N92-19431

SEL ADA REUSE ANALYSIS AND REPRESENTATIONS

Rush Kester

Computer Sciences Corporation
GreenTec II
10110 Aerospace Road
Lanham-Seabrook, MD 20706
(301) 794–1714

INTRODUCTION

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC). It was created in 1977 to investigate the effectiveness of software engineering technologies applied to the development of applications software. The SEL has three primary organizational members: NASA/GSFC, Systems Development Branch; University of Maryland, Computer Science Department; and Computer Sciences Corporation, Flight Dynamics Technology Group.

Applications developed in the SEL environment are used primarily to determine and predict the orbit and orientation of earth orbiting satellites. There are many similarities among systems developed for different satellites, and those similarities create a climate in which software reuse enhances efficiency and cost effectiveness in the development process. Consequently, reuse has always been an important SEL priority. Over the last several years, with the introduction of Ada and object-oriented design (OOD) techniques, the SEL has been able to achieve a significant increase in the amount of software reuse. Figure 1 represents graphically the increase in software reuse on recent Ada projects as compared with reuse on FORTRAN projects in a similar time period (Reference 1).

Incorporated into all SEL development is the Process Improvement Paradigm (Reference 2), which includes the following four steps:

1. Understand and characterize the current environment.
2. Try a candidate improvement.
3. Measure any change and provide feedback on experiences.
4. Adopt candidate improvements with favorable results; reject those with unfavorable results.

This Ada reuse study has as a primary objective the first process improvement step.

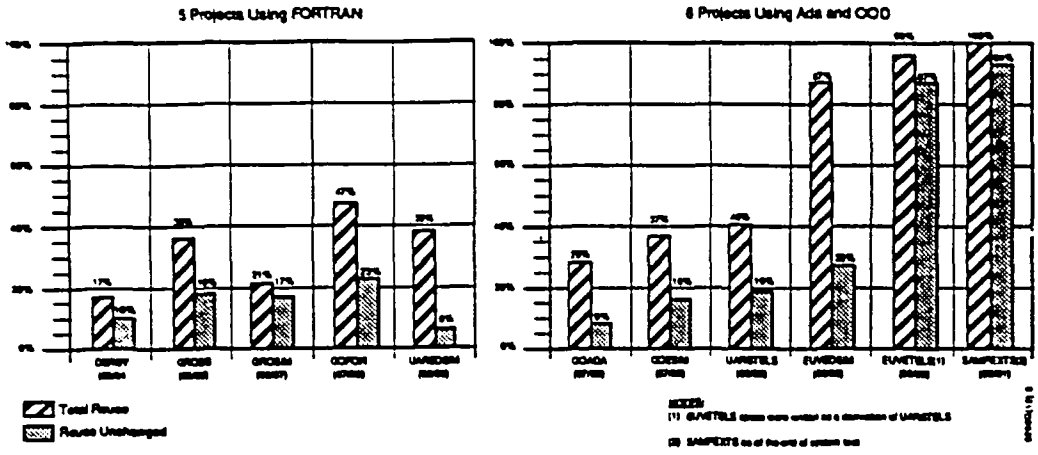


Figure 1. Reuse of Simulator Components

This paper describes the analysis performed and some preliminary findings of a study of software reuse on Ada projects at the SEL.

It describes the representations used to make reuse patterns and trends apparent. The paper focuses on those aspects of the analysis that are applicable to other environments and demonstrates graphically some of the specific patterns of reuse studied by the SEL.

THE STUDY

The study examined software components (i.e., source code files) of three general types: those that were developed on an earlier project within the SEL environment; those that were acquired from an external source, such as a public domain repository or commercial vendor; and those that were adapted from a similar component on the same project. The reuse information in the SEL data base (Reference 3), combined with the source code files and design documents for each executable program, enabled the study team to trace the evolution of software components over a 5-year period that included several generations of similar applications.

As part of the SEL's standard data collection process (Reference 4), the projects involved in the study had recorded each component of the programs comprising each system. Each component was classified according to the percentages of new and reused code it contained.

For those components considered reused, the parent project (or library) was identified and recorded.

Reuse Defined by SEL Study

This SEL Ada study focuses on the reuse of source code files obtained from existing projects or libraries. Although projects in the SEL apply other forms of reuse, such as specification or design products, those other forms were not the subject of this study.

During development, project developers classified (Reference 4) the origin of each component according to the amount of new versus reused code it contained. The highest degree of reuse was for components that were reused verbatim (i.e., unchanged). The next degree was for components that were slightly modified (25 percent or less of their source code changed). Finally, the lowest degree was for components that were extensively modified (more than 25 percent changed).

The following types of components were excluded from the definition of reuse for this study:

- All components developed from scratch, including any such component that may have contained fragments of code from one or more source files or design concepts borrowed from existing components.
- Common components developed for a given project, whether used unchanged multiple times in a given executable program or in multiple programs within the project.

Ada Projects Studied

The study included nine Ada projects. Three projects are dynamics simulators that model the spacecraft's orbit and attitude to evaluate attitude control algorithms. Three projects are telemetry simulators used to generate test data for attitude determination software. One project is an embedded orbit determination system. Another project studied developed or collected components for a reuse library. Finally, one project developed a tool for assembling systems from reusable components. Figure 2 shows a timeline of the nine Ada projects studied.

Representations of Reuse

To identify patterns of reuse over time or within the software's architecture, the SEL study created a series of graphical, textual, and combination graphical/textual representations of component origination and reuse information.

Reuse Across Projects Over Time

One group of four reports graphically represents software reuse by multiple projects over time: PROJECT REUSE SUMMARY, PROJECT REUSE NETWORK, REUSE FROM LINEAGE, and REUSE BY LINEAGE.

The PROJECT REUSE SUMMARY report shows, at the project level, the number of components reused from each project or library. The report is constructed as a matrix

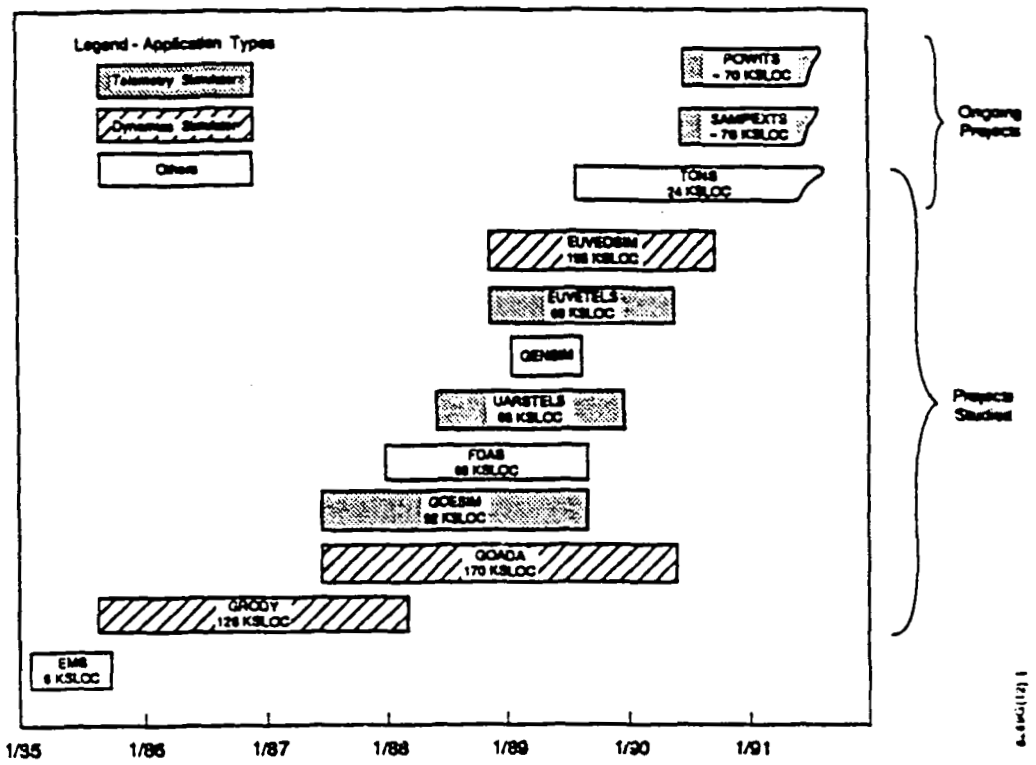


Figure 2. Ada Projects in the Flight Dynamics Division

with projects producing reusable components along the horizontal axis and projects consuming reusable components along the vertical axis. Each cell gives the number of components obtained from a producing project and reused by a consuming project. Two versions of this report could be produced: a detailed report (not shown), which gives subtotals for each of the three degrees of reuse defined for the study, and a total reuse report (shown in Table 1), which gives only the total for all three degrees of reuse.

The PROJECT REUSE NETWORK report (Figure 3) illustrates reuse as a directed graph. The nodes in the graph represent each project, while the fill pattern indicates the type of application. The thickness of each arrow indicates the rough order of magnitude for the number of components reused, and the direction of the arrow indicates the producing and consuming projects. In this report, an increase was evident in the amount of reuse for successive generations of both telemetry simulator and dynamics simulator applications.

A REUSE FROM LINEAGE report focuses on the origin of each reused component. For each instance of a component's use, the project name, the subsystem name, the

Table 1. Number of Reused Components by Project

Reusing Project	Parent Project									
	Beech	GRODY	GOADA	GOESIM	GENSIM	UARSTELS	Other Ada	Other Non-Ada	Total Reused	Total Components
GRODY		5					2	11	18	551
GOADA	2	187	17	1			6	6	219	785
GOESIM	16	4	166	4		13			203	541
FDAS	4	1							5	870
GENSIM	8	2	10					1	21	79
UARSTELS	10	45	97	8		4		6	170	425
EUVETELS	2		1			404	2		409	427
EUVEDSIM			533			34	173		740	861
TONSVAX	2				4		2	127	135	483
TOTALS									1920	[illegible]

component name, and degree of change required are given. The lineage history of each component is shown by indentation, with each level of indentation indicating a prior reuse generation. The following example shows the lineage of the project EUVEDSIM, subsystem SHEM's component EARTH_ATMOSPHERE.

EUVEDSIM SHEM EARTH_ATMOSPHERE — Reused (Unchanged) from

    GOADA SHEM EARTH_ATMOSPHERE — Reused (Extensively modified) from

        GRODY TM ATMOSB — New

Note that the names of the original component, ATMOSB, and subsystem, TM, of GRODY were changed by GOADA.

A REUSE BY LINEAGE report was used to show the project(s) that have reused each component. For each instance of a component's use, the project name, the subsystem name, the component name, and degree of change are given. The family tree of each component is shown by indentation, with each level of indentation indicating a subsequent generation of reuse. Using this report and focusing on the degree of change

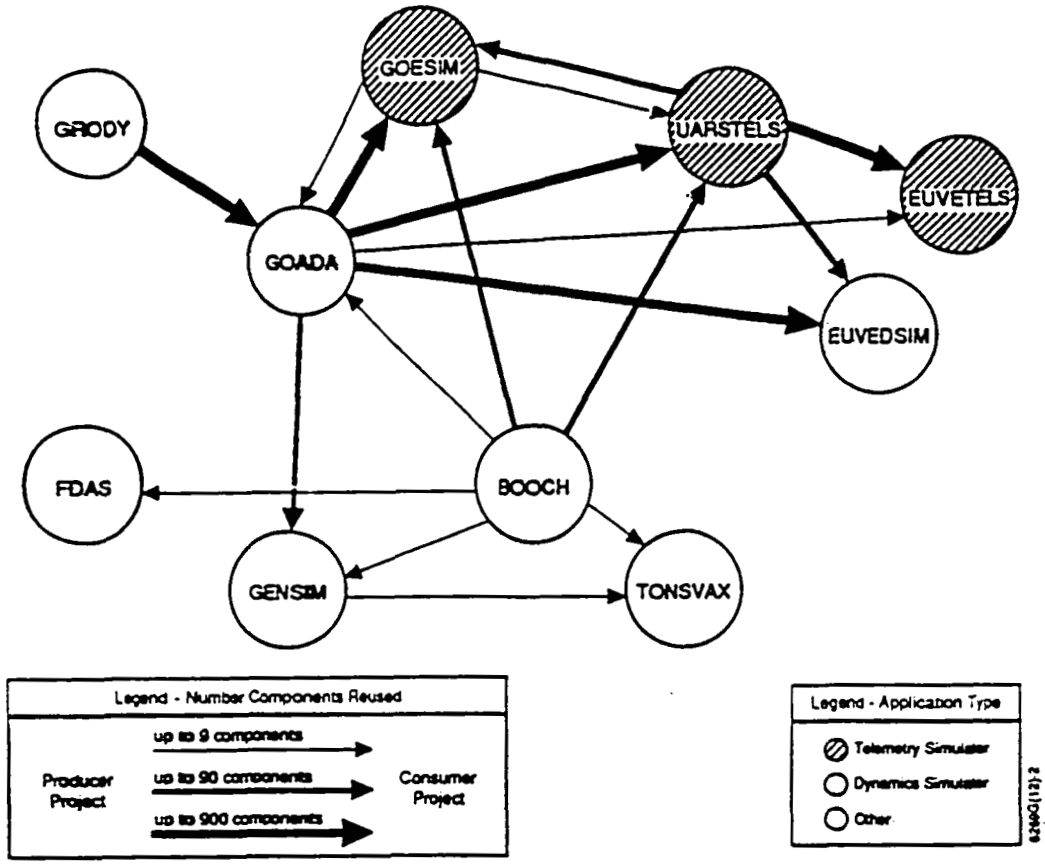


Figure 3. Project Level Reuse Network

required, the study staff noticed two common patterns. The first pattern, a sample of which follows, is for components that implement a general solution.

```
...component_name   (NEW) Reused by
...                   (UNCHANGED)
...                   (UNCHANGED)
...                   (UNCHANGED)
```

The second pattern, shown in the following example, was seen in components that have incompletely or incorrectly implemented a general solution.

```
...component_name   (NEW) Reused by
...                   (...MODIFIED)
...                   (...MODIFIED)
...                   (...MODIFIED)
```

Another component attribute that can be determined by using the REUSE BY LINEAGE report is the component's domain of reuse. Assuming, as is the case in this study, that the researcher knows the application type (or domain) of each project, the domain of reuse can be derived by examining the type of projects reusing a given component.

Both the REUSE FROM LINEAGE and the REUSE BY LINEAGE reports can also be useful for configuration management purposes, by identifying the projects using a given component. If one of the projects suggests an enhancement or correction, the other reusing projects could also be notified.

Reuse Within a Project's Architecture

In addition to examining reuse over time, this study also examined reuse within each project's architecture. To illustrate the findings on a system-wide scale, component reuse was superimposed on graphical/textual representations of each system's static structure, its calling hierarchy, and its compilation order. Architectural representations were derived from the source code and/or design documents.

Each representation of a project's architecture requires between 3 and 10 pages of graphical/textual hard copy, which made it difficult to observe the overall reuse patterns. To overcome this difficulty, the degree of change was color-coded on all representations of reuse within a project's architecture. These representations could then be posted on a wall and the color-coded reuse patterns observed from a distance.

Reuse on a Project's Static Structure

Components were organized according to the Ada package to which they belong. These packages were further organized according to the logical subsystems defined by the original developers. The degree of component reuse was then overlaid on the subsystem and the resulting representation analyzed for patterns. The following is a sample from the REUSE ON STATIC STRUCTURE report for the GOADA project. This sample shows the degree of reuse in one package, EPHEM_FILE_MANAGER, which is part of the SHEM subsystem.

Ada Name	Component Type	Degree of Reuse
EPHEM_FILE_MANAGER .	Package Spec	Unchanged
.	Package Body	Unchanged
.CLOSE	Procedure Body	Unchanged
.COORDINATE_SYSTEM_OF	Function Body	New
.DATA_INTERVAL_OF	Function Body	Unchanged
.END_TIME_OF	Function Body	Unchanged
.INITIALIZE	Function Body	Unchanged
.READ_EPHEM_POINT	Procedure Body	Unchanged
.REAL_TO_AC_TIME	Function Body	Unchanged
.START_TIME_OF	Function Body	Unchanged

Examination of this representation confirmed our intuition that in most cases the granularity of reuse was library units (such as packages or standalone procedures or

functions). The granularity of reuse for telemetry simulators changed dramatically to the entire system architecture in the EUVETELS project, in which structure and almost all components from the UARSTELS project were reused unchanged.

The study examined the hypothesis that a component's interface (i.e., its Ada specification) is reusable with fewer changes than its implementation (i.e., its body). In general, the hypothesis was confirmed by the preceding representation. The notable exception was the second generation of dynamics simulators (i.e., the GOADA project), in which previously large packages were divided into smaller packages. In that case, new package specifications were created for unchanged or slightly modified package bodies.

Another hypothesis examined was that for some groups of components (i.e., a package or a subsystem), there are some parts that must be consistently tailored to each use. Although parts of some packages or subsystems were modified in each successive generation, it was not possible to distinguish mission-tailored parts from those modified for other reasons.

Another hypothesis examined was that some parts of an application's architecture lend themselves to reuse more than do others. The study confirmed the hypothesis by revealing that the highest concentration of consistently reused components was among those implementing basic data structures and mathematical functions or operating on standardized data files.

Reuse on Project's Call Tree

The static analysis of subprogram calls in each component, starting with the main subprogram, is used to create a call tree. The call tree is represented textually as an indented outline, in which each level of indentation denotes a level of nested calls. The order of subprograms in the call tree reflects the order in which each subprogram call appears in the text and, for sequential code, reflects the execution order of the calls. The call analysis demonstrates whether reuse occurs predominantly at the branch or leaf-node level of the call tree.

Reuse on Project's Compilation Order

To study reuse on compilation order, the study group generated a textual representation combined with color-coded graphical elements. Each project's library units were ordered in a roughly bottom-up fashion according to Ada WITH dependency (i.e., the units with fewer WITH dependencies were listed first). Component reuse was then overlaid on the library units. Examination of this representation revealed that the ability in Ada to separate specification from implementation was effective in isolating higher level reused components from extensive changes in lower level components. Also evident using this representation was the ability to reuse Ada generics without change, even in cases where their functionality included entirely new capabilities implemented by generic subprogram parameters. A sample from a REUSE ON PROJECT'S COMPILATION ORDER report is shown in Figure 4.

GENERIC_HARDWARE	(Spec = OLDUC, Body = OLDUC, Subunits OLDUC = 8)
WHEEL	(Spec = OLDUC)
TORQUER	(Spec = OLDUC)
GYRO_ANALOG_RATES	(Spec = OLDUC)
TONS_DOPPLER	(Spec = NEW)
GENERIC_MODEL_GYRO_ANGLES	(Spec = OLDUC, Body = OLDUC)
GYRO_ANGLES	(Spec = OLDUC)
SCHEDULER	(Spec = OLDUC, Body = SLMOD, Subunits SLMOD = 2, OLDUC = 4)
SIMULATOR_DATABASE	(Spec = SLMOD, Body = SLMOD, Subunits SLMOD = 2)
INITIAL_PARAMETERS_REPORT	(Spec = OLDUC, Body = OLDUC, Subunits SLMOD = 1)
MINOR_FRAME	(Spec = SLMOD)
TELEMETRY_RECORD	(Spec = OLDUC)
TELEMETRY_SIMULATOR	(Body = OLDUC)

Legend: OLDUC is unchanged
SLMOD is slightly modified
NEW is developed from scratch

Figure 4. Reuse on Compilation Order

CONCLUSIONS

Overall, the study revealed that the pattern of reuse has evolved from initial reuse of utility components into reuse of generalized application architectures. Utility components were both domain-independent utilities, such as queues and stacks, and domain-specific utilities, such as those that implement spacecraft orbit and attitude mathematical functions and physics or astronomical models. The level of reuse was significantly increased with the development of a generalized telemetry simulator architecture.

The use of Ada generics significantly increased the level of verbatim reuse, which is due to the ability, using Ada generics, to parameterize the aspects of design that are configurable during reuse. A key factor in implementing generalized architectures was the ability to use generic subprogram parameters to tailor parts of the algorithm embedded within the architecture.

The use of object-oriented design (in which objects model real-world entities) significantly improved the modularity for reuse. Encapsulating into packages the data and operations associated with common real-world entities creates natural building blocks for reuse.

REFERENCES

1. Software Engineering Laboratory, SEL-89-007, "Experiences in the SEL—Applying Software Measurement," *Proceedings of the Fourteenth Annual Software Engineering Workshop*, F. E. McGarry (GSFC), S. R. Waligora (CSC), and T. P. McDermott (CSC), November 1989
2. Software Engineering Laboratory, SEL-89-007, "The Experience Factory: Packaging Software Experience," *Proceedings of the Fourteenth Annual Software Engineering Workshop*, V. R. Basili (University of Maryland), November 1989

3. Software Engineering Laboratory, SEL-89-101, *"Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 1)*, M. So, G. Heller, S. Steinberg, K. Pumphrey, and D. Speigel, February 1980

4. Software Engineering Laboratory, SEL-87-008, *Data Collection Procedures for the Rehosted SEL Database*, G. Heller, October 1987

VIEWGRAPH MATERIALS

FOR THE

R. KESTER PRESENTATION

Ada Reuse Analysis and Representation at the Software Engineering Laboratory (SEL)

Rush Kester
Rhea White
Robert Kazden

Agenda

- Background
- Representations of reuse
- Preliminary observations

Definition of Reuse

- Ada source code obtained from existing projects or libraries
- Each source file (a.k.a. component) classified according to percent of lines reused without change
- Definition does not include other forms of reuse

Potential Benefits of Reuse

With high-level reuse, delivered systems can be

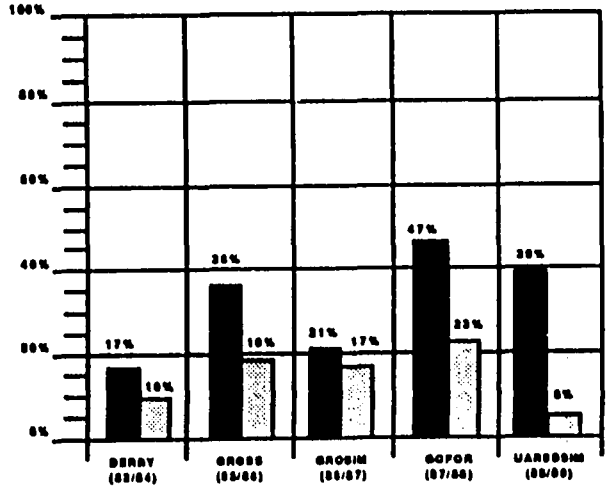
- **Delivered sooner and at lower cost**
- **Incrementally improved**
- **More reliable**

Flight Dynamics Environment

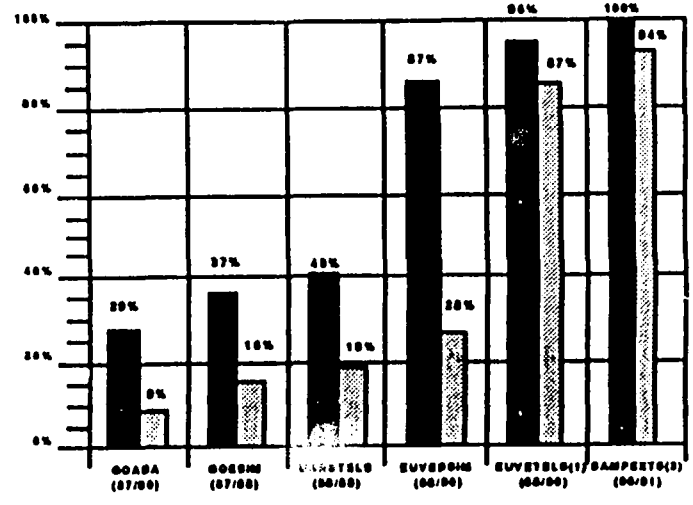
- Develop similar systems for different satellites
- Knowledge carried between missions
- Reuse an important part of culture
- Economic benefits directly related to amount of code reused without change
- Introduction of Ada and OOD significantly increased reuse of code

Reuse of Simulator Components

5 Projects Using FORTRAN



5 Projects Using Ada and OOD



TOTAL REUSE

REUSE UNCHANGED

Computer Sciences Corporation
System Sciences Division

61810(12)

Steps in Process Improvement

1. Understand and characterize current environment
2. Try candidate improvement
3. Measure change – feedback experience
4. Adopt candidates with favorable results; Reject candidates with unfavorable results

Goals of Current Phase of Study

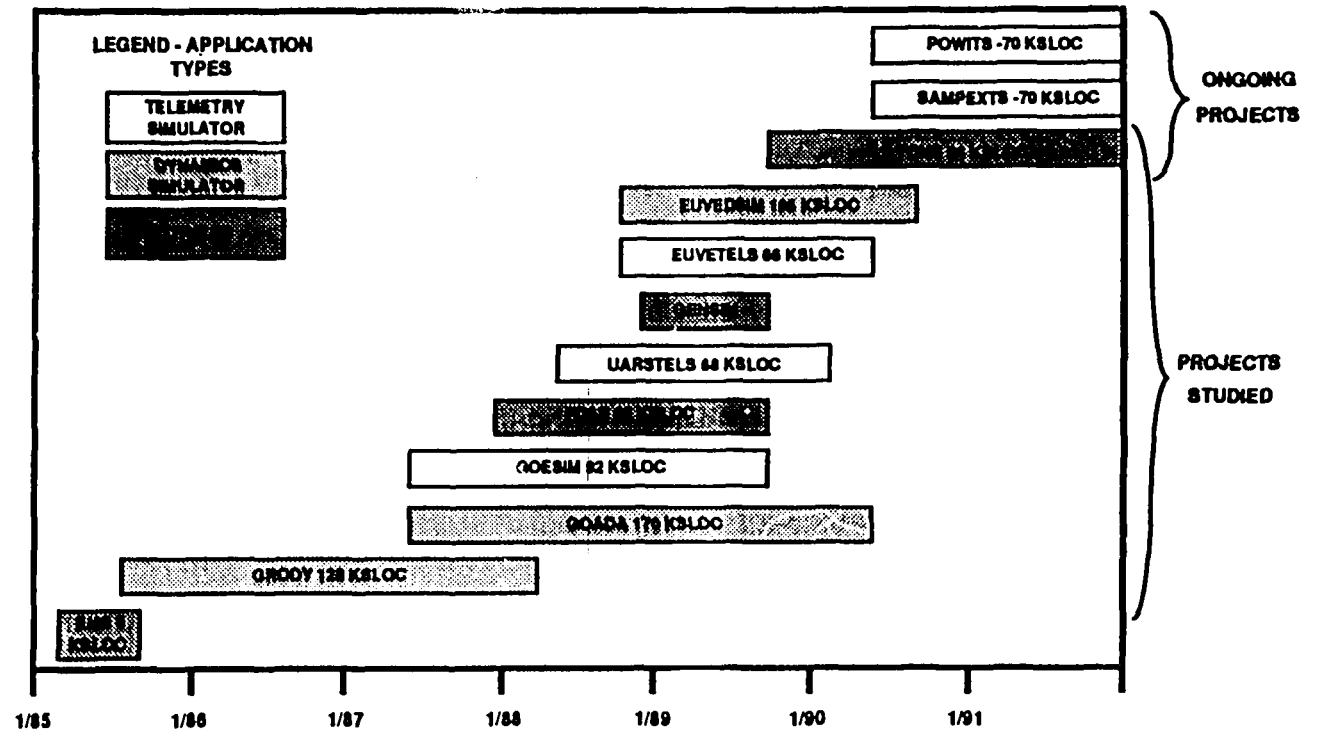
- Understand and characterize reuse
  - Determine patterns and trends of reuse
  - Determine characteristics distinguishing reused from non-reused components
- Identify candidates for reuse library
- Identify domain of component's reusability within the environment
- Address some CM issues related to reuse

Questions Addressed by Study

- Does separation of interface specification and implementation affect degree of change required?
- Do Ada generics improve the level of reuse without change?
- Does the extent of intercomponent dependencies affect reuse?
- What is the granularity of reuse?
- Where in software architecture does reuse occur?
- Do patterns of component evolution suggest guidelines for more effective reuse?

R. Kester
CSC
Page 6 of 30

Ada Projects in the Flight Dynamics Division



Computer Sciences Corporation
System Sciences Division

6151G(12)

R. Kester
CSC

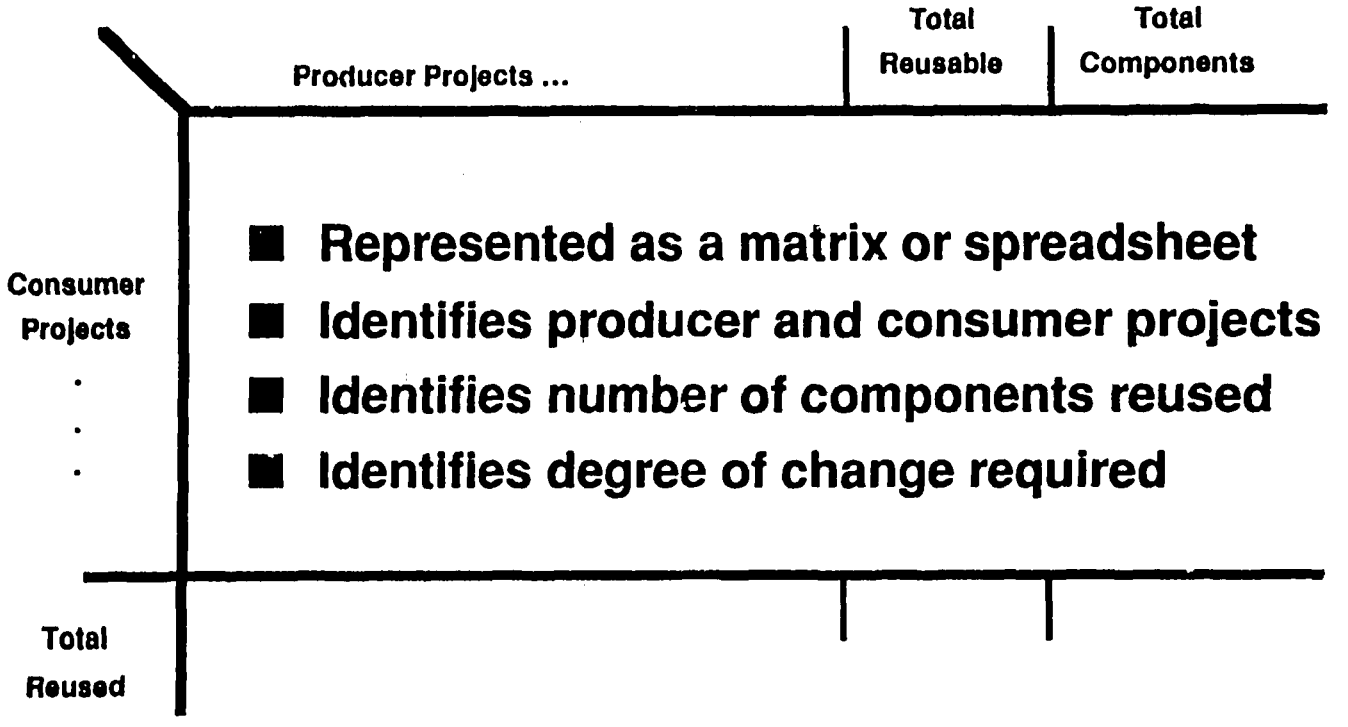
Data Used

- Size of Study
  - 9 Ada projects, over 5 years
  - Over 1900 reused components
- Input Used
  - SEL Component Origination Forms
  - Source code files
  - Representations of software design

Representations of Reuse

1. Multi-Project Reuse Summary
2. Project Level Reuse Network
3. Component Lineage Reports
4. Reuse on Software Static Structure
5. Reuse on Software Call Tree
6. Reuse on Ada Compilation Order

1. Multi-Project Reuse Summary Report

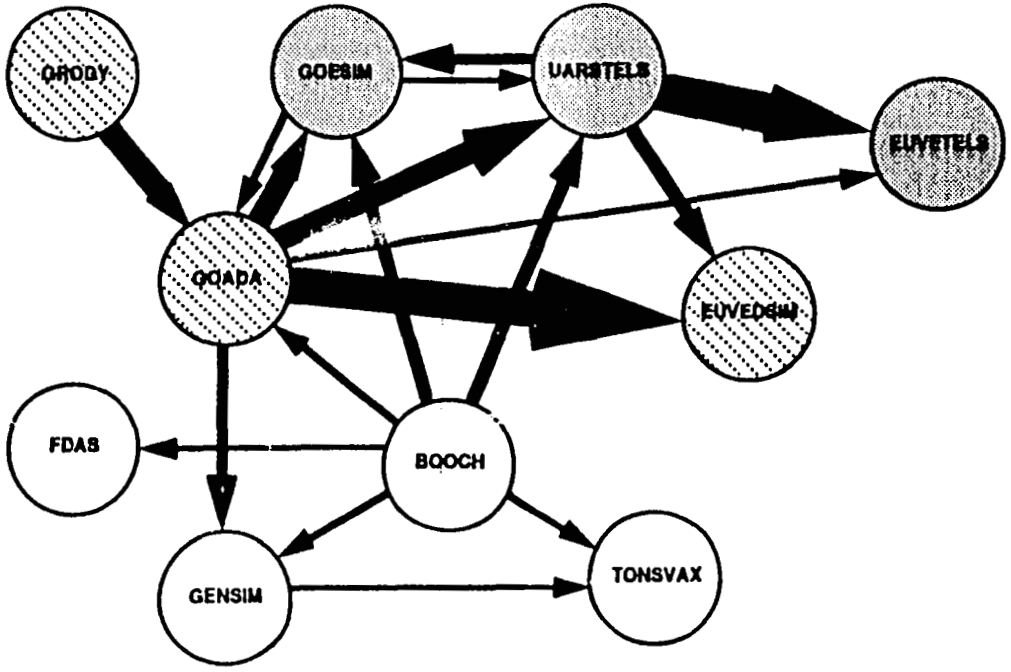


- Represented as a matrix or spreadsheet
- Identifies producer and consumer projects
- Identifies number of components reused
- Identifies degree of change required

Computer Sciences Corporation
System Sciences Division

6151G(12)

2. Project Level Reuse Network



Computer Sciences Corporation
System Sciences Division

61510(12)

R. Kester
CSC

3. Component Lineage Reports

- Represents reuse over time
  - From originating project forward
  - From reusing project back to origin
- Identifies parent - child relationship
- Identifies components:
  - Implementing general solution
  - Generalized incorrectly or incompletely
  - Domain of applicability
- Useful for CM purposes

4. Reuse on Software Static Structure

- Represents reuse at project level
- Reflects developer's logical view
- Makes visible:
  - Granularity of reuse
  - Project dependent parts

5. Reuse on Software Call Tree

- Represents reuse at project level
- Reflects actual calling hierarchy
- Makes visible:
  - Level of functionality reused
  - Location of reuse within architecture

6. Reuse on Ada Compilation Order

- Represents reuse at project level
- Reflects coupling between "Library Units"
- Makes visible:
  - Scope of change required to reuse
  - Location of reuse within architecture

Reuse Patterns and Trends Observed

- Initially application independent components reused, now majority reflect organization's problem domain
- Ada generics significantly increased the level of verbatim reuse
- OOD (where objects model real world entities) significantly improved modularity

Work Remaining

- Develop guidance for improving verbatim reuse
- Investigate rationale behind characteristics that distinguish reusable components
- Confirm hypothesis -- Achieved highly reusable solution for Telemetry Simulator applications