSFER
- SOFTWARE ENGINEERING RESEARCH CENTER (SERC) ACTIVITY
  - CODE R SUPPORTED
  - NOT A PRINCIPAL THEME AT SERC
  - ADDRESSED IN LIFE CYCLE MODEL
  - CONSULTING & WHITE PAPERS AS REQUIRED

SOFTWARE REUSE IN THE CLEAR LAKE MODEL

- "CONCEPTUAL AND IMPLEMENTATION MODELS WHICH SUPPORT LIFE CYCLE REUSABILITY OF PROCESSES AND PRODUCTS IN COMPUTER SYSTEMS AND SOFTWARE ENGINEERING"
- "SOLUTION IN THE LARGE"
- SOFTWARE REUSE IS PRESENTED AS SUBSET OF LIFE CYCLE REUSE
- SEVEN LIFE CYCLE PHASES SHOWN - REUSE AT EACH PHASE ADDRESSED
- REUSE CANDIDATES ARE MUCH MORE THAN CODED MODULES
  - REQUIREMENTS
  - SCHEDULES
  - BUDGETS
  - DESIGN
  - TOOLS
  - METHODS
- CONTRASTS WITH CODE COLLECTIONS AS THE BOOCH OR BERARD COMPONENTS - SINGLE PHASE ONLY

REUSE OVER THE SOFTWARE LIFE CYCLE

PHASE 1	PHASE 2	PHASE 3	PHASE 4	PHASE 5	PHASE 6	PHASE 7
SYSTEM RQTS. ANALYSIS	DETAILED RQTS. ANALYSIS	PRELIM. DESIGN	DETAILED DESIGN	CODING & UNIT TEST	S/W COMP. INTEG.	OPS. & SUSTAIN. ENGR.

- PHASE 1 - METHODS, SCHEDULES, BUDGETS, DEV. PLANS
- PHASE 2 - METHODS, REQUIREMENTS, INTERFACES
- PHASE 3 - METHODS, DESIGN, TOOLS
- PHASE 4 - METHODS, DESIGN, TOOLS
- PHASE 5 - METHODS, CODE, PACKAGES, STRUCTURE
- PHASE 6 - METHODS, ENVIRONMENT DESIGN, INTERFACES
- PHASE 8 - METHODS, CONFIGURATIONS, TOOLS

- META DATA FOR THESE DIFFERENT PRODUCTS AND PROCESSES WILL BE A CHALLENGE

JSC SOFTWARE REUSE SUMMARY

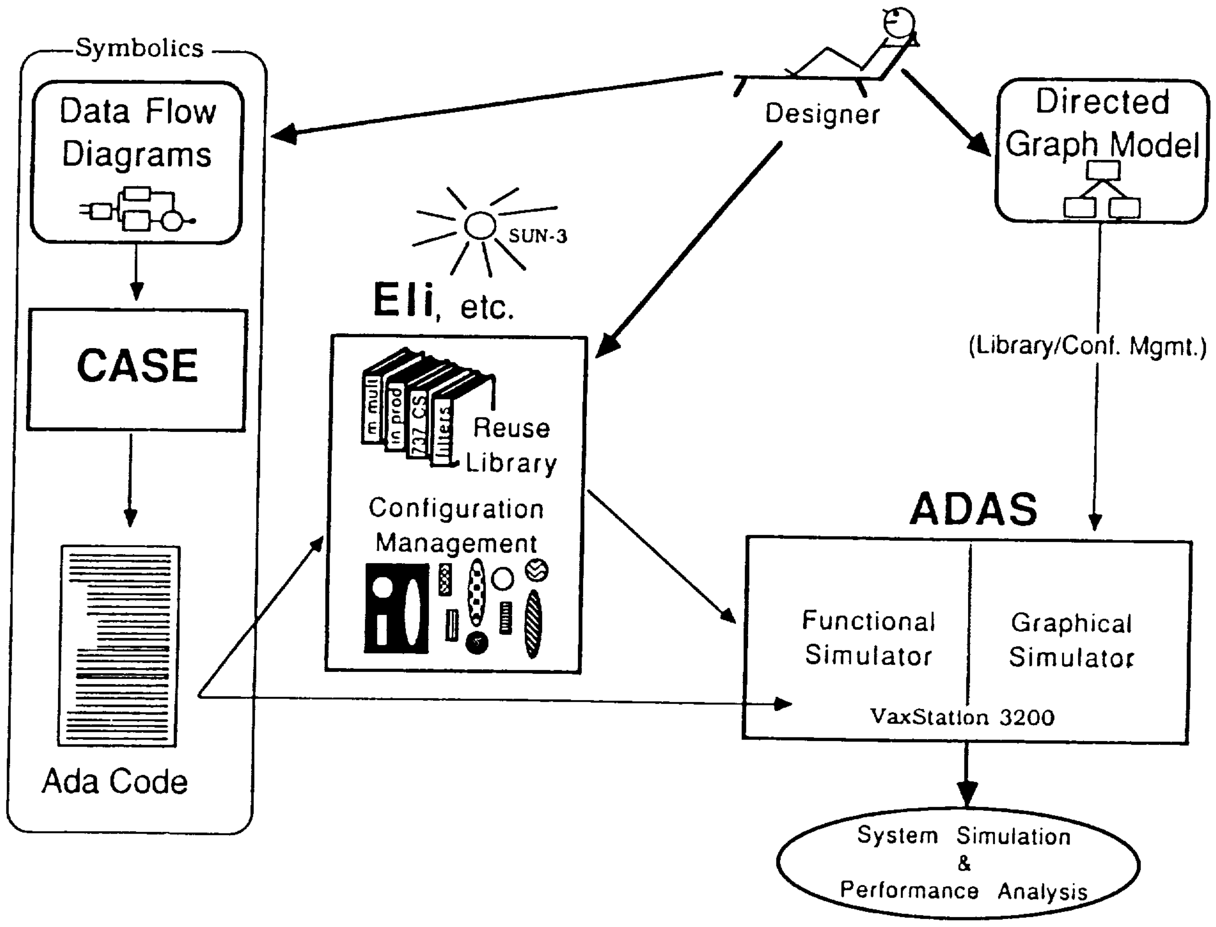
- STRONG OPERATIONAL INTEREST IN REUSE & COMMONALITY
- "RUBBER HITS THE ROAD" FOR MANY S/W PROJECTS AT JSC
- SSE IS A SIGNIFICANT ENTRY POINT FOR NEW REUSE TECHNOLOGY
  - SIGNIFICANT REUSE IS MAJOR SSFP& SSE GOAL
  - STRONG INTEREST IN NEW & BETTER APPROACHES
  - SERC AS CONSULTANT & POSSIBLE PROTOTYPER
- SERC AS INTERFACE TO OTHER NISE REUSE RESEARCHERS
- POSSIBLE "GATEWAY" RESEARCH THROUGH AdaNET - BASED ON CLEAR LAKE MODEL WITH SERC AS CONSULTANT
- CRITICAL FOR JSC TO "STAY ON TOP OF" REUSE TECHNOLOGY
  - SERC AS CONSULTANT
  - SSE PROJECT AS OPERATIONAL INTERFACE
  - AdaNET AS INTERFACE TO LARGER Ada & SOFTWARE ENGINEERING COMMUNITY

REUSE RESEARCH PLANS AT LANGLEY RESEARCH CENTER

Susan Voigt and Carrie Walker
NASA Langley Research Center

The reuse activities at Langley have centered on the development of the Eli system by SPS, as already described. The development of a computer systems design environment at Langley was described as a target application for the future Eli system. This environment combines software development tools with an architecture design and analysis tool. Specifically, a Computer-Aided Software Engineering (CASE) system, under development at Charles Stark Draper Laboratory for Langley, is being used to generate Ada code for use in architecture functional simulations using the Architecture Design and Assessment System (ADAS). The Eli system will be included in this tool set and will be used to organize and promote reuse of the functional simulation code modules.

SYSTEM DESIGN ENVIRONMENT



N90-14798

REUSE AT THE SOFTWARE PRODUCTIVITY CONSORTIUM

David M. Weiss
Software Productivity Consortium

The Software Productivity Consortium is sponsored by 14 aerospace companies as a developer of software engineering methods and tools. Software reuse and prototyping are currently the major emphasis areas. The Methodology and Measurement Project in the Software Technology Exploration Division has developed some concepts for reuse which they intend to develop into a synthesis process. They have identified two approaches to software reuse: opportunistic and systematic. The assumptions underlying the systematic approach, phrased as hypotheses, are the following: the redevelopment hypothesis, i.e., software developers solve the same problems repeatedly; the oracle hypothesis, i.e., developers are able to predict variations from one redevelopment to others; and the organizational hypothesis, i.e., software must be organized according to behavior and structure to take advantage of the predictions that the developers make. The conceptual basis for reuse includes: program families, information hiding, abstract interfaces, uses and information hiding hierarchies, and process structure. The primary reusable software characteristics are black-box descriptions, structural descriptions, and composition and decomposition based on program families. A good methodology has the following properties:

1. It answers the following key questions at any point in the development process:
   a. What should I do next?
   b. What output do I produce?
   c. What input and resources do I need to produce it?
   d. How do I know when I'm done?
2. It is based on a clear set of principles
3. It leads to quantifiable improvements in productivity and quality
4. It is useful to engineers
5. It promotes reuse
6. It supports a sound business approach

Automated support can be provided for systematic reuse, and the Consortium is developing a prototype reuse library and guidebook. The software synthesis process that the Consortium is aiming toward includes modeling, refinement, prototyping, reuse, assessment, and new construction. A number of key issues were also discussed.

TOPICS

- Concepts
  - Systematic vs Opportunistic Reuse
  - Assumptions Underlying Systematic Reuse
  - Underlying Principles
- Methodological Considerations
- Automated Support
  - Reusable Software Libraries
- Current Consortium Practice
- Direction
  - Synthesis

ORGANIZING SOFTWARE FOR REUSE

- Opportunistic Reuse – The Garage Sale Approach
  - Many individual parts
  - Search for part with desired behavior
    - -- Attributes + Behavioral Description
- Systematic Reuse – Systems Approach
  - Collections of related parts
  - Search for system that meets requirements
    - -- Attributes + Behavioral Description + Relationships + Classification

ASSUMPTIONS

- Redevelopment Hypothesis
  - Software developers solve same problems repeatedly
  - Solutions are captured as systems
  - Variations
    - Devices
    - Algorithms
    - Platforms
    - Functionality
- Oracle Hypothesis
  - Developer must be able to predict changes
- Organizational Hypothesis
  - Organize according to behavior and structure
  - Expose structures that make changeable decisions apparent
  - Identify common characteristics

DESIRABLE SOFTWARE CHARACTERISTICS

- Black-box description
  - Behavioral approach
- Structural Descriptions
- Composition and decomposition based on collections of parts

CONCEPTUAL BASIS

- Program Families
  - Characterize commonalities first
- Information Hiding
  - Encapsulate changeable decisions
- Abstract Interfaces
  - Behavioral descriptions of modules
- Uses Hierarchy
  - Protect subsettability
  - Explicit decisions about dependencies
- Information Hiding Hierarchy
  - Roadmap for change
- Process Structure
  - Performance assessment
  - Reconfigurability

REUSABLE SOFTWARE CHARACTERISTICS

- Black-Box Descriptions
  - Behavioral approach, e.g., based on A-7 module descriptions
- Structural Descriptions
  - Hierarchical views, based on information hiding and uses hierarchies
- Composition and Decomposition Based On Program Families
  - Families described structurally
  - Components of families described behaviorally
  - Many shared subfamilies

METHODOLOGICAL CONSIDERATIONS

DEFINITIONS

- PROCESS
  - Set of activities used to produce and maintain software
- METHODOLOGY: Answers to the Questions:
  - What do I do next?
  - What input do I need to do it?
  - What output do I produce?
  - What resources do I need to produce it?

ATTRIBUTES OF A GOOD METHODOLOGY

- Answers the key questions
  - What do I do next?
  - What input do I need to do it?
  - What output do I produce?
  - What resources do I need to produce it?
- Based on a clear set of principles
  - Information hiding, hierarchical structuring, etc.
- Leads to improvements in productivity and quality
  - Measurable
- Useful to engineers
- Promotes reuse
- Supports sound business approach
- Can be adopted incrementally

OBJECTS SUPPORTING SYSTEMATIC REUSE

- Requirements Specification
- Module Guide
- Abstract Interface Specification
- Allowed-to-Use Hierarchy
- Module Internal Documentation } Design
- Uses Hierarchy
- Process Structure
- Potential Family Members
- Code
- Tests

SOFTWARE EVOLUTION PROCESS

- Developers maintain collections of program families
- Requirements are identified using families that support:
  - simulation, prototyping, modeling, other forms of analysis,
  - production of specifications.
- Given the requirements for a new system:
  - the collections are searched for a family with a member that meets the requirements,
  - modules of the family are adapted and assembled to produce the new member,
  - if no such family exists, a new family is created (rare).

AUTOMATED SUPPORT FOR SYSTEMATIC REUSE

- Reuse Library
  - Repository for collections of families
- Adaptation Analysis
  - Tracing the effects of change
- Construction of specifications
  - Editor, browser
  - Design representation
- Adaptation Mechanism
  - Parameterized module/subset generation
    -- Generic, macros
- Modeling
  - Performance analysis
- Composition
  - System generation

REUSE LIBRARIES

- Storage of Life Cycle Objects (LCOs) and their Descriptions
  - Requirements
  - Module Guide
  - Module Interface Specification
  - Code

    ⋮
- Search Mechanisms
  - By Attribute
    -- Language, version, author, producing tool, etc.
  - By LCO type
  - By Relation
    -- Hierarchy traversal (uses, information hiding, composition, etc.)
  - By Classification
- Object descriptions
- Population

CURRENT CONSORTIUM PRACTICE

- Guidebook
  - Management and Technical Volumes
- Reuse Library Prototype

Software Technology Exploration Guidebook
Volume 1: Management Guidebook

Table of Contents

August 26, 1988

Software Technology Exploration Guidebook
Volume 2: Technical Guidebook

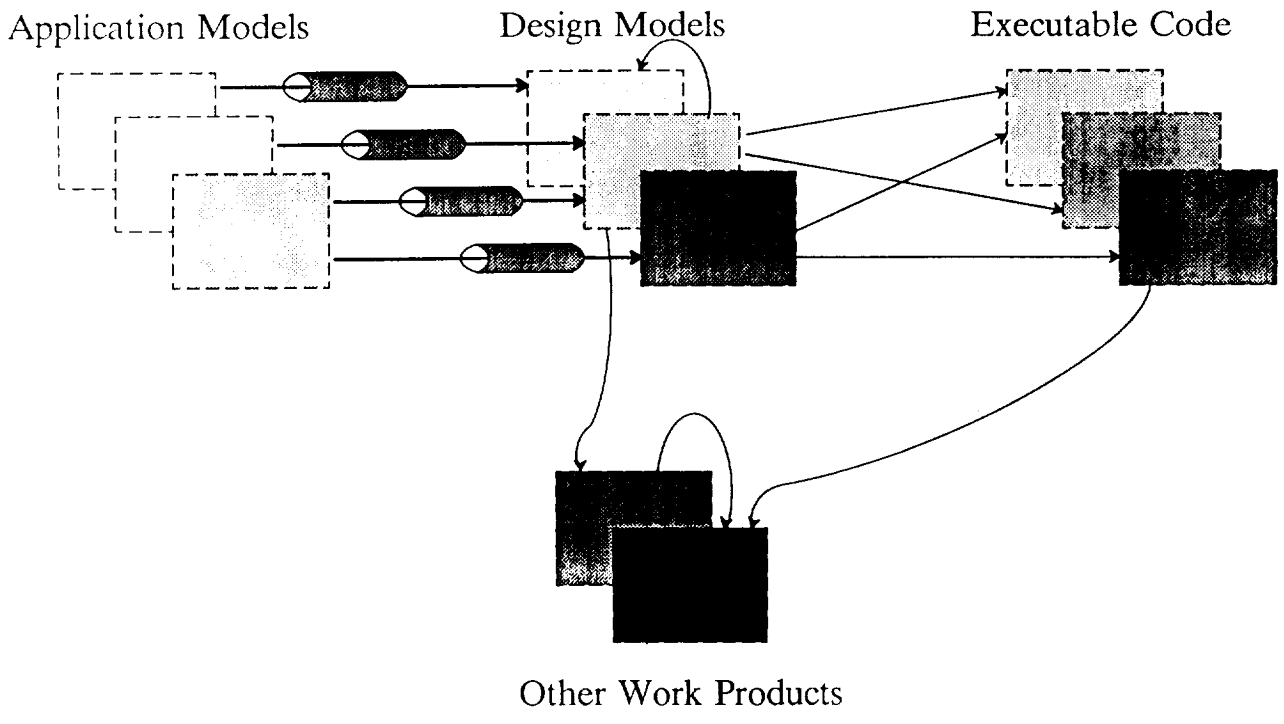
Table of Contents

August 31, 1988

DIRECTION

SYNTHESIS: A PROCESS THAT RELIES ON THE PRODUCTION OF SOFTWARE FROM MODELS SPECIFICALLY DESIGNED FOR REUSE

RELATIONSHIPS AMONG MODELS



ASPECTS OF SYNTHESIS

- MODELLING
  - APPLICATION
  - DESIGN
  - IMPLEMENTATION
- REFINEMENT
  - SUCCESSIVE APPROXIMATION OF PROBLEM
  - SUCCESSIVE APPROXIMATION OF SOLUTION
- PROTOTYPING
  - REFINEMENT THROUGH ISSUE RESOLUTION
- REUSE
  - STANDARDIZED ENGINEERING SOLUTIONS
  - SOLUTIONS REPRESENTED IN TERMS OF REUSABLE PARTS
  - COMPOSITION OF NEW SYSTEMS FROM EXISTING, ADAPTED, AND NEW PARTS
- ASSESSMENT
  - QUANTIFICATION OF APPROXIMATION
- NEW CONSTRUCTION

STEPS IN THE SYNTHESIS PROCESS

- SPECIFY REQUIREMENTS
  - DIRECTLY, IN TERMS OF PRE-DEFINED DOMAIN VOCABULARY
  - ANALOGOUSLY, IN TERMS OF DIFFERENCES BETWEEN NEW NEEDS AND EXISTING SYSTEMS
- MAKE APPLICATION MODEL
  - MODEL CONSTRUCTION
  - MODEL ASSESSMENT
- MAKE DESIGN MODEL
  - SELECTION OF CANONICAL DESIGN
  - ADAPTATION OF CANONICAL DESIGN
  - INVENTION OF NEW DESIGN
  - ASSESSMENT OF DESIGN
- IMPLEMENT
  - COMPLETION OF NEW AND ADAPTED PARTS
  - COMPOSITION OF PARTS INTO PROTOTYPES/PRODUCTS
  - VERIFICATION

SUPPORTING ELEMENTS

- REPOSITORIES
  - REUSE LIBRARY
  - PROJECT LIBRARY
- REPRESENTATION TECHNOLOGY
  - USER INTERFACE
  - SPECIFICATIONS FOR MODELS, DESIGNS, CODE PRODUCTION
- METHODOLOGY
  - PROCESS MODEL
  - NATURE OF PARTS AND RELATIONS
  - MANAGEMENT OF PROCESS
- ARCHITECTURE OF TOOLSETS
  - TOOLSET INTEGRATION

KEY ISSUES

- HOW TO DO DOMAIN ANALYSIS
  - SYSTEMATIC APPROACH
  - APPLICATION MODELLING
  - RE-ENGINEERING
- NOTATIONS AND MECHANISMS FOR MAPPING FROM APPLICATION MODEL TO SOFTWARE DESIGN
  - WHAT NOTATIONS?
  - HOW MANY INTERMEDIATE LEVELS?
- HOW TO REPRESENT DESIGNS: PARTS AND THEIR RELATIONS
  - WHAT NOTATIONS?
  - STORAGE, RETRIEVAL, AND SEARCH
  - RE-ENGINEERING
- HOW TO ADAPT, COMPOSE, AND VERIFY PARTS
  - DESIGN PARTS
  - CODE PARTS

KEY ISSUES (CONC)

- HOW TO ASSESS DESIGNS AND CODE
  - PERFORMANCE
  - FUNCTION
  - DEPENDABILITY
- HOW TO PRODUCE CODE
  - PROTOTYPE
  - PRODUCTION
- HOW SHOULD THE ENGINEERS INTERACT IN THE PROCESS?
  - INTERFACE
  - PROCESSING STEPS
  - UNDERSTANDING OF CONTEXT
- HOW SHOULD THE PROCESS BE MANAGED?
- HOW SHOULD THE EFFECTIVENESS BE MEASURED?
  - CURRENT STATE
  - GUIDE TO IMPROVEMENT

PROJECTS

- METHODOLOGY, MEASUREMENT, AND MANAGEMENT
- DESIGN REPRESENTATION, MAPPING FROM APPLICATION MODELS, AND COMPOSITION
- DOMAIN ANALYSIS
- REPOSITORIES
- ASSESSMENT
- VERIFICATION
- ADAPTATION
- INTERACTION WITH THE ENGINEERS

PROGRAM FAMILIES
HARDWARE ANALOGIES -- THE IBM 360, DEC PDP-11

- Families of computers
  - Same instruction set architecture (Behavioral description)
  - Different implementations
  - Same operating system (Different versions)
- Program family: A set of programs is a program family if the programs have so much in common that it pays to study their common characteristics before investigating the special properties of individual programs.

EXAMPLE FAMILY CLASSIFICATION

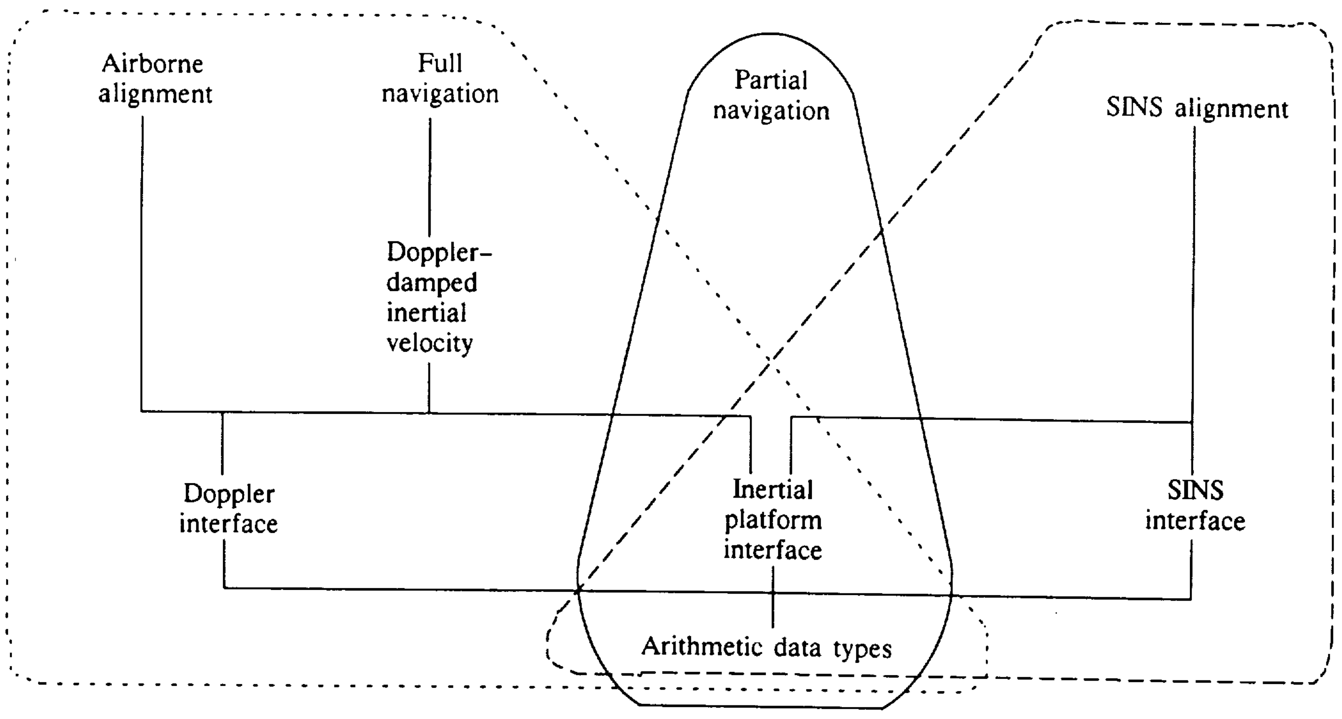
Tool Families

Process Support	Technology
Describing	Data Storage & Retrieval
Composing & Decomposing	User Interface
Assessing	Data Transformation
Retaining	Configuration

Application Families

Avionics	Communications
A-6	Control Systems
A-7	Speech Processing
727	
737	

A-7 NAVIGATION FAMILY



INFORMATION HIDING

- A decision that is likely to change should be encapsulated in a module
  - Changeable decision is the secret
    - -- data structures
    - -- algorithms
    - -- device characteristics
  - Basis for Ada packages
- Module: An information hiding module is a work assignment
  An information hiding module is a black-box
  An information hiding module is a set of programs and shared data
  An information hiding module is a finite state machine
- Interface: An interface between two modules is the set of assumptions that the programmer of one module may make about the other module
- Abstract Interface: An interface that represents many possible actual interfaces

HIERARCHIES AND STRUCTURE

- Module Hierarchy

Parts:	Modules	
Relation:	Subsecret	B is a submodule of A if B's secret is a subsecret of A's secret

- Uses Hierarchy

Parts:	Programs	
Relation:	Uses	A uses B if A requires the presence of B

- Process Structure

Parts:	Processes	
Relation:	Awaits	A awaits B if A cannot progress until B progresses

A-7 MODULE STRUCTURE

- HARDWARE-HIDING MODULE DECOMPOSITION
  - EXTENDED COMPUTER MODULE
  - DEVICE INTERFACE MODULE
- BEHAVIOR-HIDING MODULE DECOMPOSITION
  - FUNCTION DRIVER MODULE
  - SHARED SERVICES MODULE
- SOFTWARE DESIGN MODULE DECOMPOSITION
  - APPLICATION DATA TYPE MODULE
  - PHYSICAL MODEL MODULE
  - DATA BANKER MODULE
  - SYSTEM GENERATION MODULE
  - SOFTWARE UTILITY MODULE

- EXTENDED COMPUTER MODULE
  - DATA TYPE MODULE
  - DATA STRUCTURE MODULE
  - INPUT/OUTPUT MODULE
  - COMPUTER STATE MODULE
  - PARALLELISM CONTROL MODULE
  - SEQUENCE CONTROL MODULE
  - DIAGNOSTICS MODULE
  - VIRTUAL MEMORY MODULE (HIDDEN)
  - INTERRUPT HANDLER MODULE (HIDDEN)

- DEVICE INTERFACE MODULES
  - AIR DATA COMPUTER
  - ANGLE OF ATTACK SENSOR
  - AUDIBLE SIGNAL DEVICE
  - COMPUTER FAIL DEVICE
  - DOPPLER RADAR SET
  - FLIGHT INFORMATION DISPLAYS
  - FORWARD LOOKING RADAR
  - HEAD-UP DISPLAY
  - INERTIAL MEASUREMENT SET
  - PANEL
  - PROJECTED MAP DISPLAY SET
  - RADAR ALITIMETER
  - SINS
  - SLEW CONTROL
  - SWITCH BANK
  - TACAN
  - VISUAL INDICATORS
  - WAYPOINT INFORMATION SYSTEM
  - WEAPON CHARACTERISTICS
  - WEAPON RELEASE SYSTEM
  - WEIGHT ON GEAR

- FUNCTION DRIVER MODULE
  - AIR DATA COMPUTER FUNCTIONS
  - AUDIBLE SIGNAL FUNCTIONS
  - COMPUTER RADAR FUNCTIONS
  - DOPPLER RADAR FUNCTIONS
  - FLIGHT INFORMATION DISPLAY FUNCTIONS
  - FORWARD LOOKING RADAR FUNCTIONS
  - HEAD-UP DISPLAY FUNCTIONS
  - INERTIAL MEASUREMENT SET FUNCTIONS
  - PANEL FUNCTIONS
  - PROJECTED MAP DISPLAY SET FUNCTIONS
  - SINS FUNCTIONS
  - VISUAL INDICATOR FUNCTIONS
  - WEAPON RELEASE FUNCTIONS
  - GROUND TEST FUNCTIONS

- SHARED SERVICES MODULE
  - MODE DETERMINATION MODULE
  - STAGE DIRECTOR MODULE
  - SHARED SUBROUTINE MODULE
  - SYSTEM VALUE MODULE
  - PANEL I/O SUPPORT MODULE
  - DIAGNOSTIC I/O SUPPORT MODULE
  - EVENT TAILORING MODULE

- APPLICATION DATA TYPE MODULE
  - SYSTEM APPLICATION DATA TYPES
  - LOCAL APPLICATION DATA TYPES
  - SHARED APPLICATION DATA TYPES

- PHYSICAL MODEL MODULE
  - EARTH MODEL MODULE
  - AIRCRAFT MOTION MODULE

CHARACTERIZING FAMILIES

- By structure
  - module hierarchy, uses hierarchy, process structure
- By function
  - navigation, device control
- By application
  - avionics, satellite communications
- By technology
  - relational database, MonteCarlo simulation
- By ???

- Families composed of modules
- Modules characterized by externally-visible behavior
  - Black-boxes
  - Abstract interfaces specify behavior

SUMMARY

- Systematic Reuse Based on Families
- Concepts
  - Program Families (1976)
  - Information hiding (1970)
  - Abstraction
  - Behavioral specification (1972)
  - Hierarchies
- Examples
  - A-7 (1980)
- Relations to other technology
  - Ada
  - Object oriented programming
  - Object oriented design

N90-14799

SSFP APPROACH TO SOFTWARE REUSE

Peg Snyder
Space Station Freedom Program

This talk began by presenting the Space Station Freedom Program (SSFP) definitions of software commonality and software reuse. Software commonality is the use of identical, interchangeable, functionally compatible, or similar software items to satisfy different sets of functionally similar requirements. The Software Support Environment (SSE) and the Data Management System (DMS) of onboard computing facilities are examples of SSFP common software. Software reuse is the use of identical, compatible, or similar software items in either modified or unmodified form to satisfy development activities at any point in the software life cycle; in other words, taking an existing item and applying it to another development activity. Software commonality has been mandated in several critical areas (such as the SSE and DMS) and a policy directive is under review. A software reuse study group was established in May 1988 to gather background information (see Level II Software Reuse Study that follows by Scott Herman). The SSFP Program Definition and Requirements Document contains requirements for SSE support in the area of software reuse. The SSE is a collection of tools and rules, and provides the common environment to be used for the life cycle management of all SSFP operational software. Operational software includes ALL flight and ground software that either (1) interfaces with on-orbit elements in real time, (2) is critical to the mission, or (3) is SSE software. The SSE supports software development in Ada and provides tools for process management, software production, integration, test and verification, as well as training and library management. The SSE will provide the mechanisms required to implement SSFP evolving strategies for software reuse.

C-2

SPACE STATION FREEDOM PROGRAM (SSFP) APPROACH TO SOFTWARE REUSE

- **Definitions**
- **Status**
- **Introduction to the Software Support Environment (SSE)**
- **Summary**

SSFP APPROACH TO SOFTWARE REUSE

DEFINITIONS

Software Commonality: Use of identical, interchangeable, functionally compatible, or similar software items to satisfy different sets of functionally similar requirements.
- Common items are from any phase of life cycle
- Items are used without modifications

Software Reuse: Use of identical, compatible, or similar software items in either modified or unmodified form to satisfy development activities at any point in the software life cycle.
- Reusable items already exist
- Reusability is determined by how well the existing item satisfies requirements or derived attributes of current development activity

Status

- Software Commonality
  - Has been mandated in several critical areas such as:
    - -- Software Development (SSE)
    - -- Onboard data management (DMS)
  - Level II Policy directive under review
- Software Reuse Strategy
  - Level II study group was established May 1988 to gather background information
  - Level II sponsored "Reusable Software Flight Certification Requirements" task
  - Level II Policy directive under review
  - The Software Support Environment (SSE) is mandated to support the SSFP reuse strategy
  - Participation in this workshop is part of the process

SSFP APPROACH TO SOFTWARE REUSE

The SSFP Level II requirements for SSE support in the area of software reuse are baselined in SSP 30000 Sec. 11, Rev. A dated October 15, 1988 as follows:

5.3.7 REUSABLE SOFTWARE

The SSE shall provide the capability for identifying and controlling the use of software components which may be used in multiple applications via a controlled library of reusable components. The SSE shall identify, maintain, and support the dissemination of reusability standards and reusable software components for SSP operational software.

INTRODUCTION TO THE SOFTWARE SUPPORT ENVIRONMENT (SSE)

- What is the SSE?
- Why does Space Station need the SSE?
- SSE Implementation Approach
- Who are the SSE Users?
- SSE Functionality
- Conclusion

WHAT IS THE SSE?

- SSE is a collection of:
  - Tools (software)
  - Rules (procedures, standards, s/w production hardware specs, documentation, policy, training materials)
- SSE provides the common environment to be used for the life cycle management of all SSP operational software
- The SSE supports all SSP facilities involved in software life cycle management. These facilities include:
  - Work package and KSC Software Production Facilities
  - The Multi-System Integration Facility (MSIF)
  - The SSE Development Facility
- The SSE supports and provides mechanisms to enforce program-wide policies and standards such as:
  - Standard programming language (Ada)
  - Common User Interface standards
  - Software documentation standards
  - Common software verification approach

DEFINITION OF OPERATIONAL SOFTWARE

Operational Software is:

ALL flight and ground software that either

(1) interfaces with on-orbit elements in real time

or

(2) is critical to the mission,
- **such as all control center, test, and certification software**
- **including associated models and simulations**

or

(3) SSE software

WHY DOES SPACE STATION NEED THE SSE?

- **Software is high risk for the SSP in terms of both safety and cost**
  - **Large amount of software to be developed**
  - **Integration and testing are major issues - multiple developers are organizationally and geographically distributed**
  - **Sustaining engineering is a major cost factor in the SSP software life cycle**
- **SSE provides the means to control SSP software life cycle costs**
  - **The SSE is a single implementation of tools and rules rather than many**
  - **SSE enables consolidation of contractors and skills for sustaining engineering of SSP software**
- **SSE provides the means to control SSP software quality**
  - **Common program-wide standards and tools will be utilized for software integration and testing**

SSE IMPLEMENTATION APPROACH

- **Single contractor for SSE development - Contract awarded to Lockheed Missiles & Space Co., Inc. (LMSC)**
  - Contract Start (CSD) - 7/10/87
  - Contract Duration is 6 yrs with additional 3 yr option
  - Contract Type - Completion form for 1st year
    Level of effort beginning 7/10/88
  - Contractor Location - Houston, Texas

- **SSE System Project Office is located in the Reston, Va. Space Station Program Office, Information System Services Program Group**
  - Supported by Program Support Contractor (PSC) in Reston
  - Contract management support from JSC Institution, Spacecraft Software Division

- **Incremental Development**
  - SSE Interim System delivered 9/10/87
    - -- Used by LMSC to develop the Operational SSE
    - -- Available to SSP software developers for familiarization, training and early SSP development activities
  - First Operational SSE Release 11/10/89
    - -- Integrated Tools
    - -- No Proprietary Software
  - Additional Operational Releases each year

WHO ARE THE SSE USERS?

- **SSE users include all persons involved in the life cycle management of SSP software. They include:**
  - Software Project Managers
  - Requirements Analysts
  - Software Designers
  - Software Developers
  - Testers
  - Quality Managers
  - Software Configuration Managers
- **The majority of SSE users will be Work Package Contractors**
- **Other SSE users will include:**
  - NASA SSP organizations (e.g. MSIF)
  - KSC and non-prime contractors
  - Space Station users
  - International Partners
- **The SSE Users Working Group (SSEUWG) provides the forum for SSE user information exchange and input to the project**

SSE FUNCTIONALITY

- The SSE supports software development in Ada
  - Ada is baselined as the language of choice for the SSP
  - SSE can be expanded to support additional languages if required
- The SSE Ruleset provides software standards, guidelines and procedures to support software acquisition, integration, verification and maintenance
- The SSE Toolset provides all software tools necessary to acquire, integrate and deliver SSP operational software during all life cycle phases. SSE tools encompass the following functional areas:
  - SSE Process Management
  - Software Management Support
  - Software Production
  - Flight Software Integration, Test and Verification
  - Data Reconfiguration
  - Training
  - Library Management

CONCLUSIONS

- **The SSE provides a single common environment for the life-cycle management of all SSP operational software**
- **The SSE provides a mechanism for program-wide enforcement of approved standards and methodologies**
- **SSE support in the critical areas of software integration and testing will help to maintain SSP safety requirements**
- **Effective use of the SSE will minimize the cost of software ownership throughout the entire SSP life cycle**

Summary

- The SSFP understands the potential benefits of software commonality and reuse
- Some mandates are in place regarding software commonality
- The software reuse strategy is evolving
- The SSE will provide the mechanisms required to implement SSFP strategies for software reuse

N90-14800

LEVEL II SOFTWARE REUSE STUDY

Scott E. Herman
Grumman Program Support Contract

The Space Station Freedom Program (SSFP) Level II Software Reuse Study group was formed by Bob Nelson (NASA SSFP office) from members of the Information Systems Program Support Contract (PSC) team. The objectives of the study were to identify existing software reuse libraries, to identify existing reusability processes and experiences, to identify reusability analysis tools and users, and to provide recommendations for a software reusability process for the SSFP. To date the following have been delivered (1) definitions of commonality and reuse, (2) a report on existing software reuse libraries and library management systems, (3) a report on reuse process and methodology gleaned from software reuse experts, and (4) a report on software attributes for measuring commonality and reusability. Three implementation alternatives for a repository of reusable components were identified: centralized at the SSE Development Facility (SSEDF), a distributed approach across the network of Software Production Facilities, and a directory approach. A number of findings from the reuse study and several reuse strategy considerations were presented.

Study Objectives

Study group was formed by Bob Nelson at SSFP Level II from members of Information Systems Program Support Contract (PSC) team including Jim Flynn, Glenn Boyce, Scott Herman and Tammy Smith to:

1. Identify existing software reuse libraries
2. Identify existing reusability processes and experiences
3. Identify reusability analysis tools and users
4. Provide recommendations for a software reusability process for the SSFP.

Accomplishments to Date

Deliverables

1. Definitions of Commonality and Reuse
2. Report on existing software reuse libraries and library management systems
3. Report on reuse process and methodology from the software reuse teleconference
4. Report on software attributes for measuring commonality and reusability

Report on Reuse Libraries and Tools

* Provides a list of software libraries and library management systems

* Report includes for each system:

- Point of Contact
- Users
- Acquisition Requirements
- Description
- Processes and Methodology

Reusable Library and Library Management Systems Investigated

Reusable S/W Name	Library	Library Management Systems
*RSL (Intermetrics)	●	●
*RAPID	@	●
RSL (Ford)	●	●
GRACE	●	
COSMIC	●	
CAMP/AMPEE	●	●
TTCQF	●	@
SPC	●	@
JSC Ada S/W Library	●	

@ - Denotes Plans
* - Identifies Ruben Prieto - Diaz/ Peter Freeman Classification Approach Implementations

Report on Reuse Processes and Methodology From the Software Reuse Teleconference

* Need an overall model of the SSFP software development process.

* Domain analysis is very important.

* Identified need for a classification system using a controlled vocabulary to classify products.

* Good user interface to a software product library is very important.

* Must define incentives for contractors to build and use library products.

Report on Software Attributes for Measuring Commonality and Reusability

* Provides a general discussion of software classification for reuse

* Discusses domain-sensitive attributes

* Presents a perspective on High-level vs. Low-level attributes

* Recommends a specific set of classification Attributes

* Recommends attributes of information to be included in source code prologues

Reuse Study Attribute Sources:

1. **Booch Components**
2. **CAMP**
3. **Intermetrics**
4. **RAPID (Reusable Ada Packages For Information System Development)**
5. **Ruben Prieto - Diaz (Faceted Classification Scheme)**

Repository Implementation Alternatives

- **Location Of Actual Software Products**
- **Three Potential Schemes**

1). Centralized At SSEDF -
All Life-cycle Products Delivered To And Maintained At The SSEDF.

2). Distributed Approach -
SSE System Manages And Maintains Life-cycle Products Throughout The Network (Physical Location Transparent To The Users).

3). Directory Approach -
Developer Updates Library Mgmt System With Physical Location Of Life-cycle Product.

Findings:

1. Life cycle products from all phases of the software development process should be included in the reuse library.

2. Software development products of the SSFP should be included in a reuse library.

3. Access should be provided to all reusable life cycle products and notification should be given to users when new or updated products are added to the reuse library.

4. Reuse should be considered during each phase of the life cycle and an analysis of existing library products should be performed prior to initiating a new phase of the life cycle or a new project.

5. A standard taxonomy should be used to describe attributes of software life cycle products from each phase of the life cycle.

6. The use of a library management and classification system which incorporates the Faceted Classification scheme will meet the search/storage/retrieval requirements for software product reuse

Findings (conc)

7. Domain analysis should be performed at each stage of the life cycle to identify attributes of the required product and to use these attributes to identify common or reusable candidates.

8. A controlled attribute vocabulary for each phase of the life cycle will need to be established to facilitate domain analysis.

9. Maintenance responsibilities should reside with the developer or modifier. Reusers should be tracked and notified of errors and anomalies in usage of library components.

Reuse Strategy Considerations

1. Define a standard attribute vocabulary (similar to the one done by the U.S. Army) for use in describing software life cycle products. Such a vocabulary would facilitate Domain and Commonality analysis.

2. Continue the study to identify attributes for software life cycle products; to date, only the attributes for source code have been defined.

3. Create incentives for software developers: both for expending the extra effort to develop reusable products and for saving time and effort by using existing reusable products from the library.

N90-14801

AN SSE APPROACH TO REUSABILITY

David L. Badal
Lockheed Missiles and Spacecraft Co.

The SSE project has engineering analysis and design efforts under way for the development of the SSE reusability library management system. An ad hoc committee on reuse has been meeting for several months identifying design considerations and learning about Ruben Prieto-Diaz faceted classification, CAMP domain analysis, SPC activities, SEI activities, and SPS activities. A standard format was developed for the Ada prologue for reusable components (both specification and body). The SSE reusability process can be viewed as a transformation process with minimized losses and difficulties.

AGENDA

PURPOSE: D. BADAL

DESIGN CONSIDERATIONS: D. BADAL

- o AD HOC COMMITTEE
- o CLASSIFICATION SCHEMES
- o PROPOSED SOFTWARE COMPONENT PROLOG

DESIGN OVERVIEW C. SHOTTON

- o AUTOMATION OF REUSE PROCESS
- o CURRENT SSE SUPPORT FOR REUSE
- o FUTURE SSE REUSE CAPABILITIES

INITIAL ADA COMPONENTS Dr. T. MOEBES

- o ELEMENTARY FUNCTION MATH LIBRARY
- o IV&V METRICS

PURPOSE

TO PROVIDE INFORMATION ON THE ONGOING ENGINEERING ANALYSIS AND DESIGN EFFORTS FOR THE DEVELOPMENT OF THE SSE REUSABILITY LIBRARY MANAGEMENT SYSTEM

DESIGN CONSIDERATIONS

- o AD HOC COMMITTEE ON REUSE

- LIFE-CYCLE REUSABILITY OF PROCESSES AND PRODUCTS	Dr. C.W. McKAY
- TAXONOMY OF TAXONOMIES	P. RODGERS
- ACTIVITIES AT LANGLEY RESEARCH CENTER	S. VOIGT

DESIGN CONSIDERATIONS

- o CLASSIFICATION SCHEMES
  - RUBEN PRIETO-DIAZ FACETED CLASSIFICATION
  - CAMP DOMAIN ANALYSIS
  - SPC ACTIVITIES
  - SEI ACTIVITIES
  - SPS ACTIVITIES

ADA PROLOG FOR REUSABLE COMPONENTS

PROLOG TO THE VISIBLE INTERFACE (SPECIFICATION) OF ANY ADA PROGRAM UNIT:

```
- -
- -  COMPONENT_NAME:
- -  ABSTRACT:
- -  DESCRIPTION:
- -  REQUIREMENTS TRACE:
- -  TESTING TRACE:
- -  DISCLAIMER:
- -  DISTRIBUTION_AND_COPYRIGHT:
- -
- -  CONFIGURATION MANAGEMENT:
- -  SECURITY LEVEL:
- -  SUBSYSTEM CLASSIFICATION:
- -  KEY WORDS:
- -  TRACEABLE PREDECESSOR:
```

ADA PROLOG FOR REUSABLE COMPONENTS

```
- -  CREATION DATE       AUTHOR     ORGANIZATION
- -
- -
- -  REVISIONS
- -  DR#   DATE   VERSION  AUTHOR   ORGANIZATION
- -  PURPOSE
- -  PROGRAMMER'S INFORMATION
- -  DESIGN RATIONALE
- -
- -
- -
- -  TASKS SPAWNED:
- -
- -
- -  CONCURRENT ACCESS CONTROL:
- -
- -  DYNAMIC MEMORY MANAGEMENT:
```

ADA PROLOG FOR REUSABLE COMPONENTS

```
- -
- -  EXCEPTIONS PROPAGATED:
- -
- -  EXCEPTIONS HANDLED:
- -  KNOWN SIDE EFFECTS:
- -
- -  COMPILER DEPENDENCIES:
- -  LANGUAGE:
- -  VERSION #:
- -  COMPLEXITY:
- -  LINES OF EXECUTABLE CODE:
- -
- -  MACHINE DEPENDENCIES:
- -
- -  ASSUMPTIONS AND LIMITATIONS OF THIS UNIT:
- -
- -  WAIVERS:
```

ADA PROLOG FOR REUSABLE COMPONENTS

```
- -  PROPOSED EXTENSIONS AND MODIFICATIONS:
- -
- -  DESCRIPTION FOR REUSERS:
```

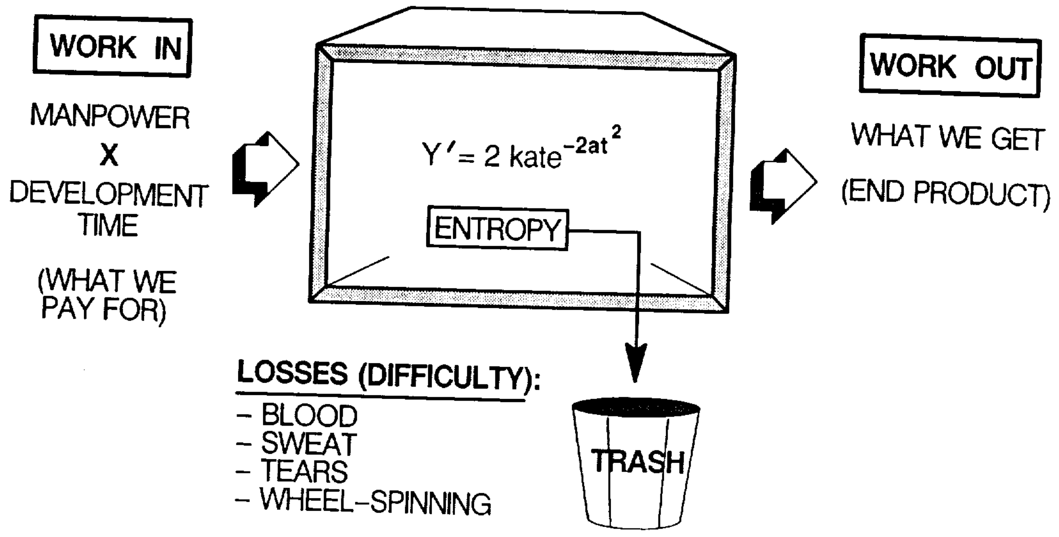
ADA PROLOG FOR REUSABLE COMPONENTS

PROLOG FOR THE IMPLEMENTATION (BODY) SECTION OF ADA PROGRAM UNITS:

```
- -
- -  COMPONENT NAME:
- -
- -  DESIGN RATIONALE:
- -
- -  SPECIAL CONSTRUCTS USED:
- -  TESTING TRACE:
- -  PROPOSED EXTENSIONS AND MODIFICATIONS:
- -
- -  DESCRIPTION FOR REUSERS:
```

SSE REUSABILITY PROCESS

(TRANSFORMATION PROCESS)



N90-14802

SUPPORT FOR LIFE-CYCLE PRODUCT REUSE IN NASA'S SSE

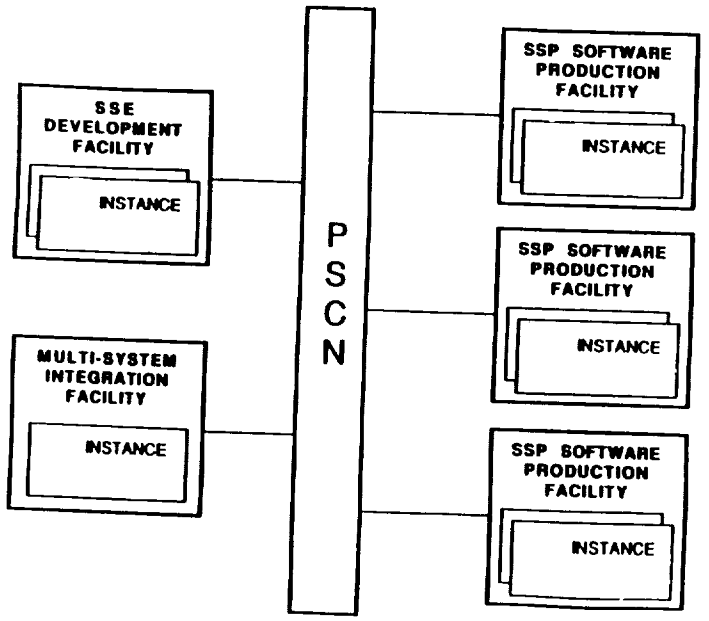
Charles Shotton
Planning Research Corporation

The Software Support Environment (SSE) is a software factory for the production of Space Station Freedom Program operational software. The SSE is to be centrally developed and maintained and used to configure software production facilities in the field. The PRC product TTCQF provides for an automated qualification process and analysis of existing code that can be used for software reuse. The interrogation subsystem permits user queries of the reusable data and components which have been identified by an analyzer and qualified with associated metrics. The concept includes reuse of non-code life-cycle components such as requirements and designs. Possible types of reusable life-cycle components include templates, generics, and "as-is" items. Qualification of reusable elements requires analysis (separation of candidate components into primitives), qualification (evaluation of primitives for reusability according to reusability criteria), and loading (placing qualified elements into appropriate libraries). There can be different qualifications for different installations, methodologies, applications and components. Identifying reusable software and related components is labor-intensive and is best carried out as an integrated function of an SSE.

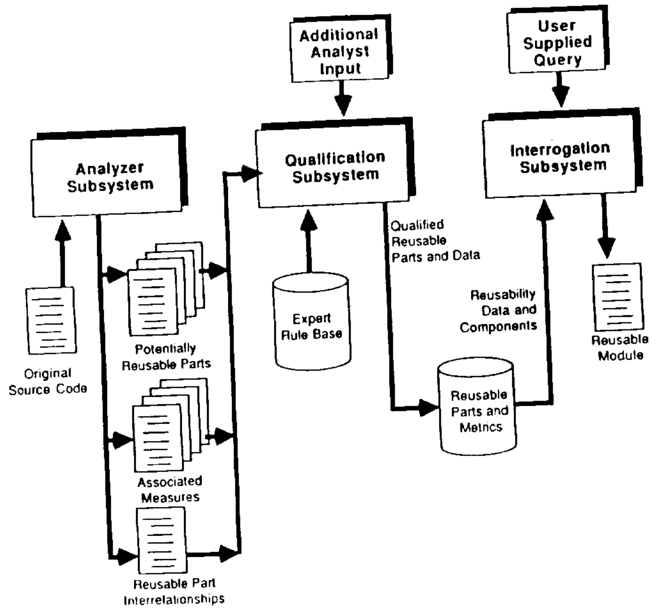
SOFTWARE SUPPORT ENVIRONMENT (SSE)

- SOFTWARE FACTORY FOR PRODUCTION OF SPACE STATION PROGRAM OPERATIONAL SOFTWARE
  - TOOLS
  - RULES
  - PROCEDURES
  - HARDWARE SPECIFICATIONS
- SSE CENTRALLY DEVELOPED AND MAINTAINED
- SOFTWARE PRODUCTION FACILITIES AND THEIR PROJECTS ARE CONFIGURED FROM SSE

SSE SYSTEM ELEMENTS


SSE DEVELOPMENT FACILITY
INSTANCE
MULTI-SYSTEM INTEGRATION FACILITY
INSTANCE
PSCN
SSP SOFTWARE PRODUCTION FACILITY
INSTANCE
SSP SOFTWARE PRODUCTION FACILITY
INSTANCE
SSP SOFTWARE PRODUCTION FACILITY
INSTANCE

REUSABILITY DESIGN CONCEPT



INTERROGATION SUBSYSTEM APPROACH

- o USE WELL-UNDERSTOOD, EXISTING DATA BASE TECHNOLOGY, AND TAXONOMIES
- o ALLOWS KEY WORD RETRIEVAL OF REUSABLE COMPONENTS
- o ALL RELATED COMPONENTS CAN BE IDENTIFIED AND RETRIEVED
- o IMPLEMENTATION WILL BUILD UPON EXISTING AND FUTURE INDUSTRY TAXONOMIES, TECHNIQUES, AND STANDARDS

REUSE OF LIFE-CYCLE COMPONENTS

- REUSE OF NON-CODE LIFE-CYCLE COMPONENTS IS REQUIRED

- NON-CODE COMPONENTS INCLUDE:

  - REQUIREMENTS
  - PRELIMINARY DESIGN PRODUCTS
  - DETAILED DESIGN (ADA PDL)
  - TESTS, TEST DATA, AND OTHER TESTING RESOURCES
  - METHODS AND PROCEDURES FOR LIFE-CYCLE DEVELOPMENT
  - TOOLS SUPPORTING DEVELOPMENT

- REUSABLE COMPONENTS CAN BE QUALIFIED FOR SPECIFIC:

  - INSTALLATIONS
  - METHODOLOGIES
  - APPLICATIONS
  - MISSIONS

POSSIBLE TYPES OF REUSABLE LIFE-CYCLE COMPONENTS

- o TEMPLATES - OUTLINE FORM OF A COMPONENT
- o GENERICS - TAILORABLE FORM OF A COMPONENT
- o AS-IS - COMPONENTS REUSED WITHOUT MODIFICATION

REUSING COMPONENTS

- o THIS APPROACH IDENTIFIES CANDIDATE COMPONENTS FOR REUSE SELECTION BY A DEVELOPER
- o GENERICS AND TEMPLATES ARE COPIED TO USER'S WORK AREA FOR MODIFICATION PRIOR TO BEING PLACED UNDER CONFIGURATION CONTROL
- o MODIFIED AND AS-IS COMPONENTS ARE LOADED DIRECTLY INTO AN SSE CONFIGURED OBJECT DATA BASE FOR TESTING

REUSING COMPONENTS

- RELATED REUSABLE COMPONENTS IN LATER PHASES ARE LOADED INTO THE SSE PROJECT OBJECT BASE WHEN AN AS-IS REUSABLE IS SELECTED
- IF AS-IS COMPONENT MUST BE REWORKED FOR THIS PROJECT, THE SSE MAY REQUIRE ITS DESCENDANTS TO BE REWORKED
- ALL MODIFIED REUSABLES ARE CYCLED BACK TO REUSABLE LIBRARIES FOR QUALIFICATION AND LOADING INTO THE LIBRARIES

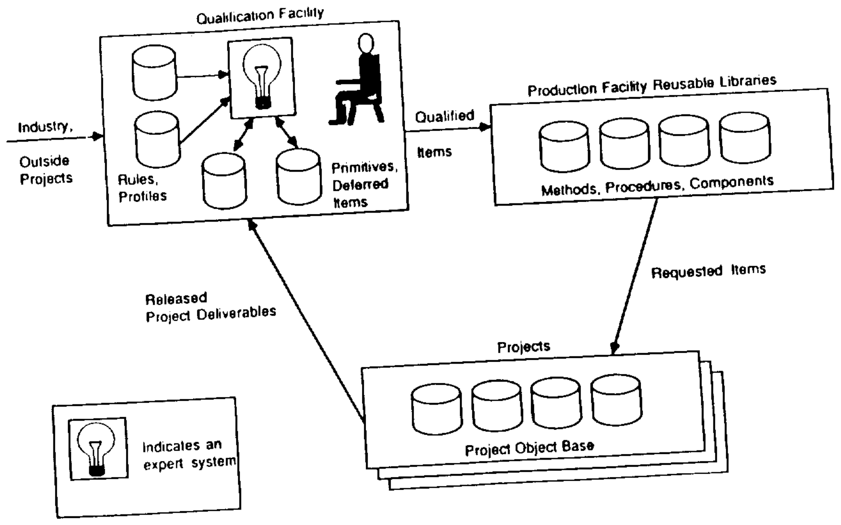
QUALIFICATION

- QUALIFICATION OF REUSABLE ELEMENTS REQUIRES:
  - ANALYSIS - SEPARATION OF CANDIDATE COMPONENT INTO PRIMITIVES
  - QUALIFICATION - EVALUATE PRIMITIVES FOR REUSABILITY ACCORDING TO REUSABILITY CRITERIA
  - LOADING - PLACING QUALIFIED ELEMENTS INTO APPROPRIATE LIBRARIES
- MANUAL INTERROGATION ALLOWS FOR INDEPENDENT ASSESSMENT OF ELEMENTS BEFORE LOADING
- QUALIFICATION RULES AND CRITERIA WOULD BE EVOLVABLE AND DEFINABLE TO REFLECT SITE/PROJECT/SYSTEM NEEDS

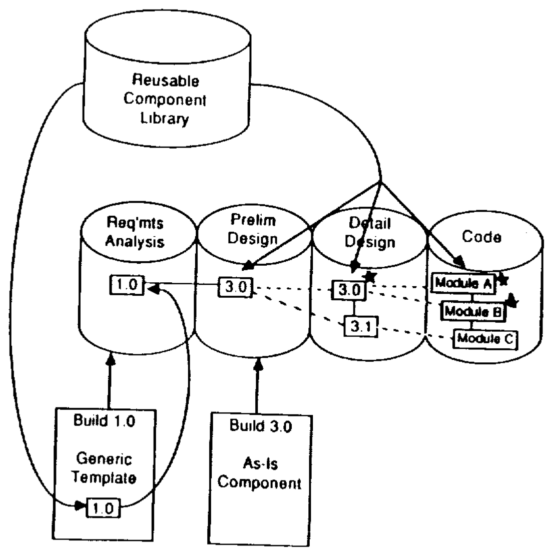
QUALIFICATION

- o PRIMITIVE PROFILES WILL BE MODIFIABLE TO ACCOMMODATE NEW DEVELOPMENT TECHNIQUES OR TO ENFORCE STANDARDS
- o CAN HAVE DIFFERENT QUALIFICATION FOR DIFFERENT
  - INSTALLATIONS
  - METHODOLOGIES
  - APPLICATIONS
  - COMPONENTS (e.g., CODE vs. DESIGN)

QUALIFICATION


Qualification Facility
Industry, Outside Projects
Rules, Profiles
Primitives, Deferred Items
Qualified Items
Production Facility Reusable Libraries
Methods, Procedures, Components
Requested Items
Released Project Deliverables
Projects
Project Object Base
Indicates an expert system

QUALIFICATION



CONCLUSIONS

- o IT WAS DETERMINED THAT, IN SOME CASES, 60% OF THE SOFTWARE WAS REUSABLE
- o IDENTIFYING REUSABLE SOFTWARE AND RELATED COMPONENTS IS LABOR-INTENSIVE AND IS BEST CARRIED OUT AS AN INTEGRATED FUNCTION OF AN SSE
- o ADA PROVIDES SUPPORT FOR REUSE DURING ANALYSIS AND IDENTIFICATION, BUT SPECIAL CONSIDERATION MUST BE GIVEN TO STORAGE AND RETRIEVAL OF REUSABLE ADA COMPONENTS, INCLUDING THE MAINTENANCE OF RELATIONSHIPS BETWEEN CODE AND OTHER LIFE-CYCLE PRODUCTS
- o SOME ASPECTS OF AUTOMATING SOFTWARE REUSE, ENFORCING SOFTWARE DEVELOPMENT STANDARDS, AND AUTOMATING IV&V FUNCTIONS ARE SIMILAR
- o AI TECHNIQUES CAN BE APPLIED IN HARMONY WITH "CONVENTIONAL" SOFTWARE WHEN ATTACKING PROBLEMS WITHIN SSEs

N90-14803

INITIAL ADA COMPONENTS EVALUATION

Dr. Travis Moebes
SAIC

SAIC has the responsibility for independent test and validation of the SSE. They have been using a mathematical functions library package implemented in Ada to test the SSE IV&V process. The library package consists of elementary mathematical functions and is both machine and accuracy independent. The SSE Ada components evaluation includes code complexity metrics based on Halstead's software science metrics and McCabe's measure of cyclomatic complexity. Halstead's metrics are based on the number of operators and operands on a logical unit of code and are compiled from the number of distinct operators, distinct operands, and total number of occurrences of operators and operands. These metrics give an indication of the physical size of a program in terms of operators and operands and are used diagnostically to point to potential problems. McCabe's Cyclomatic Complexity Metrics (CCM) are compiled from flow charts transformed to equivalent directed graphs. The CCM is a measure of the total number of linearly independent paths through the code's control structure. These metrics were computed for the Ada mathematical functions library using Software Automated Verification and Validation System (SAVVAS), the SSE IV&V tool. A table with selected results was shown, indicating that most of these routines are of good quality. Thresholds for the Halstead measures indicate poor quality if the length metric exceeds 260 or difficulty is greater than 190. The McCabe CCM indicated a high quality of software products. The SSE will include the Ada version of SAVVAS that may be used for computing these code complexity metrics.

INITIAL ADA COMPONENTS EVALUATION MATHEMATICAL FUNCTIONS LIBRARY PACKAGE

- IMPLEMENTED IN ADA BY L. J. GALLAHER, Ph.D
- LIBRARY PACKAGE OF ELEMENTARY MATHEMATICAL FUNCTIONS
  - SIN, COS, SINH, LOG, ETC.
- ROUTINES ARE BOTH MACHINE AND ACCURACY INDEPENDENT
  - ACCURACY DETERMINED WHEN LIBRARY PACKAGES ARE INTEGRATED INTO USER PROGRAM

INITIAL ADA COMPONENTS EVALUATION MATHEMATICAL FUNCTIONS LIBRARY PACKAGE

- SPECIFICATION FOR ELEMENTARY MATH FUNCTIONS PACKAGE

```
generic type real is digits < >;
package pac_el_fun is --
        function    exp(x : real)          return  real;
        function     ln(x : real)          return  real;
        function log10(x : real)           return  real;
        function   sqrt(x : real)          return  real;
        function   cbrt(x : real)          return  real;
        function    sin(x : real)          return  real;
        function    cos(x : real)          return  real;
        function   atan(x : real)          return  real;
        function   cosh(x : real)          return  real;
        function   sinh(x : real)          return  real;
        function    tan(x : real)          return  real;
        function   tanh(x : real)          return  real;
        function  atanh(x : real)          return  real;
        function   asin(x : real)          return  real;
        function   acos(x : real)          return  real;
        function  asinh(x : real)          return  real;
        function  acosh(x : real)          return  real;
        function atan2(y, x : real) return real;
        function    "**"(x, y : real) return real;
        function  root(n : integer; y : real) return real;
end pac_el_fun
```

- APPROXIMATIONS CALCULATED DIRECTLY FROM CHEBYSHEV POLYNOMIALS USING A METHOD INTRODUCED BY C. W. CLENSHAW, "CHEBYSHEV SERVICES FOR MATHEMATICAL FUNCTIONS," "NATIONAL PHYSICS LABORATORY MATHEMATICAL TABLES," VOL. 5, HER MAJESTY'S STATIONERY OFFICE, LONDON, 1962

- TECHNICAL REPORT ON THE LIBRARY GIVEN BY L. J. GALLAHER, "A LIBRARY OF ELEMENTARY MATH FUNCTIONS IN ADA," IRAD R626, LOCKHEED-GEORGIA COMPANY, MARIETTA, GEORGIA, JANUARY, 1987

INITIAL ADA COMPONENTS EVALUATION SSE IV&V CODE COMPLEXITY METRICS

WHAT ARE THE TYPES OF CODE COMPLEXITY METRICS?

- CODE COMPLEXITY METRICS
  - HALSTEAD'S SOFTWARE SCIENCE METRICS
  - McCABE'S MEASURE OF CYCLOMATIC COMPLEXITY

- COMPUTED IN SSE IV&V'S SOFTWARE AUTOMATED VERIFICATION & VALIDATION SYSTEM (SAVVAS)

WHY USE HALSTEAD'S SOFTWARE SCIENCE METRICS?

- COMPILED FROM THE NUMBER OF DISTINCT OPERATORS, DISTINCT OPERANDS, AND TOTAL NUMBER OF OCCURRENCES OF OPERATORS AND OPERANDS
- GIVES AN INDICATION OF THE PHYSICAL SIZE OF A PROGRAM IN TERMS OF OPERATORS AND OPERANDS. VARIOUS SIZE METRICS ARE GIVEN
- USED DIAGNOSTICALLY TO POINT TO POTENTIAL PROBLEMS

- **HALSTEAD'S SOFTWARE SCIENCE METRICS**
  - BASED ON THE NUMBER OF OPERATORS AND OPERANDS ON A LOGICAL UNIT OF CODE
  - OPERATORS INCLUDE ARITHMETIC OPERATORS, BOOLEAN OPERATORS, DELIMITERS
  - OPERANDS ARE VARIABLES AND CONSTANTS
  - FOR EACH UNIT OF CODE LET:

    n1 = THE NUMBER OF DISTINCT OPERATORS
    n2 = THE NUMBER OF DISTINCT OPERANDS
    N1 = THE TOTAL NUMBER OF OCCURRENCES OF THE OPERATORS
    N2 = THE TOTAL NUMBER OF OCCURRENCES OF THE OPERANDS

INITIAL ADA COMPONENTS EVALUATION
SSE IV&V CODE COMPLEXITY METRICS

- LENGTH: $N = N1 + N2$
- VOCABULARY: $W = n_1 + n_2$
- VOLUME: $V = N \times \log_2 W$
- LEVEL: $L = (2 \times n_2)/(n_1 \times N_2)$
- DIFFICULTY: $D = 1/L$
- EFFORT: $E = V/L$
- ERROR ESTIMATE: $B = E^{2/3}/E_0 ; E_0 \cong 3000$

  B = NO. OF BUGS IN A PROGRAM

- INTERPRETATION OF HALSTEAD'S METRICS

  THE LENGTH N SERVES AS A MEASURE OF MODULARITY. A LENGTH OF GREATER THAN 260 INDICATES POOR QUALITY CODE. THE CODE SHOULD (PROBABLY) BE REDUCED TO MORE AND SMALLER MODULES

  THE VOLUME V REPRESENTS THE SIZE (IN BITS) OF A LOGICAL UNIT OF CODE

  THE LEVEL L IS A MEASURE THAT RELATES TO THE EFFORT OF WRITING, PROPENSITY (INCLINATION) FOR ERROR, AND EASE OF UNDERSTANDING OF A LOGICAL UNIT OF CODE

INITIAL ADA COMPONENTS EVALUATION
SSE IV&V CODE COMPLEXITY METRICS

THE DIFFICULTY D, THE RECIPROCAL OF LEVEL, INDICATES THE DIFFICULTY IN UNDERSTANDING AND MAINTAINING THE CODE. A DIFFICULTY GREATER THAN 190 TENDS TO INDICATE A POOR QUALITY OF CODE

THE EFFORT E IS A MEASURE OF THE RELATIVE AMOUNT OF WORK INVOLVED IN PRODUCING A PIECE OF CODE

THE ERROR ESTIMATE B (BUGS) IS THE ESTIMATED NUMBER OF ERRORS IN THE CODE.

Operator Parameters

Operator	j	$f_{1,j}$
;	1	9
:=	2	6
( ) or BEGIN...END	3	5
IF	4	3
=	5	3
/	6	1
-	7	1
x	8	1
RETURN	9	1
EXIT	10	1
	$n_1 = 10$	$N_1 = 31$

INITIAL ADA COMPONENTS EVALUATION SSE IV&V CODE COMPLEXITY METRICS

Operand Parameters

Operand	j	$f_{2,j}$
B	1	6
A	2	5
O	3	3
R	4	3
G	5	2
GOD	6	2
	$n_2=6$	$N_2=21$

WHY USE McCABE'S CYCLOMATIC COMPLEXITY METRICS?

- o COMPILED FROM FLOWCHARTS TRANSFORMED TO EQUIVALENT DIRECTED GRAPHS, THE NUMBER OF EDGES OF THE GRAPHS, THE NUMBER OF NODES OF THE GRAPHS AND THE NUMBER OF SEPARATE PARTS OF THE GRAPH
- o THE McCABE CYCLOMATIC COMPLEXITY METRIC (CCM) ASSISTS IN BREAKING UP A SOFTWARE PROGRAM TO COMPONENTS THAT HAVE A SMALL CCM <10
- o PROGRAMS WITH LARGE CCM SHOULD HAVE MORE ERRORS DURING DEVELOPMENT

WHY USE McCABE'S CYCLOMATIC COMPLEXITY METRICS? (Continued)

- o THE CCM MAY BE USED TO DETERMINE THE MINIMUM NUMBER OF PATHS NEEDED.
- o THE CCM RELATES ONLY TO LOGICAL COMPLEXITY. THE CCM SHOULD BE USED IN CONJUNCTION WITH OTHER METRICS

McCABE'S CYCLOMATIC COMPLEXITY METRIC

McCABE'S CYCLOMATIC COMPLEXITY METRIC IS A MEASURE OF THE TOTAL NUMBER OF LINEARLY INDEPENDENT PATHS THOUGH THE CODES CONTROL STRUCTURE

TO CALCULATE CYCLOMATIC COMPLEXITY, FLOW-CHARTS ARE TRANSFORMED TO EQUIVALENT DIRECTED GRAPHS

McCABE'S CYCLOMATIC COMPLEXITY METRIC (Continued)

COMPLEXITY MEASURE OF A PROGRAM IS A FUNCTION OF THE NUMBER OF DECISIONS IN A PROGRAM AND IS GIVEN BY A SINGLE NUMBER KNOWN AS A CYCLOMATIC NUMBER

COMPLEXITY MEASURE IS INDEPENDENT OF THE PHYSICAL SIZE OF THE PROGRAM

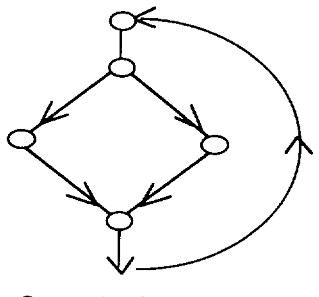
DEFINITIONS:

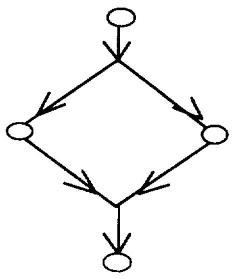
A GRAPH IS A TREE STRUCTURE CONSISTING OF NODES CONNECTED BY BRANCHES

A DIRECTED GRAPH IS A GRAPH IN WHICH A DIRECTION OR FLOW IS ASSOCIATED WITH EVERY BRANCH

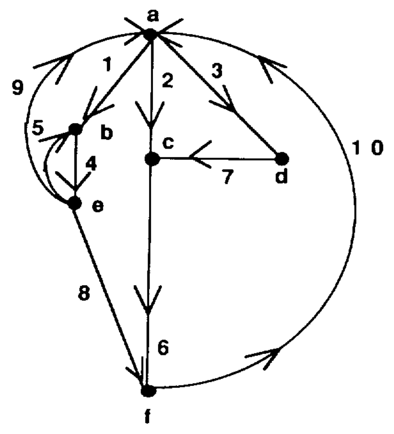
DEFINITIONS: (Continued)

A STRONGLY CONNECTED GRAPH IS A GRAPH THAT HAS A UNIQUE ENTRY AND EXIT NODE AND EACH NODE CAN BE REACHED FROM EVERY OTHER NODE


Strongly Connected


Not Strongly Connected

EXAMPLE
CONTROL GRAPH G



SOME POSSIBLE PATHS

a b e f

b e b

a b e a

a c f a

a d c f a

a b e f a 2b e b a b e a

DEFINITION:

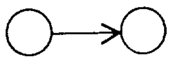
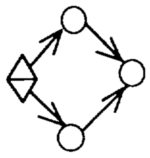
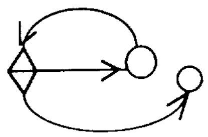
THE CYCLOMATIC NUMBER V(G) OF A GRAPH G WITH n NODES AND e BRANCHES AND p CONNECTED COMPONENTS IS

$V(G)=e-n+2P;$

IN THE EXAMPLE CONTROL GRAPH G:

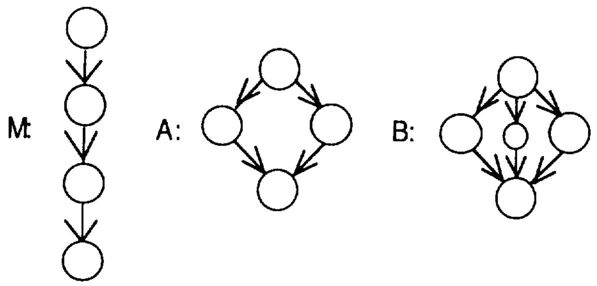
NUMBER OF NODES = 6
NUMBER OF BRANCHES = 10
$V(G) = 10-6+2(1) = 6$

GRAPH EXAMPLES:

CONTROL	STRUCTURE	CYCLOMATIC COMPLEXITY $V = e - n + 2p$
SEQUENCE		$V = 1 - 2 + 2 = 1$
IF THEN ELSE		$V = 4 - 4 + 2 = 2$
WHILE		$V = 3 - 3 + 2 = 2$

GRAPH EXAMPLE WITH p = NO. OF COMPONENTS = 3

SUPPOSE A PROGRAM M AND TWO CALLED SUBPROGRAMS A AND B HAVE THE FOLLOWING CONTROL STRUCTURE:



THEN G = MUAUB AND p = 3
THEN V(G) = V(MUAUB) = 13-13+2x3 = 6
NOTE: V(MUAUB) = V(M)+V(A)+V(B)

- PROPERTIES OF THE CYCLOMATIC COMPLEXITY

1. V(G)≥1

2. V(G) IS THE MAXIMUM NUMBER OF (LINEARLY) INDEPENDENT PATHS IN G; IT IS THE SIZE OF A BASIS SET (i.e., ALL COMBINATIONS OF PATHS IN THE CODE ARE MADE UP FROM PATHS IN G)

3. INSERTING OR DELETING FUNCTIONAL STATEMENTS TO G DOES NOT AFFECT V(G)

4. G HAS ONLY ONE PATH IF AND ONLY IF V(G)=1

PROPERTIES OF THE CYCLOMATIC COMPLEXITY
(Continued)

5. INSERTING A NEW EDGE IN G INCREASES V(G) BY UNITY

6. V(G) DEPENDS ONLY ON THE DECISION STRUCTURE OF G

   V(G) ASSISTS IN BREAKING UP A SOFTWARE PROGRAM TO COMPONENTS THAT HAVE A SMALLER COMPLEXITY NUMBER. McCABE RECOMMENDS THAT EACH COMPONENT G HAS A V(G) LESS THAN 10

SAVVAS-ADA-McCABE'S CCM

o COUNT THE NUMBER OF "CASE" STATEMENT BRANCHES AND SET THIS NUMBER EQUAL TO m_1

```
CASE TODAY IS

1 )  WHEN MON        =>  OPEN_ACCOUNTS;
                         COMPUTE_INITIAL BALANCE;
1 )  WHEN TUE..THU   =>  GENERATE_REPORT (TODAY);
1 )  WHEN FRI        =>  COMPUTE_CLOSING_BALANCE;
                     =>  CLOSE_ACCOUNTS:
1 )  WHEN SAT/SUN    =>  NULL
---
4    END CASE;
                     4 BRANCHES
```

o COUNT THE NUMBER OF "IF STATEMENT" BRANCHES AND SET THIS NUMBER TO m_2

```
1 )  IF WEATHER_CONDITION = RAIN THEN
          COMPUTE_RAINFALL;
1 )  ELSIF WEATHER_CONDITION = SUNSHINE THEN
          COMPUTE_HUMIDITY:
1 )  ELSE
          COMPUTE_PRES;
     END IF;
___
3
```

3 BRANCHES

o COUNT THE NUMBER OF "LOOP BRANCHES" AND SET THIS NUMBER TO m_3

```
1 )  FOR I  IN 1..10 LOOP
1 )       FOR J IN 1..20 LOOP
               IF A(I,J) = 0 THEN
                    M:=I ;
                    N:=J;
                    EXIT FIND;
               END IF;
          END LOOP;
     END LOOP FIND;
```

2 LOOP BRANCHES   1 "IF STATEMENT" BRANCHES

3 BRANCHES

$CCM=LOG_2\ m_1+m_2+m_3$

INITIAL ADA COMPONENTS EVALUATION SAVVAS CCM RESULTS APPLIED TO ADA MATHEMATICAL FUNCTIONS LIBRARY

UNIT NAME	McCABE V(g)	HALSTEAD's Length Metric (260)	HALSTEAD's Difficulty Metric (190)
Average	3	82	16.018
ln	1	32	5.25
tanh	3	74	16.7
t_aux_fun	1	163	25.84
pac_aux_fun	1	215	32.045
remainder	4	90	34.833
pac_el_fun	1	868	19.989

INITIAL ADA COMPONENTS EVALUATION SUMMARY

o CCM DIRECTED IV&V APPLIED TO THE SSE ADA MATHEMATICAL FUNCTION LIBRARY PACKAGE INDICATED A HIGH QUALITY OF SOFTWARE PRODUCTS

o IN HALSTEAD'S SOFTWARE SCIENCE METRICS, A LENGTH OF GREATER THEN 260 OR DIFFICULTY GREATER THAN 190 TENDS TO INDICATE POOR QUALITY CODE

o McCABE HAS SUGGESTED THAT A CYCLOMATIC COMPLEXITY OF 10 SHOULD BE THE UPPER LIMIT FOR V(G) (CYCLOMATIC COMPLEXITY MEASURE FOR A GRAPH G)

o THE ADA VERSION OF SAVVAS MAY BE USED FOR COMPUTING CODE COMPLEXITY METRICS

N90-14804

APPLICATION OF REUSABLE SOFTWARE COMPONENTS AT THE SEI

Robert Holibaugh
Software Engineering Institute

Robert Holibaugh of the Software Engineering Institute described a project which is studying the application of reusable software components. The primary goals are to gain practical experience with state-of-the-art reusable components, methods, and tools and to capture the lessons learned in the application of reuse technology. In addition the project will assess the impact of reuse on the software development process and products and will identify and validate the information that facilitates software reuse during system development. The project includes two tasks - a reuse experiment and a redevelopment effort. The reuse experiment will define a life cycle and a methodology for reuse-based development, and define and implement a data collection mechanism for measuring the development. The redevelopment effort will construct a reuse test bed and will redevelop and realistically test subsystems from an embedded mission-critical real-time application. The reuse experiment will produce several products including a tested real-time application, reuse-based components and tools evaluation, a reuse-based development method, a framework for data collection, a framework for measuring productivity, and lessons learned data. Successful development with reusable components will require a rich set of components and an integrating methodology. The Tomahawk Land Attack Missile system is the application for the redevelopment effort. A number of goals and questions relating to the reuse experiment were presented. The project environment included several types of workstations, target hardware, and a number of software support tools. Several reports from the project are already available.

Agenda

- Project Goals
- Reuse Experiment
- Project Environment
- Project Status

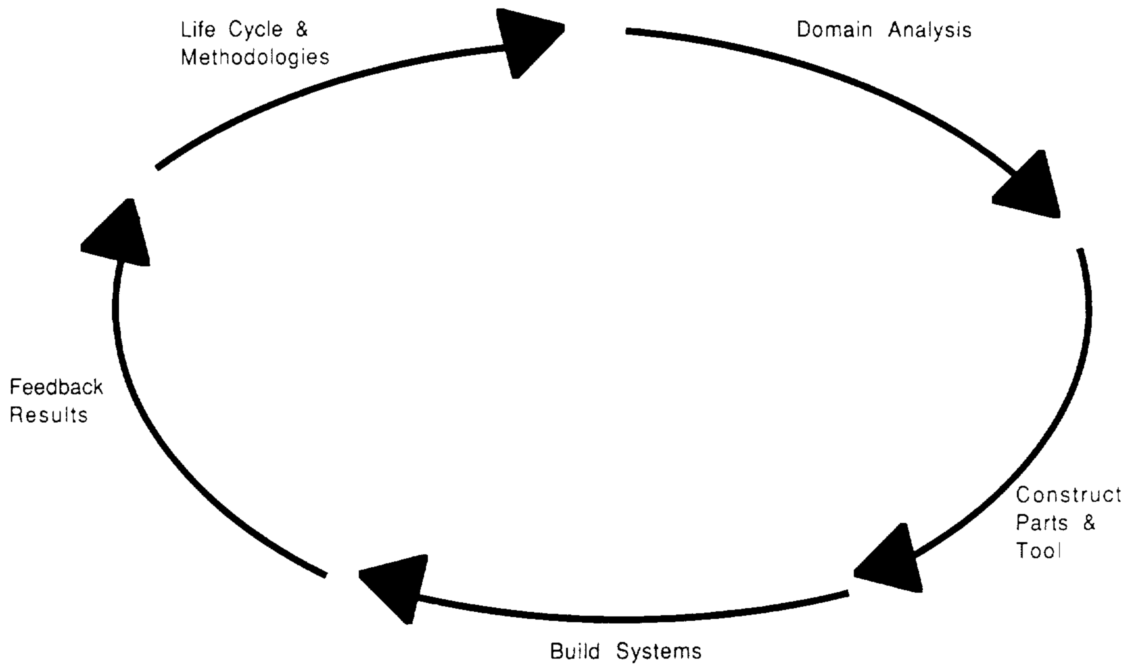
Goals

- Gain practical experience with state-of-the-art reusable components, methods, and tools; and capture lessons learned in the application of reuse technology
- Assess the impact of reuse on software development process and products (in particular, on design)
- Identify and validate the information that facilitates software reuse during system development

Project Tasks

- Reuse Experiment
  - define life cycle and a methodology for reuse-based development
  - define and implement data collection mechanism for measuring the development
- Redevelopment Effort
  - construct reuse test bed
  - redevelop and realistically test subsystems from an embedded MCCR application

Reuse Cycle



Reuse Experiment Motivation

• Motivation:	Integrate and improve the application of reuse
• Purpose:	Investigate impact of systematic reuse on 'real' development
• Value:	Establish empirically supported guidelines for reuse

Reuse Experiment Products

- Tested real-time application
- Reuse-based components and tools evaluation
- Reuse-based development method
- Framework for data collection
- Framework for measuring productivity
- Lessons learned/development data

Reuse Position

Given sufficiently rich and powerful set of reusable components, methods, and tools for an application domain, and a (integrating) methodology for systematically applying reuse
THEN successful development will:

- require significant effort for reuse related tasks
- encounter new problems
- benefit from reuse over multiple projects

Subsystem Redevelopment

- Application - Tomahawk Land Attack Missile (TLAM)
  - Acquired domain-specific reusable components - CAMP
  - Acquired domain expertise - Raytheon
  - Acquired interested DOD program office - Cruise Missile Program Office
- Acquired the original requirements, functional specification, and design

Subsystem Redevelopment (con)

- Develop and test the subsystems under constraints "similar" to MCCR contractors (e.g. 2167A and ISP)
- "Monitor" the development of the subsystem (Basili86)

Experiment Goals

- Describe the impact of software reuse on process and products (specifically design)
- Describe the use of reuse-based resources: components, methods, tools
- Identify information (through applications, documentation, reports, etc.) that will facilitate reuse
- Capture and disseminate lessons learned

Questions

- Reuse extent & frequency(Q1)
- Training, experience, & support(Q2)
- Relative contribution of reuse(Q3)
- Related reqs & design decisions(Q4)
- Properties of the System(Q5)
- Impact on quality(Q6)
- Integral concept(Q7)
- Effectiveness of CAMP(Q8)
- U.Md/NASA standard data(Q9)

Metrics

- Document the capabilities of development staff
- Instrument the development process
- Evaluate the development products
- Evaluate the reusable components, methods, and tools

Project Environment

- Hardware:
  - VAX Cluster provides a workstation for each developer
  - Symbolics 3670 supports AMPEE
  - IBM PC-AT supports Asset Library System
  - Motorola 68020 Target Hardware
- Software:
  - VAX Ada Tool Set (editors, compiler, debugger, configuration manager, etc.)

- Statemate
- ADADL (Ada PDL tool set)
- Interpretative Simulation Program (ISP) from NWC

ARSC Reusable Software

- Component libraries
  - Common Ada Missile Packages
  - Booch Abstract Data Types
  - EVB Abstract Data Types
  - Ada Software Repository
- Ada Missile Parts Engineering Expert (AMPEE)
- GTE Asset Library System

GTE Asset Library System

- Uses 'faceted' approach to Reusable Software classification
  - Missile operational parts
  - Kalman filter and math parts
  - General purpose parts
- Runs on IBM PC AT
- GTE has assisted in facet identification and parts classification

Testing Considerations

- Required to verify/validate performance of software
- Required to verify accuracy of experiment results
- Realistic to project scope
- Credible to external reviewers

Testing Options

- Code Inspections
- Unit testing on VAX
- Simulation on VAX
- Simulation on VAX with target machine(Motorola 68020) in loop
- Simulation with hardware in loop
- Flight test

Transition

- Work with GTE on evaluation of ALS and report findings
- Work with Raytheon/??? on evaluating the system attributes
- Report on software experimentation and data collection mechanisms
- Report on reuse-based methodology
- Report on lessons learned from the experiment

Project Reports

- Phase I Test bed Description: Requirements and Selection Guidelines (CMU/SEI-TR-88-013)
- Subsystem Redevelopment: Analysis (CMU/SEI/TR-88-014)
- Perspective on Software Reuse (CMU/SEI-TR-88-022)
- Experiment Design Report: High Level Design (CMU/SEI-TR-88-32)
- Experiment Planning for Software Development (Dec 1988)

- **Software Methodology in the Harsh Light of Economics (Dec 1988)**
- **Experiment Design Report: Detailed Design (Dec 1988)**
- **Reusable Software Component Construction (SEI Affiliates Symposium, Jun 1989)**

Planned Project Outputs

- Project Reports
  - Reuse Life Cycle and Methodology Report (Feb 1989)
  - Classification of Reusable Components (Apr 1989)
  - Final Project Report (Dec 1989)
- Project Presentations
  - Reuse, Where to Begin and Why (Feb 1989)

- Classification of Reusable Components (Mar 89)
- CAMP Training (Apr 1989)
- Reusable Requirements (May 1989)
- CAMP Users Workshop (Jun 1989)

Completed Tasks

- Jul 87 - Testbed definition
- Sep 87 - Domain selection
- Nov 87 - Raytheon affiliate
- Dec 87 - Installation of Vax Cluster & Symbolics
- Jan 88 - Experiment review
- Apr 88 - Software Development Plan
- Aug 88 - Requirements analysis
- Oct 88 - Software Specification Review

N90-14805

TECHNOLOGY TRANSFER IN SOFTWARE ENGINEERING

Dr. Peter C. Bishop
University of Houston-Clear Lake

The University of Houston-Clear Lake is the prime contractor for the AdaNET Research Project under the direction of NASA Johnson Space Center. AdaNET was established to promote the principles of software engineering to the software development industry. AdaNET will contain not only environments and tools, but also concepts, principles, models, standards, guidelines and practices. Initially, AdaNET will serve clients from the U.S. government and private industry who are working in software development. It will seek new clients from those who have not yet adopted the principles and practices of software engineering. Some of the goals of AdaNET are to become known as an objective, authoritative source of new software engineering information and parts, to provide easy access to information and parts, and to keep abreast of innovations in the field.

Mission

AdaNET is an electronic distribution network
(the electronic marketplace)

for software engineering

where those in the field

can exchange information and engineering parts.

AdaNET will also

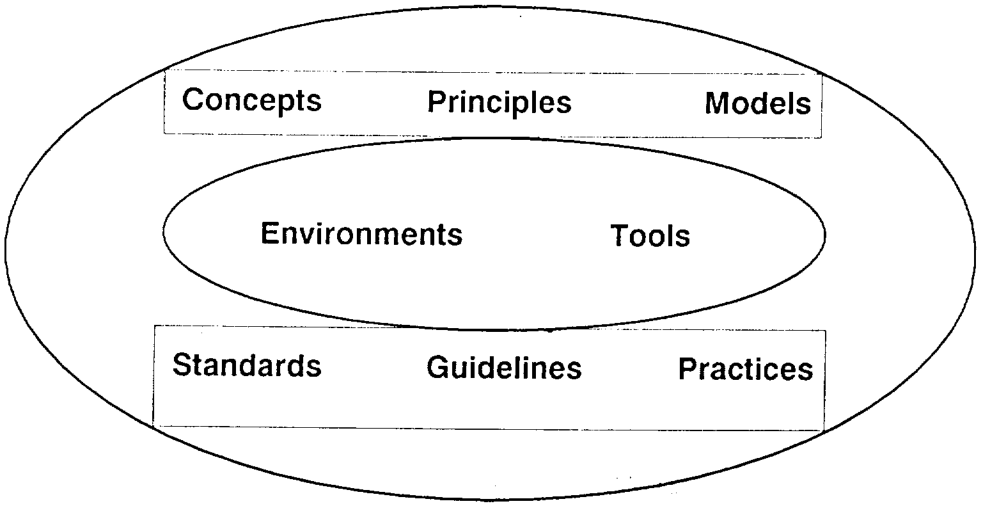
promote the principles of software engineering

to the rest of the software development industry

as a way to increase the scope of its network.

Contents

AdaNET will contain the following information and parts:



Clients

AdaNET will serve clients

from U.S. government and private industry
who are working in software development.

AdaNET will seek new clients

from the software development industry
who have not yet adopted the principles and
procedures of software engineering.

AdaNET will initially concentrate on clients

in manufacturing and administrative (MIS)
computer systems.

Goals

AdaNET will maintain the highest reputation for quality
in its research, its products and its services.

AdaNET will become known as an objective,
authoritative source of new software engineering
information and parts.

AdaNET information and parts will be easy to obtain
and easy to use.

AdaNET will be continually changing
to keep abreast of innovations in the field.

Success

AdaNET will provide its clients

demonstrated benefit
at reasonable cost.

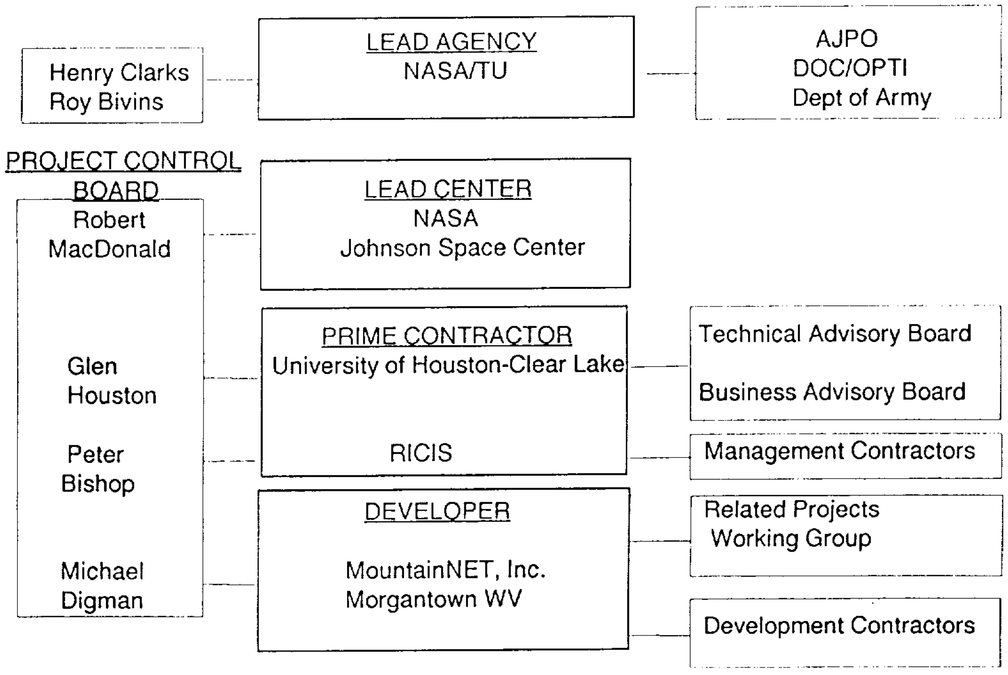
AdaNET will stress

how effectively and
how efficiently
it fulfills its mission.

AdaNET will support itself

through the sale of products and services and
through continuing research contracts.

ORGANIZATION



HISTORY

Milestones

September	1987	Unsolicited proposal submitted
October	1987	Revised proposal accepted
August	1988	Final draft report submitted NASA briefing to sponsors
October	1988	Work begins under new WBS

ADANET SERVICES

Michael Digman
MountainNet, Inc.

MountainNet is a small firm which serves as a distribution center for AdaNET Services. These services include providing host systems and telecommunications for AdaNET, developing and supporting AdaNET Information Services, and developing a dynamic software inventory. AdaNET hosts are a Data General MV8000II and DEC VAX. Telecommunications are provided via the MountainNet private network and the Telenet public access dial network. Initially, the AdaNET Information Services will include software repositories, user communications and forums, software engineering information, bibliographic and library services, and an educational directory. The dynamic software inventory, which will become available in January 1990, will contain software engineering code and parts.

AdaNET Services

- Host Systems & Telecommunications
- AdaNET Information Services
- Dynamic Software Inventory

Host Systems & Telecommunications

AdaNET Hosts:

- Data General MV8000II
- DEC VAX

Telecommunications:

- MountainNet Private Network
- Telenet Public Access Dial Network

AdaNET Information Services (AIS)

Prototype (Version 1.0)

- Software Repositories (ASR, CAMP)
- User Communications & Forums
- Software Engineering Information
- Bibliographic & Library Services
- Education, Training, & Resources Directory

Dynamic Software Inventory (DSI)

Prototype (available 1/90)

- Software Engineering Code & Parts
- Not Only Ada
- Verification & Validation

AdaNET Contacts

(304) 296-1458

Project Director	
Michael Digman	RMD
Administration	
Linda K. Braun	LKB
Library Services	
Rebecca Bills	RBILLS
Market Development	
James Rautner	JWR
Prototype Development	
Donn Philpot	DPHILPOT
Strategic Planning	
Kevin Dyer	DYER
Systems & Telecomm.	
Marcus Hamden	MARCUS
User Services	
Peggy Lacey	LACEY

N90-14807

ADVANCED SOFTWARE DEVELOPMENT WORKSTATION PROJECT*

Daniel Lee
Inference Corp.

The Advanced Software Development Workstation Project, funded by Johnson Space Center, is investigating knowledge-based techniques for software reuse in NASA software development projects. Two prototypes have been demonstrated and a third is now in development. The approach is to build a foundation that provides passive reuse support, add a layer that uses domain-independent programming knowledge, add a layer that supports the acquisition of domain-specific programming knowledge to provide active support, and enhance maintainability and modifiability through an object-oriented approach. The development of new application software would use specification-by-reformulation, based on a cognitive theory of retrieval from very-long-term memory in humans, and using an Ada code library and an object base. Current tasks include enhancements to the knowledge representation of Ada packages and abstract data types, extensions to support Ada package instantiation knowledge acquisition, integration with Ada compilers and relational databases, enhancements to the graphical user interface, and demonstration of the system with a NASA contractor-developed trajectory simulation package. Future work will focus on investigating issues involving scale-up and integration.

* Funded under NASA Contract NAS-9-17766.

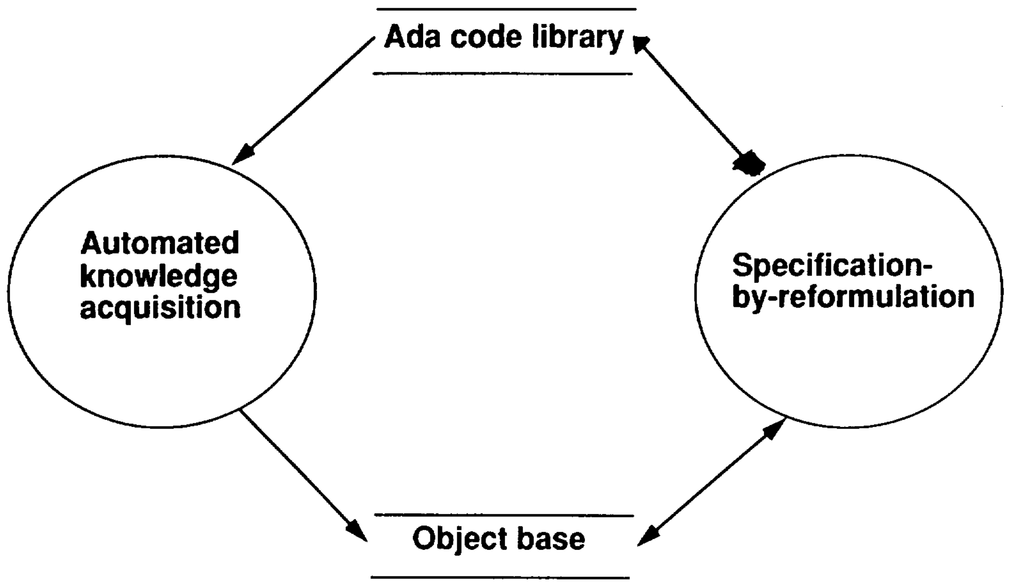
The Software Reuse Process
(from [Prieto-Diaz and Freeman 87])

```
begin
  retrieve matching components from catalog
  if identical match
    then use matching component
    else
      begin
        select best matching component
        modify matching component
      end
end
```

ASDW Technical Approach

- Build a foundation that provides passive reuse support (catalog retrieval and parts composition).
- Add a layer that uses domain-independent programming knowledge to provide interactive support (syntactic constraint checking and code generation).
- Add a layer that supports the acquisition of domain-specific programming knowledge to provide active support (semantic constraint checking).
- Enhance maintainability and modifiability through an object-oriented approach.

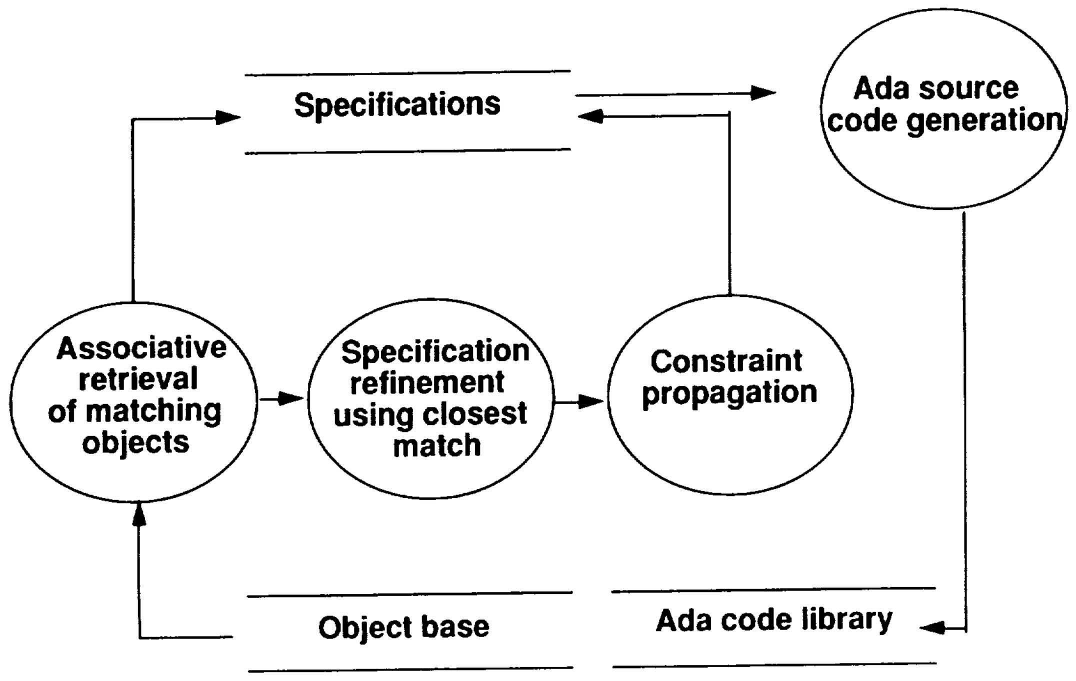
Application Development Using the ASDW


Ada code library
Automated knowledge acquisition
Specification-by-reformulation
Object base

Specification-by-reformulation

- A generic user interface architecture for task support.
- Based on a cognitive theory of retrieval from very-long-term memory in humans.
- A specification-by-reformulation environment consists of:
  - A specification language.
  - A mechanism for providing feedback to the user about the current specification.
  - A mechanism for performing actions on specifications.
- Using specification-by-reformulation for software parts composition:
  - Domain object descriptions and library package specifications form an application-specific specification language.
  - Constraint propagation provides feedback.
  - Specialization and generalization of specifications and code generation are actions.

The Specification-by-reformulation Process


Specifications
Ada source code generation
Associative retrieval of matching objects
Specification refinement using closest match
Constraint propagation
Object base
Ada code library

ASDW Project: Work in Progress

- Current tasks:
  - Enhancements to the knowledge representation of Ada packages and abstract data types.
  - Extensions to support Ada package instantiation knowledge acquisition.
  - Integration with Ada compilers and relational databases.
  - Enhancements to the graphical user interface.
  - Demonstration of the system with a NASA contractor software library (trajectory simulation package).
- Third prototype demonstration: 2/89.

ASDW Project: Future Work

- Goal: Investigate issues involving scale-up and integration.
- Tasks:
  1. Develop and integrate associative retrieval algorithms for use with large software libraries.
  2. Develop and integrate conceptual clustering algorithms for automatic taxonomy generation.
  3. Integrate prototype with NASA software development environment.
  4. Conduct prototype evaluation in conjunction with an ongoing NASA contractor Ada development project.
- Fourth prototype demonstration: 10/89.

WORKSHOP DISCUSSION

Several common themes emerged clearly from the workshop. One prominent theme was the need for systematic methods of reusing software. Central to all approaches for reuse is the reusable software library; the descriptions of the requirements for such libraries showed considerable commonality. In particular, libraries were assumed to contain different types of objects, including requirements, design, and code. Objects were assumed to have attributes and to be categorized according to some classification scheme. In addition, relationships between objects need to be represented and identified for the library user.

A second theme was the need for experimentation to understand how a reuse-based software development process would work. Of particular interest was quantifying the results of the experimentation in some way, i.e., collecting and analyzing data that would aid such understanding. Some unanswered questions were "What data should be collected on these empirical studies?" and "What should be measured?" It was suggested that we need a theory or model of the software reuse process. Many participants thought that we already knew how to produce reusable software, but that we didn't know how to reuse it very well.

A third theme was the need for economic models of reuse that would quantify the cost trade-offs involved in reuse. We wish to know how cost-effective reuse can be and what factors affect this significantly. For example, given parameters such as the cost per part to produce a reusable part, such models would allow calculation of the number of times a part would have to be reused in order to realize a payoff.

The Space Station Freedom Project presentations clearly indicated an immediate concern for what could be done now to achieve a payoff for Space Station that would not require a research investment. The project is looking for ways to reuse software (over a long lifetime and by geographically distributed contractors) which could be implemented from the beginning of software development and yet still be useful in later years.

A list of issues was generated during a lively discussion period, and these have been organized into three categories: technical, managerial, and legal.

TECHNICAL ISSUES

Experimentation

- Need experimentation and empirical studies.
- Need information concerning how to measure reuse, what data should be collected, and how these data should be classified.
- Need economic models and cost/benefit studies related to software reuse.
- Need feedback from users and to capture user experience.

Engineering for reuse

- When is something reusable?
- What functions have high potential for reuse?
- At what level should reuse be supported? specifications, design, code.
- Which parts of the lifecycle would benefit most by reuse? specifications, design, maintenance.
- What standards are needed for reuse?
  terminology, interfaces, taxonomies.

- What process should be used to produce reusable software?
- What process is the best to support reuse of parts?

Reuse libraries

- What should search strategies/query logic be? How effective are they?
- What is an effective knowledge representation?
- How should reusable parts be stored? Should they be stored in one large library or many small libraries?
- What is the optimum size for a library?
  - Many small parts versus few large parts.
  - Restricted domain (scale/size of library).
  - How to limit library parts to useful ones.

Technology transfer

- Should approach to reuse be evolutionary or revolutionary?
- How can reuse be introduced gradually and become more widely used over time?

MANAGERIAL ISSUES

Policies/decisions.

Cost effectiveness of designing for future reuse.

Amount of adaptation of reusable component before rebuilding.

Reward system to encourage reuse (individuals and contractors).

Commitment.

Risk.

Quality Assurance, Certification.

Technology insertion.

LEGAL ISSUES

What software is in the public domain?

How effective is software reuse with proprietary software?

If proprietary software is put into a "public" library for reuse, who is liable if it does not perform as expected?

What are the liability, warranty, and ownership responsibilities?

RECOMMENDATIONS

One of the goals of this workshop was to identify areas for research direction and collaboration among the NASA Centers. The list of technical issues provides potential areas for further work to be pursued. There was consensus that it is premature for NASA to establish policies and procedures for software reuse at this time.

Some specific recommendations were also developed at the close of the workshop: Experimentation and data collection and analysis are needed to evaluate the effectiveness of reuse approaches, and systematic methods for reuse of software components are needed.

First, NASA should encourage experimentation and data collection in ways to reuse software. Such experimentation should probably start on a small scale and move to larger scale experiments as workable reuse processes are found. Furthermore, such experimentation should be cooperative across NASA centers so that projects such as the Space Station Freedom Program (SSFP) will be able to take advantage of the results. The space station Software Support Environment (SSE) could provide an unparalleled opportunity to collect large quantities of data (representing the development and long-term maintenance of millions of lines of code) under reasonably well-controlled conditions (uniform environment and life cycle model). This could be a tremendous opportunity for software engineering in general (not just reuse). It is important that NASA software researchers determine what kinds of data would be important to collect and that the SSE incorporate the ability to collect such data with minimum possible overhead. We need a Memorandum of Understanding (MOU) between the Information Systems Division in OAST (NASA Office of Aeronautics and Space Technology) and the SSE project to facilitate such activities. We also need to do some schedule coordination so we will know when SSE will be able to absorb such requirements and when NASA researchers will be able to generate and validate them. The MOU should address issues such as:

1) SSE will accept and implement requirements to collect automatically certain types of statistical data and make these available to NASA researchers;

2) OAST will conduct a long-term (e.g., 30 years - to match SSFP time scale) research program which will have, as one class of outputs, experimentally validated and calibrated advice and technologies back to SSFP.

Since many of the concepts underlying reuse libraries seem to have been worked out, there should be considerable effort expended in establishing such libraries. This could and should be done in conjunction with reuse experimentation. In particular, attention should be paid to means of populating the libraries efficiently.

As the techniques for reusing software become better known, policies to promote them should be established. These policies could vary from the use of technical standards, for example for establishing uniform design representations to be stored in reuse libraries, to strategies for adopting reuse techniques incrementally, to incentives for reusing software. In particular, the Space Station Freedom Program is already struggling with standards for the use of reuse libraries and the distribution of reusable parts. Support for the project, perhaps by helping to define an acceptable policy across NASA centers, could lead to the collection of data on the efficacy of the policy.

Working groups and other NASA meetings should be held to follow up on this interest in software reuse, such as Software Management and Assurance Program workshops and SSE User's Working Group meetings.

APPENDIX A
PARTICIPANTS

David Badal (713)282-6587	LMSC 1150 Gemini Ave. Houston, TX 77058	NASAMAIL and/or Internet Address
Peter Bishop (713)488-9300	UH Clear Lake 2700 Bay Area Blvd. Houston, TX 77058	PETERBISHOP
James W. Brown (818)354-3614	JPL 301-440 4800 Oak Grove Dr. Pasadena, CA 91109	JWBROWN
Wayne H. Bryant (804)864-1692	NASA LaRC MS 478 Hampton, VA 23665	WBRYANT wayne@uxv.larc.nasa.gov
Edward Comer (407)984-3370	Software Productivity Sol. P. O. Box 361697 Melbourne, FL 32936	ecomer@ajpo.sei.cmu.edu
Michael Digman (304)296-1458	MountainNet P. O. Box 370 Dellslow, WV 26531-0370	
Cammie Donaldson (407)984-3370	Software Productivity Sol. P. O. Box 361697 Melbourne, FL 32936	cdonalds@ajpo.sei.cmu.edu
Kathy Gilroy (407)984-3370	Software Productivity Sol. P. O. Box 361697 Melbourne, FL 32936	kgilroy@ajpo.sei.cmu.edu
Steve Gorman (713)483-5272	NASA/JSC FR3 Houston, TX 77058	SGORMAN
Lionel Hanley (713)488-8806	GHG Corp. 1300 Hercules Suite 111 Houston, TX 77058	LGHANLEY
Scott Herman (703)438-5255	Grumman PSC P. O. Box 4650 Reston, VA 22090	SHERMAN
Robert Holibaugh (412)268-6750	Software Eng. Inst. Carnegie-Mellon Univ. Pittsburgh, PA 15213	rrh@sei.cmu.edu
Tim Kaufman (818)354-4404	JPL 301-440 4800 Oak Grove Dr. Pasadena, CA 91109	
Bob Kirkpatrick (412)268-7634	Software Eng. Inst. Carnegie-Mellon Univ. Pittsburgh, PA 15213	rjk@sei.cmu.edu

Daniel Lee (213)417-7997	Inference Corporation 5300 W. Centry Blvd. L. A., CA 90045	trwrb!smpvax1!sdl @ucbvax.berkley.edu
Travis A. Moebes (713)282-6455	SAIC 1150 Gemini Ave. Houston, TX 77058	
Allen Nikora (818)354-9694	JPL 301-476 4800 Oak Grove Dr. Pasadena, CA 91109	
Dolly Perkins (301)286-6887	NASA/GSPC Code 522 Greenbelt, MD 20771	DPERKINS/GSFCMAIL
Lawrence Prevatte (447)799-3117	MDAC-KSC F932 P. O. Box 21233 Kennedy Space Center Cape Canaveral, FL 32815	LPREVATTE
Charles Shotton (713)282-6444	Planning Research Corporation 1150 Gemini Avenue Houston, TX 77058	
Kathryn Smith (804)864-1699	NASA Langley M/S 478 Langley Research Center Hampton, VA 23665-5225	kas@csab.larc.nasa.gov
Peg Snyder (703)487-7165	Space Station Program Office Reston, VA	PSNYDER
Don Sova (804)453-2154	NASA Headquarters Washington, D.C.	DSOVA
Walt Truszkowski (301)286-8821	NASA/GSFC Code 522.3 Greenbelt, MD 20771	WTRUSZKOWSKI/GSFCMAIL
Lois Valley (407)984-3370	Software Productivity Sol. P. O. Box 361697 Melbourne, FL 32936	
Susan Voigt (804)864-1711	NASA Langley M/S 478 Langley Research Center Hampton, VA 23665-5225	SVOIGT suev@csab.larc.nasa.gov
Carrie Walker (804)864-1705	NASA Langley M/S 478 Langley Research Center Hampton, VA 23665-5225	carrie@csab.larc.nasa.gov

Robert Waterman (301)231-1409	Vitro Corp. 14000 Georgia Ave. Silver Springs, MD 20906	RWATERMAN
David Weiss (703)742-8877	Software Productivity Con. 2214 Rock Hill Road Herndon, VA 22070	weiss@software.org

APPENDIX B
AGENDA

FOLLOWED AT
WORKSHOP ON NASA RESEARCH IN SOFTWARE REUSE
NOVEMBER 17-18, 1988
Melbourne, FL

Thursday, November 17

8:30-9:00	Opening Remarks & Workshop Goals Introduction of Participants	Susan Voigt
9:00-11:50	Software Reuse Activities at SPS	Kathy Gilroy Cammie Donaldson
11:50-1:00	Lunch	
1:00-2:00	SPS Software Reuse continued	Kathy Gilroy Lois Valley
2:00-2:30	Reuse Research Plans at GSFC	Walt Truszkowski
2:30-3:00	Reuse Research Plans at JPL	Jim Brown
3:00-3:20	Reuse Research Plans at JSC	Steve Gorman
3:20-3:30	Reuse Research Plans at LaRC	Susan Voigt Carrie Walker
3:30-4:30	Reuse Projects at the SPC	David Weiss
4:30-5:30	Discussion	Susan Voigt

Friday, November 18

8:30-9:20	Plans for Reuse and Commonality in the Space Station Freedom Program	Peg Snyder
9:20-10:15	SSFP Commonality and Reuse Study	Scott Herman
9:40-11:45	SSE Presentation on Reusability	Dave Badal Chuck Shotton
11:55-1:30	Lunch and Demonstrations at SPS ARCS, Classic-Ada, DesignER, DOCGEN, ALICIA, FastFind	SPS Staff
1:45-2:25	SSE Presentation continued	Travis Moebes
2:25-3:05	Reuse Projects at the SEI	Bob Holibough
3:05-3:30	AdaNET	Peter Bishop Michael Digman
3:30-4:15	JSC/Inference Workstation Project	Daniel Lee
4:15-4:30	Workshop Conclusion	Susan Voigt

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CP-3057	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Software Reuse Issues		5. Report Date December 1989
		6. Performing Organization Code
7. Author(s) Susan J. Voigt and Kathryn A. Smith, Editors		8. Performing Organization Report No. L-16667
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665-5225		10. Work Unit No. 505-65-11-02
		11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546-0001		13. Type of Report and Period Covered Conference Publication
		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

NASA Langley Research Center sponsored a Workshop on NASA Research in Software Reuse on November 17-18, 1988 in Melbourne, Florida, hosted by Software Productivity Solutions, Inc. Participants came from four NASA centers and headquarters, eight NASA contractor companies, and three research institutes. Presentations were made on software reuse research at the four NASA centers; on Eli, the reusable software synthesis system designed and currently under development by SPS; on Space Station Freedom plans for reuse; and on other reuse research projects. This publication summarizes the presentations made and the issues discussed during the workshop.

17. Key Words (Suggested by Authors(s)) Software reuse Reuse libraries	18. Distribution Statement Unclassified–Unlimited Subject Category 61

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 179	22. Price A09

NASA FORM 1626 OCT 86

NASA-Langley, 1989

For sale by the National Technical Information Service, Springfield, Virginia 22161-2171