Technical Report
CMU/SEI-94-TR-22
ESC-TR-94-022

Software Process Improvement in the NASA Software Engineering Laboratory

Frank McGarry
Rose Pajerski
NASA/Goddard Space Flight Center

Gerald Page
Sharon Waligora
Computer Sciences Corporation

Victor Basili
Marvin Zelkowitz
University of Maryland

December 1994

Technical Report
CMU/SEI-94-TR-22
ESC-TR-94-022
December 1994

Software Process Improvement in the NASA Software Engineering Laboratory

Frank McGarry
Rose Pajerski
Gerald Page
Sharon Waligora

Technology Transition Initiatives

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

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

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Suite C201, Pittsburgh, PA 15212. Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8274 or toll-free in the U.S. — 1-800 225-3842).

Table of Contents

List of Figures

Foreword

This report describes the work of the first winner of the IEEE Computer Society Software Process Achievement Award. This award was jointly established by the Software Engineering Institute (SEI) and the IEEE Computer Society to recognize outstanding achievements in software process improvement. It is given annually, if suitable nominations are received at the SEI on or before November 1 of any year. To obtain further information about the award, contact the award coordinator at the SEI.

For the 1994 award, a total of 11 nominations were received and evaluated by a review committee consisting of Vic Basili, Barry Boehm, Manny Lehman, Bill Riddle, and myself. Because of his intimate involvement with the winning organization, Vic Basili excused himself from the committee's decisions concerning this organization.

As a result of reviewing the nominations and visiting several laboratories, the committee selected two finalists and the winner. The finalists are the process improvement staff of the Motorola Cellular Infrastructure Group and the software engineering process group of the Raytheon Equipment Division. The winner is the Software Engineering Laboratory (SEL) which is jointly operated by the NASA Goddard Space Flight Center, the Computer Sciences Corporation, and the University of Maryland. As a condition of the award, one or more representatives of the winning organization writes an SEI technical report on the achievement. This is that report.

Many organizations have found that the lack of adequate data on the costs and benefits of software process improvement is a significant deterrent to their progress. This award thus emphasizes both quantitative measures of process improvements as well as their significance and potential impact. While no single improvement approach will be appropriate for every organization and while process improvement methods will evolve, the broad availability of such explicit improvement information should be of broad and general value.

The granting of this award does not imply endorsement of any one improvement approach by the IEEE Computer Society or the SEI. The award committee does, however, endorse the excellence of the work described in this technical report. We also feel it is particularly appropriate that the SEL win this first achievement award. This is both because of the technical significance of the SEL's work and because of its long history of leadership in software process research and improvement. The SEL was formed a full 10 years before the SEI was established and its distinguished record of contribution has had a profound impact on the work of software process professionals throughout the world.

Watts S. Humphrey
Chairman, Award Committee

Preface

Since its inception, the SEL has conducted experiments on approximately 120 Flight Dynamics Division (FDD) production software projects, in which numerous software process changes have been applied, measured, and analyzed. As a result of these studies, appropriate processes have been adopted and tailored within the environment, which has guided the SEL to significantly improve the software generated. Through experimentation and sustained study of software process and its resultant product, the SEL has been able to identify refinements to its software process and to improve product characteristics based on FDD goals and experience.

The continual experimentation with software process has yielded an extensive set of empirical studies that has guided the evolution of standards, policies, management practices, technologies, and training within the organization. The impacts of these process changes are evident in the resulting characteristics of FDD products. Over the period 1987 through 1993, the error rate of completed software has dropped by 75 percent; the cost of software has dropped by 50 percent; and the cycle time to produce equivalent software products has decreased by 40 percent. Additionally, the SEL has produced over 200 reports that describe the overall software process improvement approach and experiences from the experimentation process.

This report describes the background and structure of the SEL organization, the SEL process improvement approach, and the experimentation and data collection process. Results of some sample SEL studies are included. It includes a discussion of the overall implication of trends observed over 17 years of process improvement efforts and looks at the return on investment based on a comparison of the total investment in process improvement with the measurable improvements seen in the organization's software product.

As this report is a summary of many of the activities and experiences of the SEL which have been reported in more detail elsewhere, some of the material has been taken directly from other SEL reports. These sources are listed in the Reference section of this document. Individual copies of SEL documents can be obtained by writing to

Software Engineering Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

Software Process Improvement in the NASA Software Engineering Laboratory

Abstract: The Software Engineering Laboratory (SEL) was established in 1976 for the purpose of studying and measuring software processes with the intent of identifying improvements that could be applied to the production of ground support software within the Flight Dynamics Division (FDD) at the National Aeronautics and Space Administration (NASA)/Goddard Space Flight Center (GSFC). The SEL has three member organizations: NASA/GSFC, the University of Maryland, and Computer Sciences Corporation (CSC). The concept of process improvement within the SEL focuses on the continual understanding of both process and product as well as goal-driven experimentation and analysis of process change within a production environment.

1 Background

1.1 SEL History

The Software Engineering Laboratory (SEL) was created in 1976 at NASA/Goddard Space Flight Center (GSFC) for the purpose of understanding and improving the overall software process and products that were being created within the Flight Dynamics Division (FDD). A partnership was formed between NASA/GSFC, the University of Maryland, and Computer Sciences Corporation (CSC) with each of the organizations playing a key role: NASA/GSFC as the user and manager of all of the relevant software systems; the University of Maryland as the focus of advanced concepts in software process and experimentation; and CSC as the major contractor responsible for building and maintaining the software used to support the NASA missions. The original plan of the SEL was to apply evolving software technologies in the production environment during development and to measure the impact of these technologies on the products being created. In this way, the most beneficial approaches could be identified through empirical studies and then captured once improvements were identified. The plan was to measure in detail both the process as well as the end product.

At the time the SEL was established, significant advances were being made in software development (e.g., structured analysis techniques, automated tools, disciplined management approaches, quality assurance approaches). However, very little empirical evidence or guidance existed for selecting and applying promising techniques and processes. In fact, little evidence was available regarding which approaches were of any value in software production. Additionally, there was very limited evidence available to qualify or quantify the existing software process and associated products, or to aid in understanding the impact of specific methods. Thus, the SEL staff developed a means by which the software process could be

understood, measured, qualified, and measurably improved. Their efforts focused on the primary goal of building a clear understanding of the local software business. This involved building models, relations, and empirical evidence of all the characteristics of the ongoing software process and resultant product and continually expanding that understanding through experimentation and process refinement within a specific software production environment.

1.2 SEL Process Improvement Strategy

As originally conceived, the SEL planned to apply selected techniques and measure their impact on cost and reliability in order to produce empirical evidence that would provide rationale for the evolving standards and policies within the organization. As studies were performed, it became evident that the attributes of the development organization were an increasingly significant driver for the overall definition of process change. These attributes include the types of software being developed, goals of the organization, development constraints, environment characteristics, and organizational structure. This early and important finding provoked an integral refinement of the SEL approach to process change. The most important step in the process improvement program is to develop a baseline understanding of the local software process, products, and goals. The concept of internally driven, experience-based process improvement became the cornerstone of the SEL's “bottom-up” process improvement program.

Incorporating the key concept of change guided by development project experiences, the SEL defined a standard paradigm to illustrate its concept of software process/product improvement. This paradigm is a three-phase model (Figure 1) which includes the following steps:

1. *Understanding:* Improve insight into the software process and its products by characterizing the production environment, including types of software developed, problems defined, process characteristics, and product characteristics.
2. *Assessing:* Measure the impact of available technologies and process change on the products generated. Determine which technologies are beneficial and appropriate to the particular environment and, more importantly, how the technologies (or processes) must be refined to best match the process with the environment.
3. *Packaging:* After identifying process improvements, package the technology for application in the production organization. This includes the development and enhancement of standards, training, and development policies.

In the SEL process improvement paradigm, these steps are addressed sequentially, and iteratively, for as long as process and product improvement remains a goal within the organization.

The SEL approach to continuous improvement is to apply potentially beneficial techniques to the development of production software and to measure the process and product in enough detail to determine the value of the applied technology within the specific domain of application. Measures of concern (such as cost, reliability, and cycle time) are identified as the organization determines its major short- and long-term objectives for its software product. Once

these objectives are known, the SEL staff designs an experiment(s), defining the particular data to be captured and the questions to be addressed in each experimental project. This concept has been formalized in the goal-question-metric paradigm developed by Basili [Basili 84].

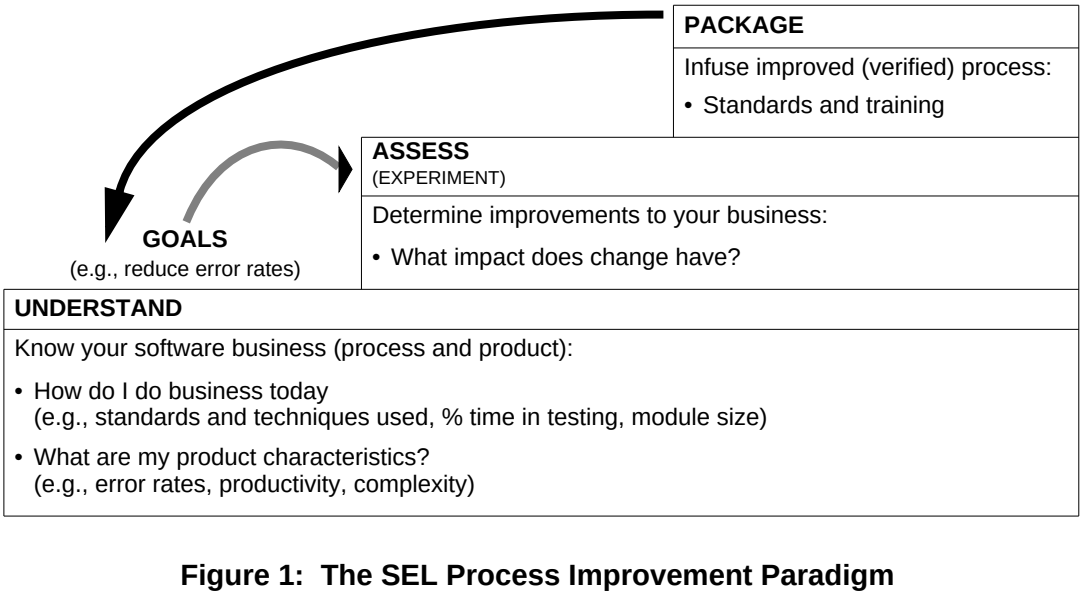


Figure 1: The SEL Process Improvement Paradigm

All SEL experiments have been conducted in the production environment of the FDD at NASA/GSFC, which consists of projects that are classified as mid-sized software systems. The systems range in size from 10 thousand source lines of code (KSLOC) to over 1.5 million SLOC. The original SEL production environment had approximately 75 developers generating software to support a single aspect of the flight dynamics problem. Over the years, the SEL operation has grown to include more extensive software responsibilities and, consequently, a larger production staff of approximately 300 developers and analysts. The SEL has been in continuous operation since 1976, and it will continue to operate as long as process and product improvement remain a priority within its software domain.

2 The SEL Organization

The SEL comprises three partner organizations: the Software Engineering Branch at NASA/GSFC, the Institute for Advanced Computer Studies and Department of Computer Science at the University of Maryland, and the Software Engineering Operation at CSC. The total organization consists of approximately 300 persons. These personnel are divided into 3 functional components, not necessarily across organizational lines. The 3 functional areas are

- Software development/maintenance
- Process/product analysis
- Database support

The three components (developers, analysts, and database support) are separate, yet intimately related to each other. Each has its own goals, process models, and plans, but they share an overall mission of providing software that is continually improving in quality and cost effectiveness. The responsibilities, organizational makeup, and goals of the SEL components are discussed in the sections that follow. Figure 2 provides a graphic overview of their function and size, and Figure 3 depicts the difference in focus among the three groups.

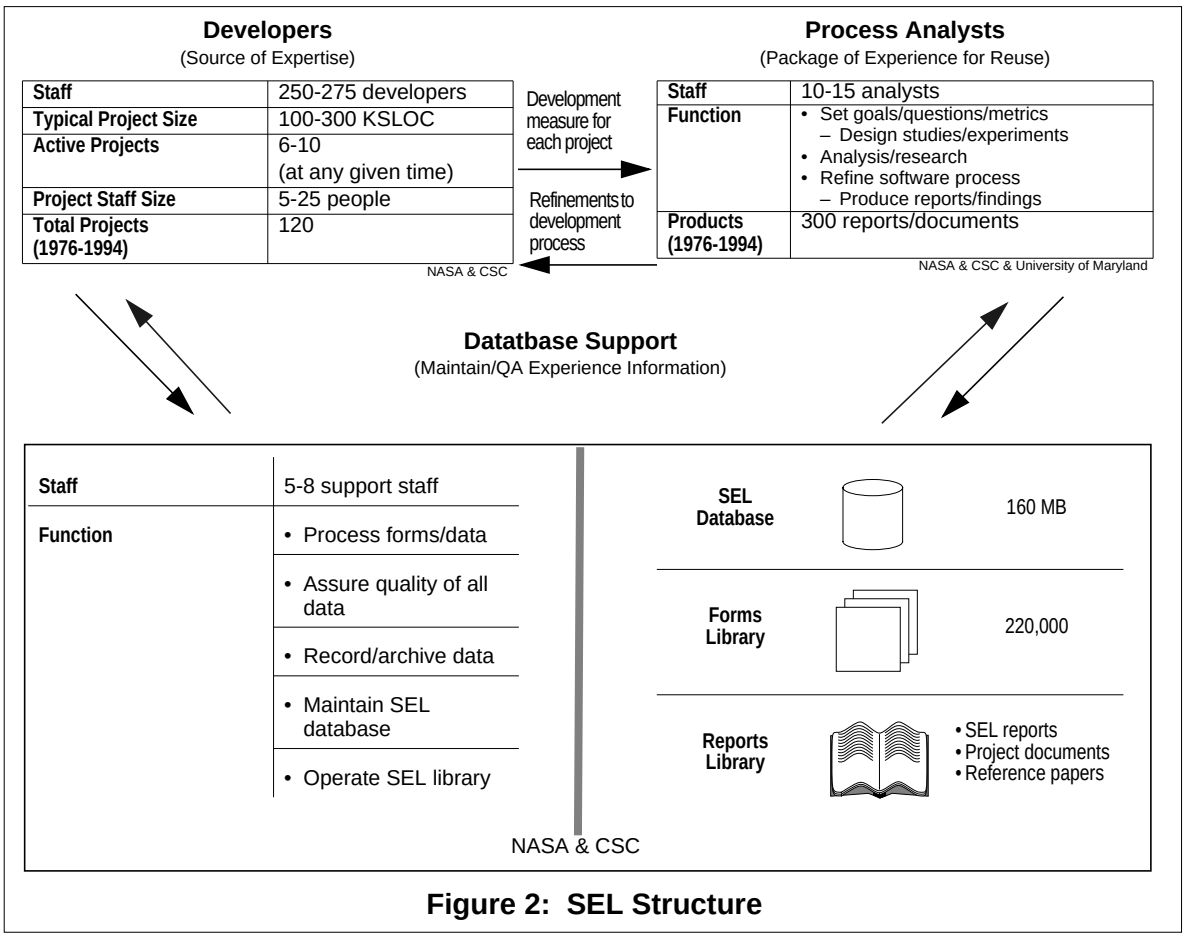


Figure 2: SEL Structure

	Developers	Analysts	Database Support Staff
Focus and Scope	• Specific software project	• Domain (multiple projects)	• Domain (multiple projects)
Goals	• Produce and maintain software • Satisfy user requirements	• Analyze development and maintenance experience to define improvement process • Support developers	• Archive, maintain, and distribute development and maintenance experience
Approach	• Use the most effective software engineering techniques, as provided by the analysts	• Assess the impact of specific technologies • Produce models, standards, and training materials	• Maintain a library of experiences, models, and standards
Measure of Success	• Validate and verify software products	• Packaging and reuse of empirical software experience • Improved software products	• Efficient processes for information retrieval (data models, reports)

Figure 3: Focus of SEL Organizational Components

2.1 Software Development/Maintenance

The FDD development organization, comprising approximately 250–275 professional software developers, is responsible for development and maintenance of one segment of the ground support software used by GSFC. The majority of the software developers are CSC employees under contract to NASA/GSFC; approximately 35 of the developers are employees of NASA/GSFC. SEL staff at the University of Maryland do not participate directly in the development or maintenance of flight dynamics software.

For a typical project, FDD developers are provided a set of functional requirements for a mission, from which they design, code, test, and document the software. The systems developed are primarily FORTRAN, non-real time, non-embedded, ground-based applications, and there are usually 4 or 5 projects in development at any one time. After the newly developed mission support software is tested and accepted, another team from this same organization takes over maintenance of the operational system. Approximately 50 percent of the development staff is allocated to software maintenance.

The primary task of the development organization is to produce quality software on-time and within budget. They rely on another element of the SEL to carry out the analysis and packag-

ing of the process improvement studies. The development organization is not expected to produce standards, policies, or training; nor are the developers expected to analyze data. The success of the development organization is measured by their ability to deliver a quality software product that meets the needs of the user.

2.2 Process/Product Analysis

The second major function within the SEL is analysis and process improvement. This effort is supported by personnel from all 3 member organizations: approximately 4 full-time people from NASA/GSFC; 5-10 individuals, each spending approximately 20 percent of their time, from the University of Maryland; and approximately 5-8 full-time people at CSC. This team defines studies to be conducted, analyzes process and products generated by the developers, and packages its findings in the form of updated standards, revised training programs, and new models specific to this development environment. All of the SEL analysts are experienced software engineers, many of whom have a number of years of experience in flight dynamics software development and/or maintenance.

The analysts use information such as development environment profiles, process characteristics, resource usage, defect classes, and statistics to produce models of products and processes, evaluations, and refined development information. Their products include cost and reliability models, process models, domain-specific architectures and components, policies, and tools.

The goal of the analysts is to synthesize and package experiences in a form useful to the development group. Their success is measured by their ability to provide in a timely way products, processes, and information that can assist the developers in meeting their goals.

2.3 Database Support

The third function within the SEL is the data processing and archiving of the project's experiences in the SEL's measurement database. This is supported by approximately three full-time people at NASA/GSFC and approximately five full-time people at CSC. The database support staff collect the data that have been defined and requested by the analysts; assure the quality of those data; organize and maintain the SEL database; and archive the reports, papers, and documents that make up the SEL library (see Figure 2).

The group includes both professional software engineers, who define and maintain the database, and data technicians, who enter the data, generate reports, and assure the quality of the information that is submitted to the SEL library. The goal of the database support organization is to manage the SEL measurement data and analysis products efficiently. Their success is measured by the efficient collection, storage, and retrieval of information, conducted in a way that doesn't burden the overall organization with unnecessary activities and waiting periods.

3 The SEL Process Improvement Concept

The SEL process improvement concept has matured over more than a decade, with the most significant changes to it being driven by experience at attempts to infuse process change and improvement within a production organization. The SEL improvement concept has occasionally been called an “experience factory” [Basili 92] and has also occasionally been termed a “bottom-up” software process improvement approach [McGarry 94]. The organization focuses on continually using experiences, lessons, and data from production software projects to ensure that subsequent development efforts benefit, in terms of improved software products and processes, from the experience of earlier projects. The underlying principle of this concept is the *reuse* of software experiences to improve subsequent software tasks. This reuse of experience is the driving element for change and improvement in the software process.

3.1 Bottom-Up Improvement

Although the term “*process* improvement” is the term most commonly used to characterize the efforts of an organization to improve its software business, the SEL philosophy asserts that the actual goal of the organization is to improve the software *product*. The *process* improvement concept stems from an assumption that an improved process will result in an improved product. However, if a changed process has no positive impact on the product generated, then there is no justification for making change. A knowledge of the products, goals, characteristics, and local attributes of a software organization is needed to guide to the evolutionary change to process that focuses on the desired change to the product as defined by the goals of the organization.

Two approaches to software process improvement have been developed and applied in the industry. The top-down approach (which is based on the assumption that improved process yields improved product) compares an organization’s existing process with a generally accepted high-quality standard process. Process improvement is then defined as the changes made to eliminate the differences between the existing process and the standard set of practices. This approach assumes that after change is made to the process, the generated products will be improved, or at least there will be less *risk* in the generation of new software. The most widely accepted and applied top-down model is the capability maturity model (CMM) [Paulk 93], developed by the Software Engineering Institute (SEI). For the SEL, the CMM provides an excellent model for assessing process and for selecting potential process changes that mesh with local goals and needs.

The SEL approach (sometimes called “bottom-up”) assumes that changes must be driven by local goals, characteristics, and product attributes. Changes are defined by a local domain instead of by a universal set of accepted practices. In this approach, software process change is driven by the goals of the particular development organization as well as by the experiences derived from that local organization. For example, an organization whose primary goal is to

shorten “time-to-ship” may take a significantly different approach to process change than would an organization whose primary goal is to produce defect-free software.

The top-down approach is based on the assumption that there are generalized, universal practices that are required and effective for all software development, and that without these practices, an organization’s process is deficient. This paradigm has been accepted in many software organizations that have applied generalized standards, generalized training, and even generalized methods defined by an external organization (external to the developers) to all their software. This concept does not take into account the performance issues, problems, and unique software characteristics of the local organization. The implicit assumption is that even if an organization’s goals are being met and exceeded, if that organization does not use the commonly accepted practices, it has a higher risk of generating poor-quality products than an organization that adheres to the defined processes. The goals and characteristics of the local organization are not the driving elements of change.

The underlying principle of the SEL approach is that “not all software is the same.” Its basic assumption is that each development organization is unique in some (or many) aspects. Because of that, each organization must first completely understand its local software business and must identify its goals before selecting changes meant to improve its software process. If, based on that understanding, change seems called for, then each change introduced is guided by “experience”—not by a generalized set of practices.

Neither the top-down approach nor the bottom-up approach can be effective if used in isolation. The top-down approach must take into consideration product changes, while the bottom-up approach must use some model for selecting process changes aimed at improving product characteristics. Each concept plays an important role in the goal of improving the software business.

The CMM is designed as a framework that can help organizations better understand their software process and provide guidance toward reducing risk in software production. It provides an excellent procedure for identifying potentially beneficial additions to the organization’s software business practices. The SEL capitalizes on this paradigm to guide efforts at characterizing the software development process and to identify improvement areas. As a complement to the CMM (which provides specific approaches to assessing goals, products, and product attributes), the SEL model provides the tools for a complete and effective improvement program.

Both approaches have similar difficulties capturing the image of the “local” software business, in defining exactly the scope or size of the local organization. Some judgment must be applied as to the components and boundaries of this single entity. SEL experience has shown that a smaller defined organization allows for a more detailed process definition and more focused refinements to the process.

3.2 Measurement

The SEL approach uses a detailed understanding of local process, products, characteristics, and goals to develop insight. This insight forms the foundation of a measurable, effective change program driven by local needs. Because of this dependence on understanding the software within the subject environment, measurement is an inherent and vital component of the SEL approach: measurement of process and product from the start, measurement of the effect of process change on the product, and measurement of product improvement against the goals of the organization. The CMM provides guidance in building an understanding of software process within the development organization, but the SEL paradigm extends this concept to include product characteristics such as productivity, error rates, size attributes, and design characteristics.

In the SEL approach, measurement is not viewed as a process element that is added as an organization matures, but rather as a vital element present from the start of any software improvement program. An organization must use measurement to generate the baseline understanding of process and product that will form the basis of the improvement program.

The CMM includes the “software process assessment” tool, which is effective for generating baseline process attributes. The SEL’s bottom-up approach adds to those measures measurement of specific product characteristics, so that change can be effectively guided and observed.

The SEL concept is driven by the principle that each domain or development organization must develop and tailor specific processes that are optimal for its own usage. Certainly, some processes and technologies are effective across a broad spectrum of domains (possibly even universal), but before a development organization settles on a particular process it must take the critical steps of understanding its software business and determining its goals. From there, change can be introduced in a structured fashion and its impact measured against the organizational goals.

3.3 Reuse of Experience

Historically, a significant shortcoming in software development organizations has been their failure to capitalize on experience gained from similar completed projects. Most of the insight gained has been passively obtained instead of being aggressively pursued. Software developers and managers generally do not have the time or resources to focus on building corporate knowledge or planning organizational process improvements. They have projects to run and software to deliver. Thus, reuse of experience and collective learning must become a corporate concern like a business portfolio or company assets. Reuse of experience and collective learning must be supported by an organizational infrastructure dedicated to developing, updating, and supplying upon request synthesized experiences and competencies. This organizational infrastructure emphasizes achieving continuous sustained improvement over identifying possible technology breakthroughs.

The SEL represents this type of organizational element. It is focused solely on reuse of experience and software process improvement with the goal of improving the end product. Because these activities rely so significantly on actual software development experiences, the developers, analysts, and database support staff organizations, while separate, are intimately related to each other. Developers are involved in process improvement activities only to the extent that they provide the data on which all process change is based. Process/product analysts and database support personnel are dedicated to their process improvement responsibilities and are in no way involved in the production of software product. Additionally, the SEL research/database support teams have management and technical directors separate from the development projects. This ensures continuity and objectivity in process improvement activities and the availability of resources for building, maintaining, and sustaining the process improvement program.

4 SEL Experimentation and Analysis

Each production project in the FDD is considered an opportunity for the SEL to expand its knowledge base of process understanding and improvement. There are typically 4 or 5 projects under development at any one time, and an additional 15 to 20 projects in the maintenance phase. All of the projects in the FDD environment are considered experiments, and the SEL has completed over 120 project studies over the years. For each of these projects, detailed measurements were provided toward the end goal of analyzing the impact that any change to software process had on the resultant software product.

When research in the production environment is being planned, the following activities occur: the SEL analysis team defines a set of goals that reflects current goals in process/product improvement and writes an experiment plan in which required data are identified and experimental processes are outlined; a SEL representative is assigned to the project/experiment; and technology/process training needs are assessed. SEL software development/maintenance project personnel then provide the requested information (defined in the experiment plan) to the SEL database support staff who add it to the database for access by the analysts conducting the experiment. These SEL activities are described in the sections that follow.

4.1 Defining Experiments

SEL analysts identify software process modifications that they hypothesize are likely to improve the resultant product, and then design an experiment to test the hypothesis. As experiments are being defined, the analysts consult the development team to determine if proposed changes (such as applying a particular technique) could be studied on a project without undue risk. Even if risk is significant, a team may be willing to try the new process provided a contingency plan is developed to assure that a disaster can be avoided. It is important that the development team be factored into decisions on the proposed changes and that their full support is obtained.

Once a project is identified and a modified process is selected, an experiment plan is written describing the goals, measures, team structure, and experimental approach. A sample SEL experiment plan is included in Appendix A. If the study is very small (e.g., collect inspection data to measure the cost of software inspections), a formal experiment plan may not be written.

The basic project/experiment information is then provided to the SEL database support group so that project names, subsystem names, personnel participating, and forms expected can be logged, and the database can be readied for data entry.

Once an experiment is defined and the study objectives have been agreed upon with the developers, a representative from the analysts is assigned to work directly with the development team for the duration of the project. This representative keeps the development team informed of experimental progress, provides information on the particular process

changes being applied, and answers any questions the development team may have with regard to SEL activities. The SEL representative does not manage or direct the development project in any way. The SEL representative attends reviews and development status meetings and looks at measurement data collected. At the conclusion of the project, the SEL representative also writes a section for inclusion in the project's development history report which discusses the experimental goals and results.

For most projects, the experiment being conducted does not have a significant impact on the development procedures and typically does not involve major changes to the technologies being applied. If there is a more significant change (e.g., using Ada, applying Cleanroom technique, or using inspections with a team unfamiliar with the technology), the analysts arrange for training for the development team. For example, when the SEL studied Cleanroom technique on one project, approximately 40 hours of training in the technique were provided to the first development team using it in this environment [Green 90].

4.2 Collecting Measures

In support of the SEL experiments, technical and management staff responsible for software development and maintenance provide the requested measurement data. Although the types of data requested may vary from project to project to satisfy the requirements of particular experiments, the core set of information is invariant. Basic data are collected from every project, including effort, defects, changes, project estimates, project dynamics (e.g., staffing levels), and product characteristics. These data are provided on data collection forms. Figures 4 and 5 are samples of the forms used to report effort data and defect/change data. Details of the core measures used, as well as the measurement program in general, can be found in the *Software Measurement Guidebook* [Bassman 94]. The full set of data collection forms and procedures can be found in the *Data Collection Procedures for the Software Engineering Laboratory Database* [Heller 92].

Personnel Resources Form

Name: _____ **Date (Friday):** _____

Project:

SECTION A: Total Hours Spent on Project for the Week: ☐

SECTION B: Hours by Activity

(Total of hours in Section B should equal total hours in Section A.)

Activity	Activity Definitions	Hours
Predesign	Understanding the concepts of the system. Any work prior to the actual design (such as requirements analysis).	
Create Design	Development of the system, subsystem, or components design. Includes development of PDL, design diagrams, etc.	
Read/Review Design	Hours spent reading or reviewing design. Includes design meetings, formal and informal reviews, or walkthroughs.	
Write Code	Actually coding system components. Includes both desk and terminal code development.	
Read/Review Code	Code reading for any purpose other than isolation of errors.	
Test Code Units	Testing individual components of the system. Includes writing test drivers.	
Debugging	Hours spent finding a known error in the system and developing a solution. Includes generation and execution of tests associated with finding the error.	
Integration Test	Writing and executing tests that integrate system components, including system tests.	
Acceptance Test	Running/supporting acceptance testing.	
Other	Other hours spent on the project not covered above. Includes management, meetings, training hours, notebook, system description, user's guides, etc.	

SECTION C: Effort on Specific Activities (need not add to Section A)

(Some hours may be counted in more than one area; view each activity separately.)

Rework: Estimate of total hours spent that were caused by unplanned changes or errors. Includes effort caused by unplanned changes to specifications, erroneous or changed design, errors or unplanned changes to code, changes to documents. (This includes all hours spent debugging.) ☐

Enhancing/Refining/Optimizing: Estimate of total hours spent improving the efficiency or clarity of design, code, or documentation. (These are not caused by required changes or errors in the system.) ☐

Documenting: Hours spent on any documentation on the system. (This includes development of design documents, prologs, in-line commentary, test plans, system descriptions, user's guides, or any other system documentation.) ☐

Reuse: Hours spent in an effort to reuse components of the system. (This includes effort in looking at other system(s) design, code, or documentation. Count total hours in searching, applying, and testing.) ☐

For Librarian's Use Only	
Number:	
Date:	
Entered by:	
Checked by:	

Figure 4: Effort Data Collection Form

CHANGE REPORT FORM

Name: _____ **Approved by:** _____

Project: _____ **Date:** _____

Section A — Identification

Describe the change (what, why, how):

Effect: What components are changed?

Prefix	Name	Version

(Attach list if more space is needed.)

Effort (What additional components were examined in determining what change was needed?):

Location of developer's source files _____

Need for change determined on: | month | day | year |

Change completed (incorporated into system): | month | day | year |

Effort in person time to isolate the change (or error): | 1 hr/less | 1 hr/1 day | 1/3 days | >3 days |

Effort in person time to implement the change (or correction): | 1 hr/less | 1 hr/1 day | 1/3 days | >3 days |

Check here if change involves Ada components. (If so, complete questions on reverse side.) ☐

Section B — All Changes

Type of change (check one):

- ☐ Error correction
- ☐ Planned enhancement
- ☐ Implementation of requirements change
- ☐ Improvement of clarity, maintainability, or documentation
- ☐ Other:
- ☐ Improvement of user services
- ☐ Insertion/deletion of debug code
- ☐ Optimization of time/space/accuracy
- ☐ Adaptation to environment change

Effects of Change

Y	N	
		Was the change or correction to one and only one component? (Must match *Effect* in Section A.)
		Commission error (i.e., something incorrect was included)

Section C — For Error Corrections On

Source of Error (check one)

- ☐ Requirements
- ☐ Functional specifications
- ☐ Design
- ☐ Code
- ☐ Previous change

Class of Error (check most applicable*)

- ☐ Initialization
- ☐ Logic/control structure (e.g., flow of control incorrect)
- ☐ Interface (internal) (module-to-module communication)
- ☐ Interface (external) (module-to-external communication)
- ☐ Data value or structure (e.g., wrong variable used)
- ☐ Computational (e.g., error in math expression)

*If two are equally applicable, check the one higher on the list.

Characteristics

Y	N	
		Omission error (i.e., something was left out)
		Commission error (i.e., something incorrect was included)
		Transcription (clerical) error

For Librarian's Use Only

Number:	
Date:	
Entered by:	
Checked by:	

November 1991

Figure 5: Defect/Change Data Collection Form

As the developers/maintainers complete the forms, they submit them to the database support personnel who assure the quality of the information by checking the forms and data for consistency and completeness. When data are missing (e.g., if an expected form is not submitted), the developer is informed of the discrepancy and is expected to provide or correct the data. Database support staff then enter the data in a central database and perform a second quality assurance step by checking for data entry errors by comparing the database information against the original paper forms.

In addition to the forms that are completed by the developers and managers, several tools are used to gather automatically information such as source code characteristics (e.g., size, amount of reuse, complexity, module characteristics) or changes and growth of source code during development. Database support personnel execute the tools to gather these additional measures, which are then entered in the SEL database.

Additionally, subjective measures are recorded on the development process. These data are obtained by talking with project managers and by observing development activities. Data such as problem complexity, adherence to standards, team experience, stability, and maturity of support environment are captured at the termination of each project. (See [Bassman 94] for details on these measures.)

Figure 6 depicts the life-cycle phases during which the core SEL measures are collected. Each project provides these data and may provide additional measures required for the specific experiment in which it is participating.

4.3 Analyzing Data

The analysts use these data together with information such as trend data, previous lessons learned, and subjective input from developers and managers, to analyze the impact of a specific software process and to build models, relations, and rules for the corporate memory. As specific processes are studied (such as inspections, Cleanroom), the analysts, joined by willing participants from the development organization, complete analysis reports on the study and may even prepare a paper or report for publication in the open literature. Development team participation is strictly voluntary in this step, as the analysts are ultimately responsible for producing the report.

As the project information becomes available, the analysts use it not only to assess particular processes, but also to build models of the process and product so that the experiences of each development effort can be captured and applied to other projects where appropriate. Data are used to build predictive models representing cost, reliability, code growth, test characteristics, changes, and other characteristics. The analysts also look at trends and processes applied to determine whether or not any insight can be gained from data describing particular methodologies used during development or maintenance.

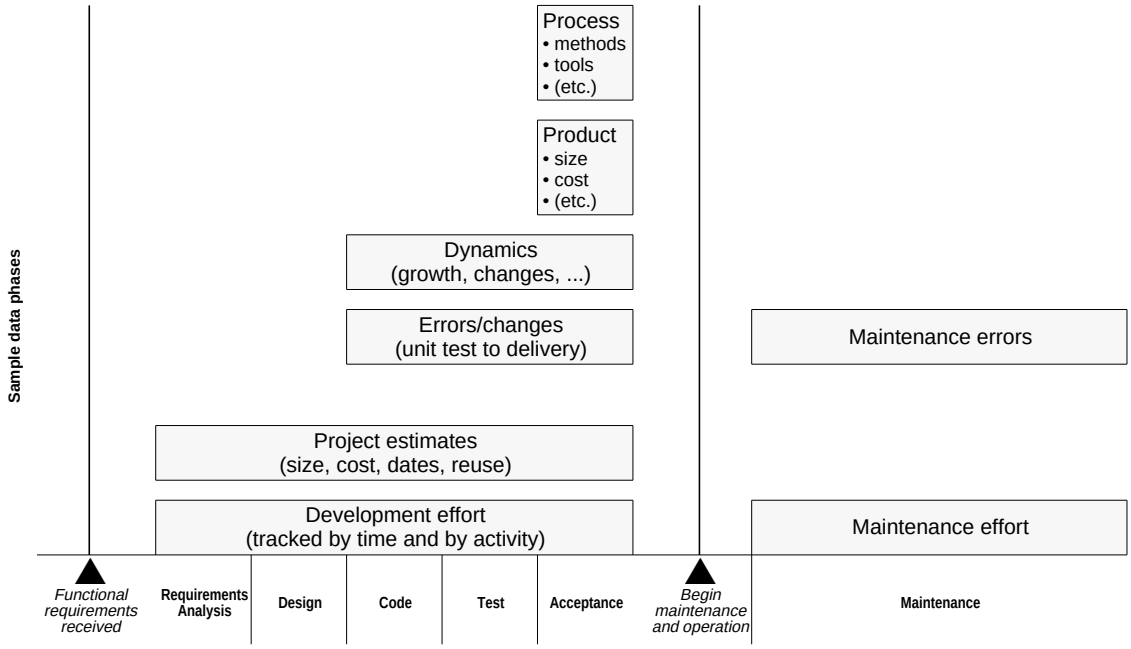


Figure 6: SEL Core Measures

One of the most important facts that the SEL has learned from its experience with analysis of software data is that the actual measurement data represent only one small element of experimental software engineering. Too often, data can be misinterpreted, used out of context, or weighted too heavily even when the quality of the information may be suspect. Having learned from its extensive data analysis experience over the years, the SEL now follows these key rules:

- *Software measures will be flawed, inconsistent, and incomplete; the analysis must take this into account. Do not place unfounded confidence in raw measurement data.*

  Even with the extensive quality-assurance process and the rigor of the software measurement collection process in the SEL, the uncertainty of the data is still quite high. An analyst must consider subjective measures, qualitative analysis, definition of the context, and an explanation of the goals. If one merely executes a high number of correlation analysis studies on a high number of parameters, chances are that some (possibly very questionable) statistic will appear. Extreme caution must be applied when using software measurement data, especially when the analyst is not intimately familiar with the environment, context, and goals of the studies.

- *Measurement activity mst not be the dominant element of software process improvement; analysis is the goal.*

  When the SEL began to study software process, the overhead of the data collection process dominated the total expenditures for experimental activities. As the SEL matured, it found that the successful analysis of experiments should consume approximately three times the amount of effort that data collection activities require. This ratio was attained through a gradual cutback in data collection to where the only information requested (beyond the core measures) was that which could be clearly defined as relevant to the goals of a particular experiment.

- *Measurement information must be treated within a particular context; an analyst cannot compare data where the context is inconsistent or unknown.*

  Each set of measurement data that is archived in the SEL database represents a specific project, with unique characteristics and unique experimental goals. These goals may have significantly influenced the process used, the management approach, and even the general characteristics of the project itself. Without knowledge of the context in which the data were generated and the overall project goals as well as process goals, significant misinterpretations of the data can result.

4.4 Improving Process

Measurement activities represent a relatively small element of the overall process improvement task. Results of analysis of experimental data must be judiciously applied toward optimizing the software development and maintenance process. The experimental software engineering results are captured both in studies as well as in refined processes available to the production personnel. The SEL packages its analysis results in the form of updated standards, policies, training, and tools. This packaging facilitates the adoption of revisions to the standard processes on ongoing and future software projects.

The SEL conducts three general types of analysis, all of which are active continually in the environment. They include

- Pilot studies of specific techniques and technologies on a project or set of projects [e.g., Cleanroom impact on design, impact of object-oriented design (OOD) on code reuse, impact of inspection on coding errors].
- Studies of completed projects for development and refinement of local process and product models (e.g., cost models, error characteristics, reuse models).
- Trend analysis of completed projects to track the impact of specific process changes on the environment as a whole (e.g., tailored Cleanroom, OOD, software standards).

All of the analyses are dependent on the project measures and all require a thorough understanding of context, environment, goals, problem complexity, and project characteristics to be able to derive results that can be fed into the overall process improvement program.

A study of a specific process or technique is usually termed a “pilot study.” Although these studies often occur in the university environment, they are also conducted on production projects where some risk can be tolerated. These projects are testing new and unfamiliar techniques to determine their value in the production environment and to determine whether more extensive studies would be beneficial. On pilot projects, the analyst typically analyzes each phase of the project in detail and reports back to the development team the intermediate results as the project progresses toward completion. In general, the SEL conducts no more than two pilot studies at any one time because the amount of analysis and reporting is so extensive. These studies normally yield multiple reports and papers that look at every aspect of the impact of the new technology, make recommendations for tailoring, and project the value of the enhanced process in an expanded application.

The second class of study involves multiple projects, where the goal is to expand and update the understanding of process and product attributes. Cost models are enhanced, error attributes are studied, and relations between process and product characteristics are analyzed for classes of projects. These studies normally do not use data from projects under development, but focus on completed projects. This type of analysis requires not only the archived measurement data, but also a detailed knowledge of each project’s context (including goals, processes used, problem complexity, size, and other product attributes). Trends in software quality, productivity, as well as profiles of the software product are produced so that specific needs and potential process enhancements can be identified.

Trend analysis also looks at multiple completed projects. The goal of these studies is to determine the appropriate application of evolving technology and methods within the environment as a whole, or at least for a specific class of projects. After pilot projects have been completed and appropriate tailoring or enhancement of process changes have been made, additional projects apply the tailored process. The additional application of the methods may involve only a single element of the originally defined process. For instance, although the Cleanroom methodology includes specific techniques for design, testing, management, implementation, and the inspection process, it may turn out that only the testing and implementation techniques are appropriate for further application. Once it is determined which process changes are appropriate for a broader class of projects (or possibly the entire development environment), these elements of the process are incorporated into the software standards and policies. Additionally, the training program may be updated to reflect the refined process. (See the discussion of packaging in Chapter 5 for a detailed description of the SEL training program.)

5 SEL Experiences: Understanding, Assessing, and Packaging

The SEL paradigm has been applied on approximately 120 production projects in the FDD. Each project has provided detailed measurement data for the purpose of providing more insight into the software process, so that the impact of various software technologies could be empirically assessed. Projects have ranged in size from 10 KSLOC to 1.5 million SLOC, with the majority falling in the 100–250 KSLOC range. Appendix B lists the general characteristics of the projects that have been studied in the SEL to date. All of the information extracted from these development and maintenance projects is stored in the SEL database and used by the analysts who study the projects and produce reports, updated standards, policies, and training materials.

During the *understanding* phase of the SEL paradigm, the goal is to produce a baseline of development practices and product attributes against which change can be measured as process modifications are applied. Additionally, the understanding process generates the models and relations used to plan and manage the development and maintenance tasks. The goal of the *assessing* or experimental phase is to determine the impact of specific process changes on the overall goals of the organization. In the *packaging* phase of the paradigm, those practices that have proven measurably beneficial are incorporated into the organization's standards, policies, and training programs.

5.1 Understanding

Probably the most critical element of the SEL's process improvement program is the understanding step—where the only goal is to gain insight into the local software business. This first step cannot provide the justification for claiming that one process is better than another, but instead yields a baseline of the characteristics of the software, including both process and products, upon which change and meaningful comparison can be based.

Although the initial plan was to begin experimenting with various techniques, the SEL soon learned that without a firm, well-understood baseline of both process and product characteristics, valid experimentation was impossible. In order to build this understanding, information gathered from the first 5–10 projects was primarily used to generate models, relations, and characteristics of the environment. These models and their understanding proved to be a significant asset to the management, planning, and decision making needed for effective software development.

The understanding process, begun with those first 5–10 projects, continues today on all projects. The various models are continually updated as the process is better understood, and as new technologies and methods change the way the SEL views software development. Fig-

ure 7 lists 11 projects active between 1985 and 1990 that were included in the early SEL baseline.

Project	Start Date	End Date
GROSIM	8/85	8/87
COBSIM	1/86	5/87
GRODY	9/85	7/88
COBEAGSS	6/86	7/88
GROAGSS	8/85	4/89
GOESIM	9/87	7/89
GOFOR	6/87	9/89
GOESAGSS	8/87	11/89
UARSTELS	2/88	12/89
GOADA	6/87	4/90
UARSAGSS	11/87	9/90

Figure 7: SEL Baseline (1985-1990)

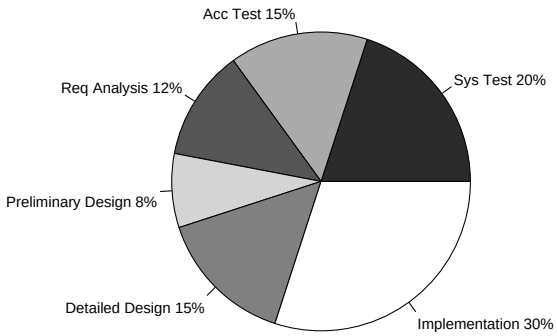
By examining the effort data of these projects, the SEL built its baseline of software cost expenditures by phase and by activity. This is some of the most basic, yet often overlooked, information for software environments. By looking at a series of projects, a simple model of effort distribution can be built to depict the cost of design, code, test, and other activities. Such data are accumulated weekly from all developers, managers, and technical support using a data collection form. The form captures effort expended on software design, testing, coding, and the amount of time spent on code reading vs. code writing. (See Figure 4 for a sample effort data collection form.)

Figure 8 illustrates distribution for the effort data based on the projects in this baseline. These data represent 11 projects over 5 years, consuming a total of approximately 65 staff-years of effort. The data show that approximately 1/4 of the cost of producing the software is spent on activities other than designing, coding, or testing. This “other” activity includes meetings, travel, reviews, training, etc. The SEL has found that the value for “other” activities has remained almost constant for the entire time the SEL has been closely monitoring projects; in fact, it has increased slightly over time instead of decreasing as SEL staff first expected that it would. This time represents an important component for project budgets, one that is often overlooked by managers who lack a thorough understanding of their baseline process. One

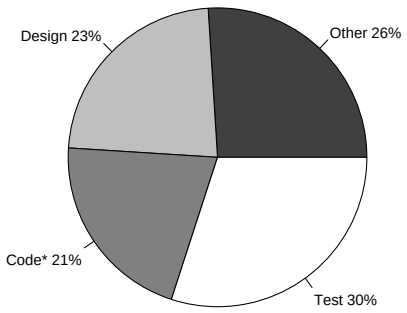
surprising observation has been that the basic characteristics of this environment do not radically change from year to year even with continuous modifications being made to the underlying processes. The profile of the software environment changes very slowly. In Figure 8, the data are represented in two ways:

- One representation is effort by phase, where the total hours reported each week are attributed to the phase that the project is currently executing: i.e., designing from start through review and acceptance of design, coding from start through beginning of system testing, and testing from the start through system delivery. These data require only that the phase dates be known and that the total hours worked each week be reported by the development staff.

- The second representation is effort by activity, where weekly information is broken down to the particular activity that the programmers were performing during that week. For example, they may report design hours even though the project was well into the coding phase. This modeling of the data provides a more accurate view of project interactions, as compared to the model that relies on (somewhat arbitrary) phase dates often set before project initiation.



Effort Distribution by Life-Cycle Phase



Effort Distribution by Activity

* code writing: 85%
code reading:15%

Figure 8: Effort Distribution by Phase and Activity

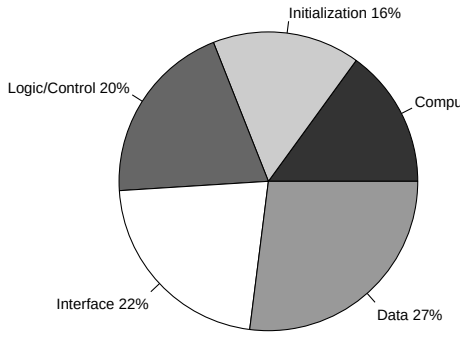
Along with cost and schedule, reliability and correctness of the resulting code are considered attributes of interest to management. These attributes also contribute to the expanding understanding of the software process and product in the environment. The SEL captures these attributes by collecting defect data. The SEL defined its own classes of errors to ensure internal consistency in the data. Types of errors include

- Computational errors—improper calculations within the source program, such as writing the wrong form of an expression.
- Interface errors—include both internal and external errors and represent invalid information (e.g., wrong data) being passed between modules, such as in a subroutine call.
- Logic/control errors—errors in flow control in a program, such as incorrect branches as the result of evaluating an if-statement expression.
- Initialization errors—improper settings of the initial value of variables.
- Data errors—wrong variable used in a calculation.

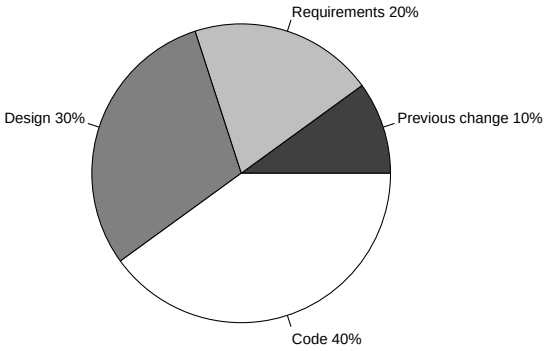
The SEL continually collects error data (starting when unit test is completed and continuing through delivery of the software and during maintenance) so that it is possible to continually understand the numbers and types of errors occurring in the software. This information is as important as the effort data. Together, they constitute two of the most critical core measures that the SEL has found. On maintenance projects, defect data are collected on a modified form which the SEL developed in 1990 when the organization became responsible for software maintenance as well as development.

Over 2000 errors were classified and studied from the projects in the 1985-1990 baseline. The error class distribution as well as the origin of errors (i.e., during what phase/activity the defect entered the software) are shown in Figure 9.

An earlier SEL study of errors provides an example of how models of software characteristics can be developed. By tracking five projects of similar complexity and size, the uncovered errors showed a decreasing step function for their rate of detection during sequential phases of the projects. From these data and trends, the SEL developed an internal model of expected error occurrence and detection rates for its class of software (see Figure 10). More recent studies show that the step function is still present, although the error rates have decreased significantly.



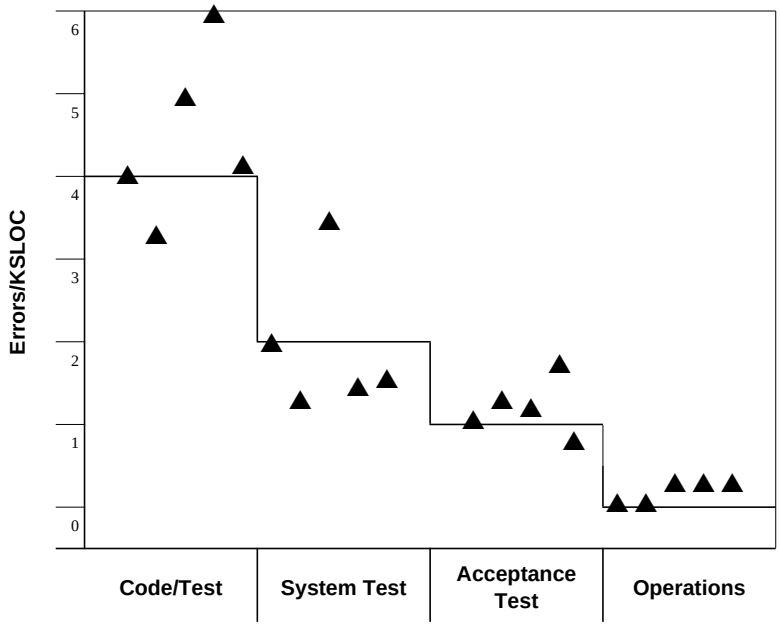
Classes of Errors*



Origin of Errors*

* Data from approximately 11 projects over 5 years (over 4000 errors sampled)

Figure 9: SEL Error Characteristics



* **Based on five similar projects in SEL**

Figure 10: Error Detection Rate Model

In addition to effort and defect data, other parameters are useful for developing a total understanding of the local environment. By counting defects found during the development of the software, then counting defects found during the operation and maintenance phases, the SEL developed a general understanding of the overall reliability of the software. Models of characteristics such as defects, change rate, effort distribution, and documentation size all provide useful information toward the development of improved models of software. This leads toward the capability of engineering the software process with well-understood relations, models, and rules. Using a sampling of projects developed during the early years of the SEL (late 1970s to mid-1980s), a set of models and relations was produced which was used as the baseline for planning, managing, and observing change over time (see Figure 11). One of the more surprising observations was that, after years of operation, the models changed very slowly—even with the significant technology and process changes introduced over time. Figure 12 describes the characteristics of another set of software projects active during the latter part of the 1980s. The differences between this and the earlier models/relations are surprisingly small, but there is change.

Productivity	Code rate = 26 lines per day
Effort Distribution	
Pages of Documentation	Doc = 34.7 ($KSLOC^{.93}$)
Maintenance Cost	~12% development cost per year
Reuse Cost	

		Date	Activity
Design	=	26%	23%
Code	=	38%	21%
Test	=	36%	30%
Other	=	—	26%

FORTRAN	=	20% of new
Ada	=	30% of new

Figure 11: Initial SEL Models/Relations

Of all the models and relations that the SEL has developed during the understanding phase, the most useful for project planning and management and for observing change have been

- Effort distribution (cost characteristics).
- Error characteristics (numbers, types, origins).
- Change and growth rates (of the source code during development).

The first two of these have been described in some detail in this section. These very basic pieces of information are being collected continually; they are used to observe change and improvement and to assess process impact.

Component	Type	Size (SLOC)	% Reused Each Mission	Development Duration (months)	Effort per Mission (staff-years)
Attitude (e.g., determination, control, calibration, simulation)	Mission-unique	250,000	25	27	40
Orbit/tracking data processing	Mission-general	800,000	95	12-18	2
Mission design/analysis	Mission-general	200,000	85	12-18	5
Orbit maneuver support	Mission-general	100,000	60	12-18	5

Figure 12: More Recent SEL Software Characteristics (late 1980s)

5.2 Assessing

After establishing a baseline of process, product, and environment characteristics and determining organizational goals, the next step in applying the SEL paradigm is to assess the value of any process change. In the SEL, these assessments are called “experiments,” and each project that is developed in the production environment is viewed as an experiment. Some of the studies are meant only to establish models of process or product, while other experiments are designed to evaluate the impact that a significant process change may have on the local software business—both process and product. Some of the experiments do not make overt changes to the established development process in the SEL, but are monitored mainly to establish the baseline understanding of the process. Additionally, some technologies require multiple projects to be completed before the impact of the change can be fully understood and before recommendations can be made for tailoring the process for local use.

It is perhaps useful to comment here on the relationship between the SEL paradigm and the CMM. Others have contrasted the two as alternative approaches to solving the same problem—process improvement. However, the SEL’s understanding/assessing/packaging model and the CMM are quite distinct. The SEL uses a specific process model to drive the organization. Via this model, decisions are made as to how to manage a software development project. Other development models certainly exist, such as a straightforward “waterfall” or MIL-STD-2167A. On the other hand, the CMM is an assessment model, and can be used to assess the process improvement of the SEL as well as other organizational structures. Thus the two concepts are actually complementary and can be used separately or jointly in any organization.

The structure of the SEL, as a partnership of GSFC, CSC, and the University of Maryland, has permitted a wide variety of experiments to be conducted, maximizing the skills and resources of each of the contributing organizations. Experiments have ranged across numerous technologies, from minor process change (e.g., adding code-reading techniques to measure resulting error rates) to major process change (e.g., object-oriented design, Cleanroom, Ada). Through the experimentation process, the SEL has gained broad insight into the impacts of these technologies and processes and has reported extensively on its findings. Some representative studies are discussed in the paragraphs to follow. They include assessments of

- Design approaches
- Testing techniques
- Cleanroom methodology
- Ada/OOD
- Independent verification and validation (IV&V)

5.2.1 Studies of Design Approaches

Some studies require only an understanding of the current development environment. These are low-impact studies that can be undertaken with little risk to projects under development. The following design study is one such experiment.

In 1985, several experiments were conducted to determine the value of various design characteristics on the quality of the end product. This particular study used available information already being captured from development projects; there was no need to retrain the development personnel in particular design techniques. The goal was to determine if the “strength and coupling” criteria described by Constantine and Meyers [Stephens 74] could be used as a predictive metric to determine the reliability of software.

A set of 453 software modules was selected from 9 completed projects for which detailed measurement information existed. The measures included design characteristics, number of defects found in the modules, and module size. This study was described in detail in a paper presented at the 1985 International Conference on Software Engineering [Card 85].

Strength was measured by the number of functions performed by an individual module, as determined by the authoring programmer. The 453 modules were classified in the following way:

- 90 modules were of low strength and averaged 77 executable statements.
- 176 modules were of medium strength and averaged 60 executable statements.
- 187 modules were of high strength and averaged 48 executable statements.

As a control, module size was also used. Small modules had up to 31 executable statements; medium-sized modules had up to 64 executable statements; and large modules had more than 64 executable statements. Error rates were classified as low (0 errors/KLOC), medium (<3 errors/KLOC), and high (> 3 errors/KLOC).

In analyzing error rates for these modules, strength proved an important criterion for determining error rates (see Figure 13) and proved more effective than simply using size as a predictor for defects. For example, 44 percent of the low-strength modules had high error rates; for high-strength modules, error rates ranged from 44 percent to only 20 percent. On the other hand, using size as a predictor of error, 27 percent of large modules were error prone while 36 percent of small modules were error prone, indicating that high strength was a good indicator of fewer defects.

Using all of the data available for the study, the SEL's baseline understanding for strength became:

- Good programmers tend to write high-strength modules.
- Good programmers tend not to show any preference for particular module size.
- Overall, high-strength modules have a lower fault rate and cost less than low-strength modules.
- Fault rate is not directly related to module size.

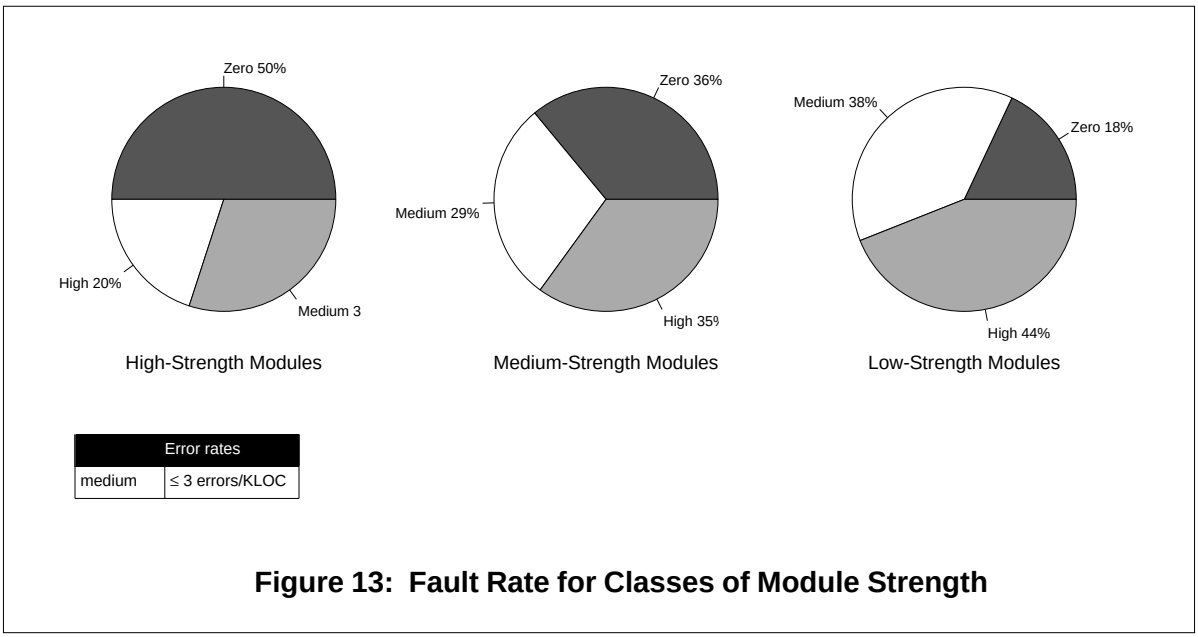


Figure 13: Fault Rate for Classes of Module Strength

5.2.2 Studies of Testing

Some studies are best carried out in small controlled environments. Using the university environment as an initial testing laboratory is useful for these studies. After validating the results in the university environment, the concept can be applied in an operational setting. The following testing experiment is an example of that approach.

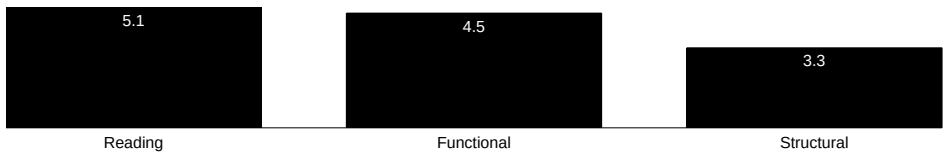
Reliability of the software produced is of continuing concern to the SEL. The goal of one study was to evaluate several testing techniques in order to determine their effectiveness in discovering errors. The techniques evaluated in this experiment were

- Code-reading of the source program by programmers other than the authors.
- Functional (i.e., black box) testing of the source program to the specifications (i.e., in-out behavior) of the program.
- Structural (i.e., white box) testing by developing test cases that execute specific statement sequences in the program.

Initially, a study was performed at the University of Maryland using 42 advanced software engineering students. Based upon positive results of this initial study, 32 programmers from NASA and CSC were recruited. All knew all 3 techniques, but were most familiar with the functional testing approach generally used at NASA. Three FORTRAN programs were chosen (ranging from 48 to 144 executable statements containing a total of 28 faults). All 32 programmers evaluated the 3 programs using a different testing technique on each program.

The main results of this study can be summarized as follows:

- Code reading was more effective at discovering errors than was functional testing, and functional testing was more effective than structural testing (see Figure 14).
- Code reading was more cost effective than either functional testing or structural testing in number of errors found per unit of time (see Figure 15). Structural testing and functional testing had about the same costs.



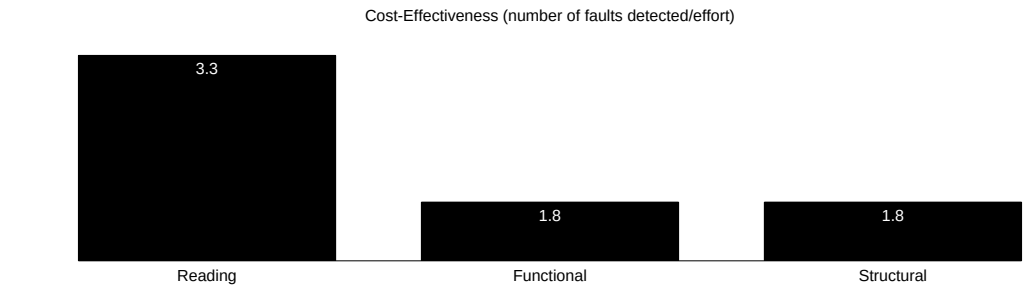
Code reading uncovered more errors than other methods; functional testing uncovered more errors than structural testing ($\alpha < .005$).

While different quantities of faults were detected in each program, the percentage of faults detected/program was the same.

Advanced students uncovered more faults than other students ($\alpha < .005$); intermediate students uncovered the same amount of faults as junior students.

Percent faults detected correlates with percent felt by tester to have been uncovered. R = .57 ($\alpha < .001$).

Figure 14: Fault Detection Rate by Testing Method



Code reading was more cost-effective than other methods [(α < .005), est +1.5(.4)].

There was a different overall detection rate for one program.

Techniques did not vary in total detection time.

Figure 15: Cost of Fault Detection by Testing Method

The study also produced some interesting results concerning programmer expertise and the discovery of faults. Space does not permit a full explanation here (see [Basili 87] for further details), but the results can be summarized as follows:

- The FORTRAN program built around abstract data types had the highest error discovery rate. This was an early indicator of the value of OOD.
- More experienced programmers found a greater percentage of the faults than less experienced programmers.
- Code reading and functional testing found more omission and control faults than structural testing. Code reading found more interface faults than the other two techniques.

This study, besides providing an assessment of the value of each of the testing techniques, adds to our understanding of the underlying baseline technology for later experiments.

5.2.3 Studies with Cleanroom

The following study of Cleanroom software development is an example of the use of pilot studies of new processes that pose great risks to the development organization. In this case, the method was studied for several years at the University of Maryland before being testing in the SEL operational environment.

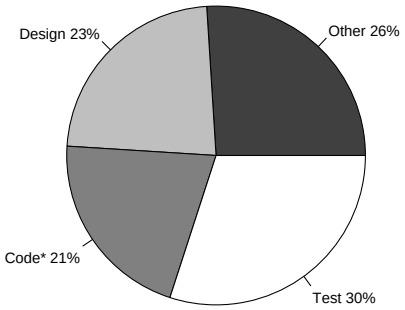
Reliability and defect rates have always been important components of understanding the environment. The Cleanroom technique, developed by Harlan Mills of IBM, proposed to radically alter how programs are developed in order to affect these rates. The SEL looked at Cleanroom as another process that might significantly improve their development process.

The idea behind Cleanroom is relatively simple. After a programmer implements a function, the programmer must verify that the function meets its specification, rather than relying on unit testing to show that it apparently works. Cleanroom, then, has the following attributes:

- Coding takes longer than traditional development because the verification step must be added. Programmers must truly understand their programs in order to verify the functions.
- Understanding and verifying functions results in significantly fewer errors, which results in much less system test—an expensive part of development.
- Overall result is lower cost and improved reliability.

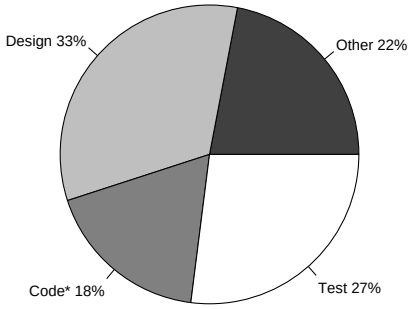
Since 1988, several projects have been developed in the SEL using the Cleanroom methodology. To prepare developers for using the Cleanroom technique, a series of training courses were given. A pilot project was undertaken which proved to be very successful. Time to understand the method (from training until the start of the second Cleanroom project) was approximately 26 months. Two follow-on Cleanroom projects were undertaken. A smaller in-house development was very successful, but a larger contracted project was not so successful. It was not clear whether problems on the larger project were due to scaling up of Cleanroom to larger tasks or to a lack of training and motivation of the development team on this project. Because of the differences that Cleanroom imposes on the development process, a fourth Cleanroom project is now underway for evaluation before declaring the technique "operational."

Compared to the SEL baseline process, it was clear that the Cleanroom development process was different (Figure 16). Design time and code reading grew significantly, while code writing and testing times all dropped. Defect rates improved (Figure 17) although productivity remained about the same using this new technology. The results of these studies are reported in more detail in [Basili 94].



Typical SEL Effort Distribution

* code writing: 85%
code reading: 15%



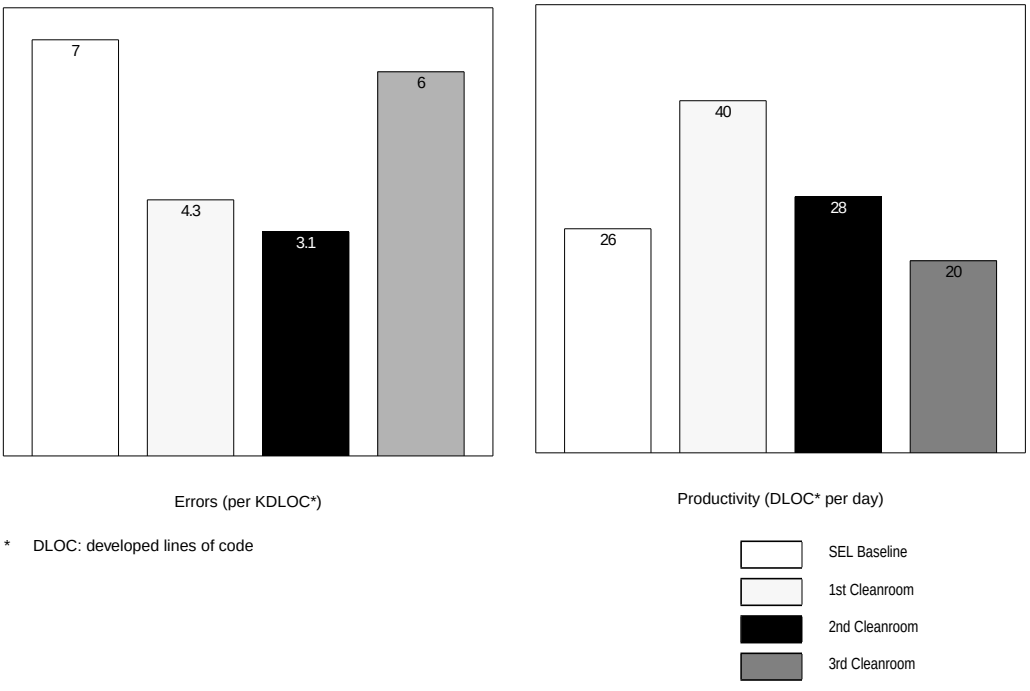
SEL Cleanroom Effort Distribution

* code writing: 48%
code reading: 52%

- Increased design effort with Cleanroom
- Code writing/code reading breakdown:

	SEL Baseline	SEL Cleanroom
writing	85%	48%
reading	15%	52%

Figure 16: Results of Cleanroom Experiment



* DLOC: developed lines of code

Figure 17: Assessing Cleanroom Against Goals and Expectations

5.2.4 Studies with Ada and OOD

Some studies impose a great risk on the development organization. In such cases, experiments must be carefully controlled. The SEL evaluation of Ada was one such study. This experiment also shows the difficulty of trying to isolate single processes for evaluation.

FORTRAN had always been the preferred programming language within NASA, but during the mid-1980s there was considerable interest in whether Ada should become their "language of choice." The SEL had a baseline understanding of the FORTRAN development environment, but needed to develop a corresponding baseline for Ada. A controlled experiment was designed where the same onboard computer simulator would be developed in both Ada and FORTRAN in order to compare the two languages.

In 1984, the GROSS project developed the operational FORTRAN simulator while a few months later an independent group, after first undergoing an intensive training program in the use of the language, developed the same simulator (GRODY) using Ada.

The major result from this initial study was an improved understanding of the requirements used to specify NASA software. As the Ada simulator was being designed, it soon became apparent that the requirements document typically used in flight dynamics applications contained many functional design decisions inherent with an assumed use of FORTRAN. Based upon this finding, requirements for the simulator were respecified using an object-oriented approach indicating the use of OOD technology, data abstraction, and information hiding. Because of this redesign of the requirements, the SEL study encompassed both the applicability of Ada in the FDD and the use of OOD techniques.

The GROSS-GRODY experiment was considered successful enough to try to use Ada on an actual mission, so several additional Ada projects were developed between 1987 and 1990 (see Figure 18). As the SEL learned about Ada, the characteristics of Ada programs began to change: packages became smaller; use of generics rose; use of tasking dropped; and there was a greater use of the Ada typing mechanism as the programming staff became more familiar with the features of the language (Figure 19).

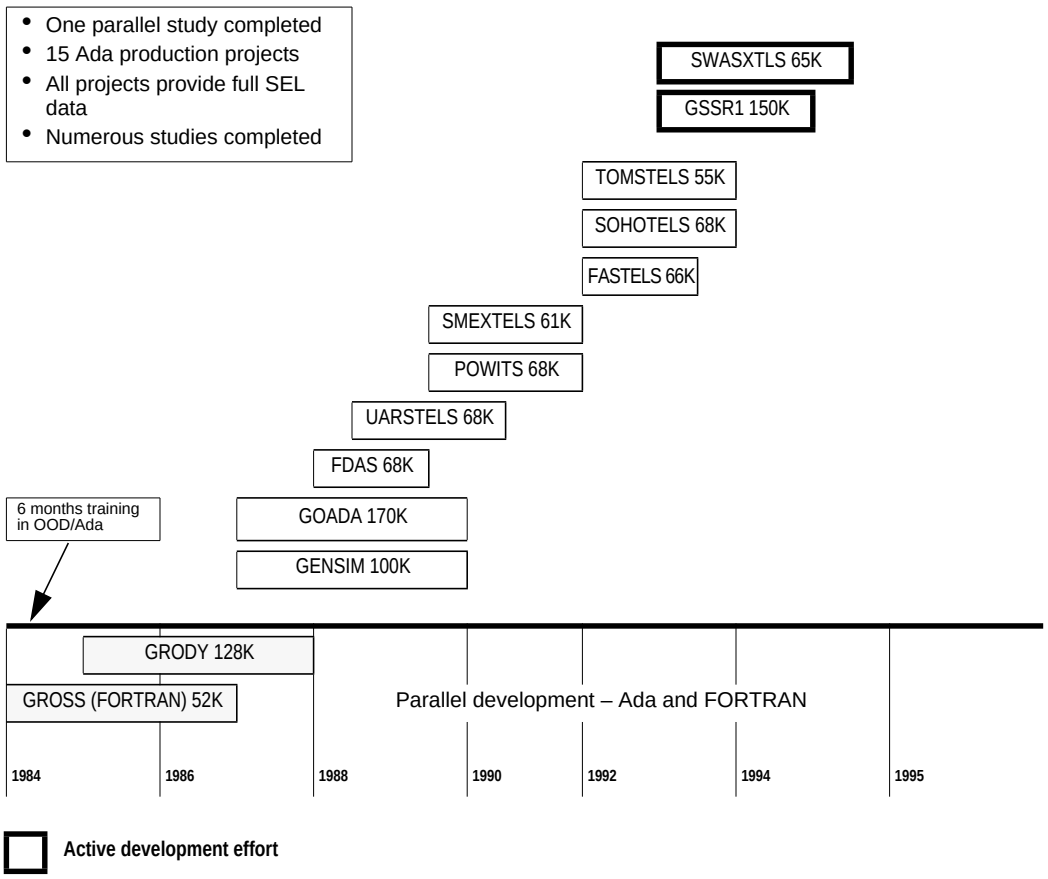


Figure 18: SEL Ada/Object-Oriented Technology (OOT) Projects

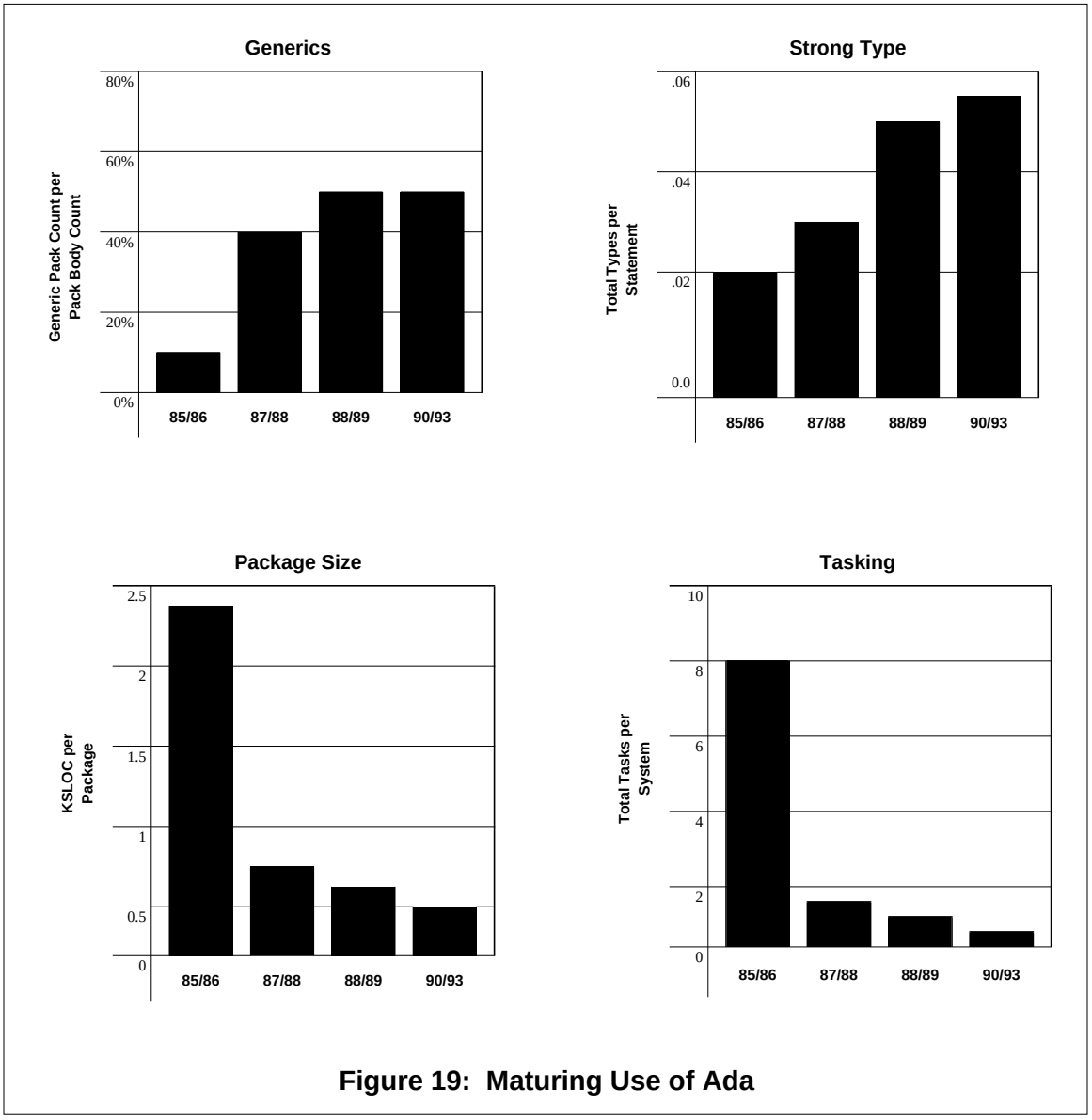


Figure 19: Maturing Use of Ada

From these initial Ada studies, the SEL developed a model of Ada software development as compared to the traditional FORTRAN baseline:

- First-time use of Ada resulted in a 30 percent increase in costs.
- In general, line-by-line, Ada code is more expensive than FORTRAN code.
- Reuse of Ada source code is higher than for FORTRAN, resulting in a decrease in program costs for Ada software.
- Error rates were similar to error rates in FORTRAN.

Some of the attributes in Figure 18 are not unique to the Ada language but, rather, represent general OOD features. Given that, the knowledge obtained from these studies was packaged as the *General Object-Oriented Software Development* [Seidewitz 86] for application on multiple projects in the environment. The result has been that FORTRAN programs, too, have greatly improved in their use of object-oriented techniques and in the reuse of components from system to system. Figure 20 shows the shortened schedules that have resulted from increases in reuse as object-oriented technology is increasingly employed on flight dynamics software. FORTRAN has continued to remain a competitive alternative to Ada as the technology has evolved.

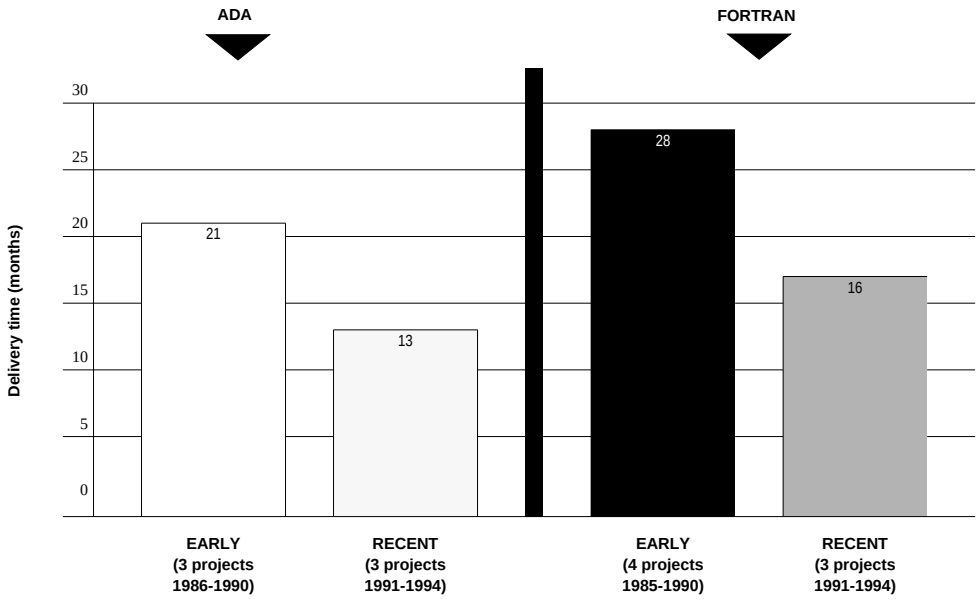


Figure 20: Reuse Shortened Project Duration

5.2.5 Studies with IV&V

Some process changes may not be appropriate for certain development organizations. The needs and goals must match the process. The following evaluation of IV&V was one such study.

A study conducted in the mid-1980s is representative of the more formal experimentation process that the SEL typically uses. Much literature had been published indicating the value of using IV&V during the development of large software systems, so the SEL considered adopting the methodology within the FDD production environment. However, before decisions were

made as to whether or not IV&V should become part of the standard process, several experiments were conducted to assess the cost, benefits, and compatibility of the technology for the SEL class of systems.

Two experiments were designed to test IV&V on two major software development efforts. (These studies are described in detail in [Page 84].) The goal of using the technology was to drive software error rates down, while maintaining a relatively cost-effective development process. Each project was approximately 65 KSLOC and was typical of previous SEL tasks. The IV&V tasks had 3 full-time programmers and each project took approximately 16 months from design through acceptance. The initial expectations for these projects were

- Increases in discovery of defects and quality of the operational software.
- Decreases in design flaws, costs of correcting errors, and system test effort.
- No changes in total defects reported.

The requirements on the IV&V team were

- Verify the requirements and design of the implemented system.
- Perform separate system testing. Validate consistency of the system to its requirements.
- Do not debug the programs, but report all anomalies.

The results of the IV&V study are shown in Figure 21 and are summarized below:

- Productivity dropped due to the increased costs of performing the IV&V function.
- Errors found before system test were generally higher than the SEL average, but not excessively so.
- IV&V did not significantly affect the overall error rate of SEL software.
- IV&V errors cost about the same to fix as errors in previous SEL projects.

While IV&V has been proposed in environments where it is critical to achieve a high degree of reliability, that situation was not apparent in the SEL environment. For the class of software that the SEL develops, IV&V was not deemed to be effective in improving either the reliability or overall cost of developing flight dynamics software.

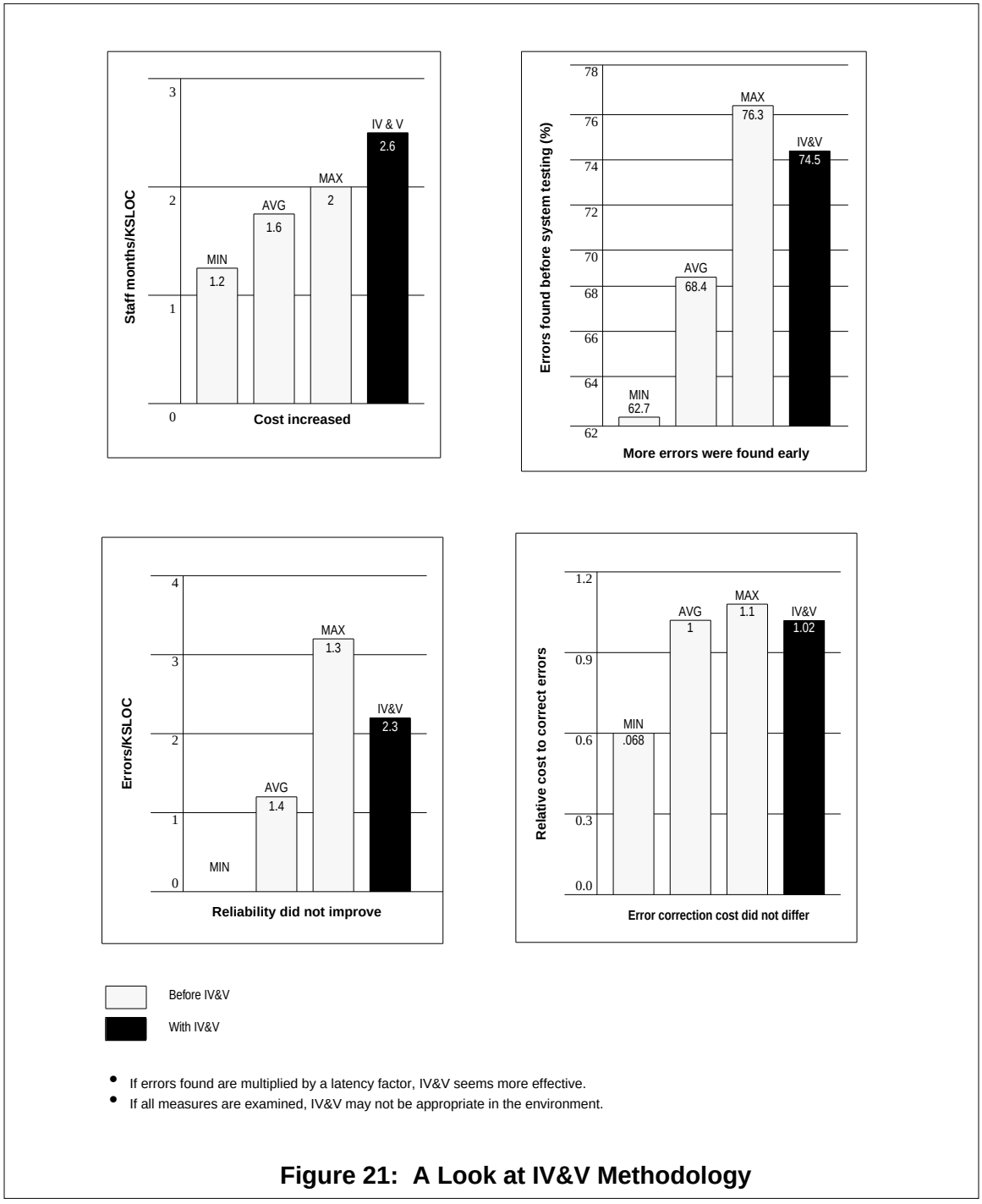


Figure 21: A Look at IV&V Methodology

5.2.6 Additional Studies

In addition to the studies described, the SEL has experimented with numerous other technologies including testing coverage, code-reading techniques, computer-aided software engineering (CASE) technology, structured techniques, documentation approaches, defect

causal analysis, reuse approaches, and functional testing vs. structural testing, as well as many variations of these methodologies. For a complete list of SEL reports and publications see the *Annotated Bibliography of SEL Literature* [Morusiewicz 93].

Probably the most important lesson that has been derived from the studies is that specific techniques can help the overall goals of process improvement when appropriately selected and tailored. However, the most effective element of the improvement paradigm is the continuous analysis of the software business and the continuous expansion of the understanding of the software process and product.

5.3 Packaging

As the experiments provide additional insight into the most appropriate techniques, tools, and processes, results are identified and captured in the form of policies, standards, tools, and training for the development organization. The results of the understanding and analysis phases are captured and packaged for “reuse” by ensuing projects, so that they become part of the routine software business.

For each of the studies conducted, the SEL analysts generate reports of results and conclusions. The reports may be papers for professional conferences, internal reports, or technical reports. Although the reports are valuable, the full value of the process analysis is felt when modifications and enhancements are made to the instruments that actually guide the way the development/maintenance organization carries out its business. These include standards, policies, and training classes. The SEL development organization uses a standard set of policies that is updated on a periodic basis to reflect new experimentation results. For instance, the *Manager’s Handbook for Software Development* [Landis 90] reflects the process that the managers use on the flight dynamics systems. This handbook contains the models, guidelines, and acceptable processes expected to be applied on each of the development efforts. It is periodically modified to reflect the completed studies of production projects. A full set of standards represents the changing process for this environment [Landis 92].

An important packaging concept is the infusion of technology in the form of support tools for use by project personnel. The SEL developed a project management tool called the Software Management Environment (SME). SME provides project managers access to the SEL database of previous project data and access to the baseline set of SEL process models. Using the SME, a manager can, for example, compare the growth rate of source programs or the growth rate of errors, or, using data from similar projects in the database, the manager can predict future activities on the current project. (For more details on the SME, see [Hendrick 92].) Tools such as SME help institutionalize the packaging of the SEL process, because they do not require operational personnel to know all of the details of each model in order to use them to gain insight into their software projects.

SEL experiences have been packaged in a variety of formats. Figure 22 summarizes some of the recent SEL processes that have been understood, assessed, and then packaged.

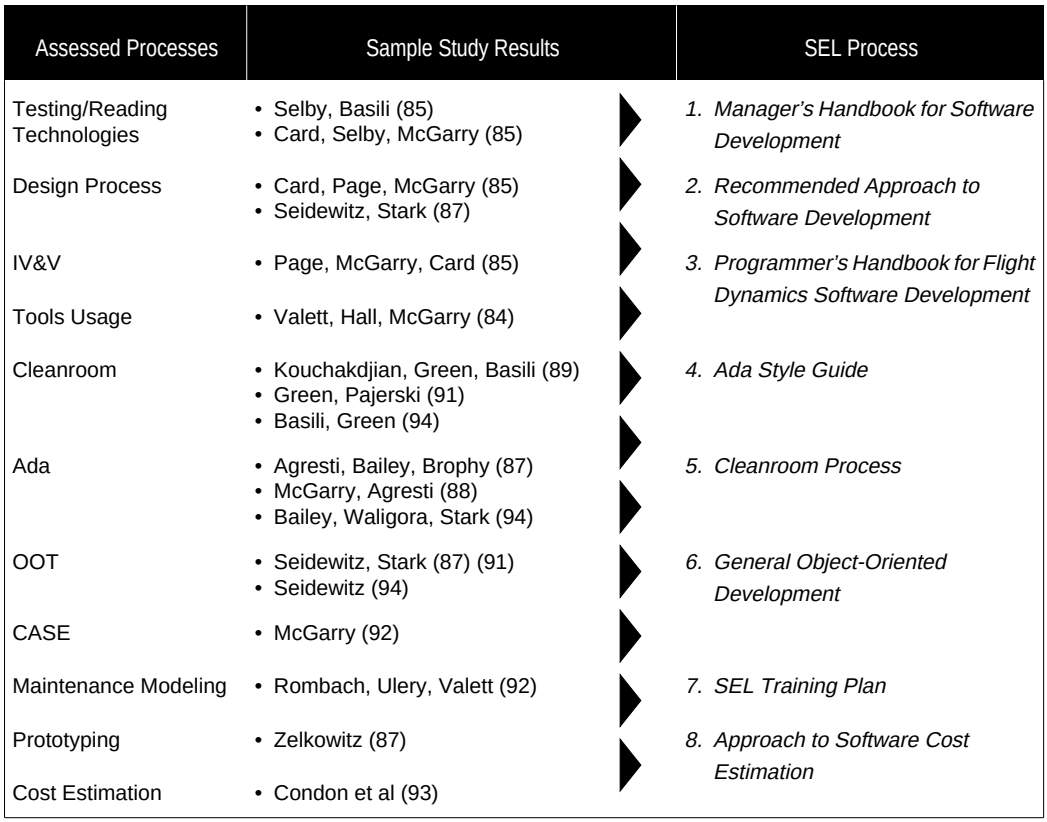


Figure 22: SEL Packaging: SEL Software Process

As part of the packaging process, the SEL has developed a standard set of training courses that is designed to provide all of the developers, managers, analysts, and database support staff with the information needed to function effectively in the FDD environment. Courses cover the SEL software process improvement concepts, software development methodology, software management approaches, standards, and organizational guidelines. This core set of courses reflects the experimental results, the process improvement approach, and, in general, all of the experiences of the SEL. These core courses are continually updated to reflect new and changing experiments within the SEL.

In addition to the core courses, the SEL staff provides training in any technology, methodology, or process that is planned as part of a SEL study when the technology or process is unfamiliar to the development teams. For instance, extensive training was provided in Ada and OOT before any attempt was made to apply these technologies on development projects. Other training has included Cleanroom, inspections, and CASE. If the SEL staff does not possess

the skills or knowledge to teach the courses, appropriate instructors may be recruited from elsewhere in the organization or outside vendors may be contracted to provide the training.

All SEL staff (managers, developers/maintainers, analysts, and database support) are required to participate in the core set of training classes, while the staff from specific development experiments attend specialized training addressing the processes under study.

6 The SEL Impact

The SEL has invested extensive time, energy, and resources in its efforts to better understand software process and its impact on software products. SEL studies have involved over 120 projects and perhaps as many software technologies, ranging from development and management practices (e.g., structured technologies), to automation aids (e.g., CASE and development tools), to technologies that affect the full life cycle (e.g., Ada, OOD). The results of SEL experiments have been captured in a large collection of reports, papers, and documents. These publications are available to the public at no charge and are used as the foundation for extending the studies within the SEL. See [Morusiewicz 93] for a complete list of SEL-published and SEL-related literature.

6.1 Impact on Product

Individual studies often resulted in specific improvements on the project being studied, but many experiments resulted in no measurable improvements or even negative impact on the end product. The major goals of the SEL from the beginning called for significant overall improvement in three product measures:

- Decrease in the defect rate of delivered software.
- Decrease in the cost of software to support similar missions.
- Decrease in the average cycle time to produce mission support software.

The additional measure of predictability has also always been a goal, but this is a more subjective measure and is more difficult to quantify. Detailed measures from the projects allowed the SEL staff to observe trends in the key measures over time and to analyze specific changes by comparing similar classes of software supporting similar classes of projects. In addition to the information that characterizes the measures identified above, additional data collected on all projects support more extensive comparisons of other product attributes.

To determine the general impact of the sustained efforts of the SEL as measured against its major goals, comparisons were made between a group of projects developed between 1985 and 1989 (the early baseline) and a group of similar projects developed between 1990 and 1993 (the current baseline). Projects were grouped based on size, mission complexity, mission characteristics, language, and platform. Similar types of comparisons have been made over longer periods of time, as well as comparisons made on smaller sets of projects in varying classes. The goal of these analyses was to assess the return on investment derived from the process improvement program. This was measured as improvement in the end product in the three key measures: defects, cost, and cycle time.

The early baseline comprises 8 projects completed between 1985 and 1989 (see Figure 23). These projects were all ground-based attitude determination and simulation systems developed on large IBM mainframe computers ranging in size from 50-150 KSLOC. All of these projects were considered successful in that they met mission dates and requirements within acceptable cost, and all of these projects applied some variation on the standard software process as part of SEL experimentation. The current SEL baseline comprises 7 projects completed between 1990-1993 (see Figure 24). The analysis focused on a comparison of defect rates, cost, cycle time, and levels of reuse. Additionally, the reuse levels were studied carefully with the full expectation that there would be a correlation between higher reuse and lower cost and defect rates.

Project	% Reuse	Cost [1] (Staff-months)	Reliability (Error/KDLOC)
GROAGSS	14	381	4.42
COBEAGSS	12	348	5.22
GOESAGSS	12	261	5.18
UARSAGSS	10	675	2.81
GROSIM	18	79	8.91
COBSIM	11	39	4.45
GOESIM	29	96	1.72
UARSTELS	35	80	2.96

[1] **Mission cost = cost of telemetry simulator + cost of attitude ground support system.**

Figure 23: Early SEL Baseline (1985-1989)

Project	% Reuse	Cost [1] (Staff-months)	Reliability (Error/KDLOC)
EUVEAGSS	81	155	1.22
SAMPEX	83	77	.76
WINDPOLR	18	476	n/a[2]
EUVETELS	96	36	.41
SAMPEXTS	95	21	.48
POWITS	69	77	2.39
TOMSTELS	97	n/a[3]	.23
FASTELS	92	n/a[3]	.69

[1] **Mission cost = cost of telemetry simulator + cost of attitude ground support system**

[2] **Cleanroom project excluded because errors are counted differently.**

[3] **Ongoing projects; final costs unavailable.**

Figure 24: Current SEL Baseline (1990-1993)

The early baseline projects had a development defect rate that ranged from a low of 1.7 errors per KSLOC to a high of 8.9 errors per KSLOC with the average rate being 4.5 defects per KSLOC. The current baseline projects had a defect rate ranging from a low of 0.2 to 2.4 errors per KSLOC with the average being 1 error per KSLOC. This reliability measure showed a decrease in the defect rate of approximately 75 percent over the 8-year period (see Figure 25).

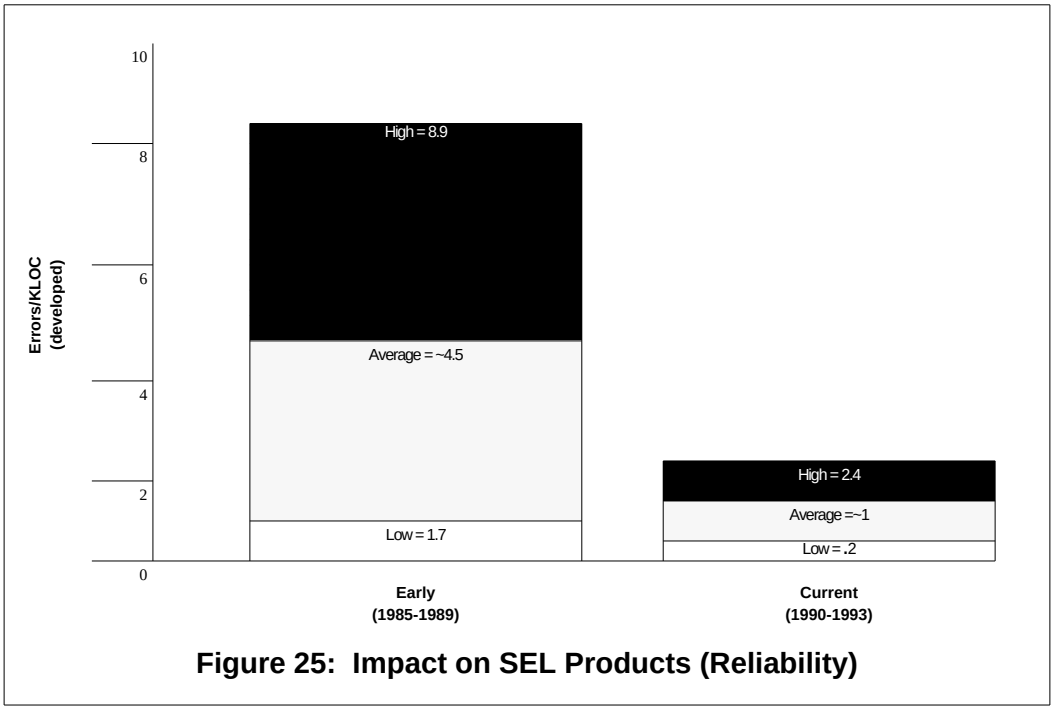


Figure 25: Impact on SEL Products (Reliability)

Software cost was also compared between the 2 baselines. The mission cost is defined as the total cost of all the flight dynamics software required to support the flight project. An examination of the selected missions from the 2 baselines revealed that while the total lines of code produced to support the specific missions has remained relatively close, the total mission cost has decreased significantly. The average mission cost in the early baseline ranged from a low of 357 staff-months to a high of 755 staff-months with an average of 490 staff-months. The current baseline projects had costs ranging from a low of 98 staff-months to a high of 277 staff-months with an average of 210 staff-months. Figure 26 shows the comparison of the cost data. The significant decrease in cost can be attributed to increases in both productivity and code reuse (Figure 27). This comparison shows that the average cost per mission has decreased by over 50 percent over the 8-year period.

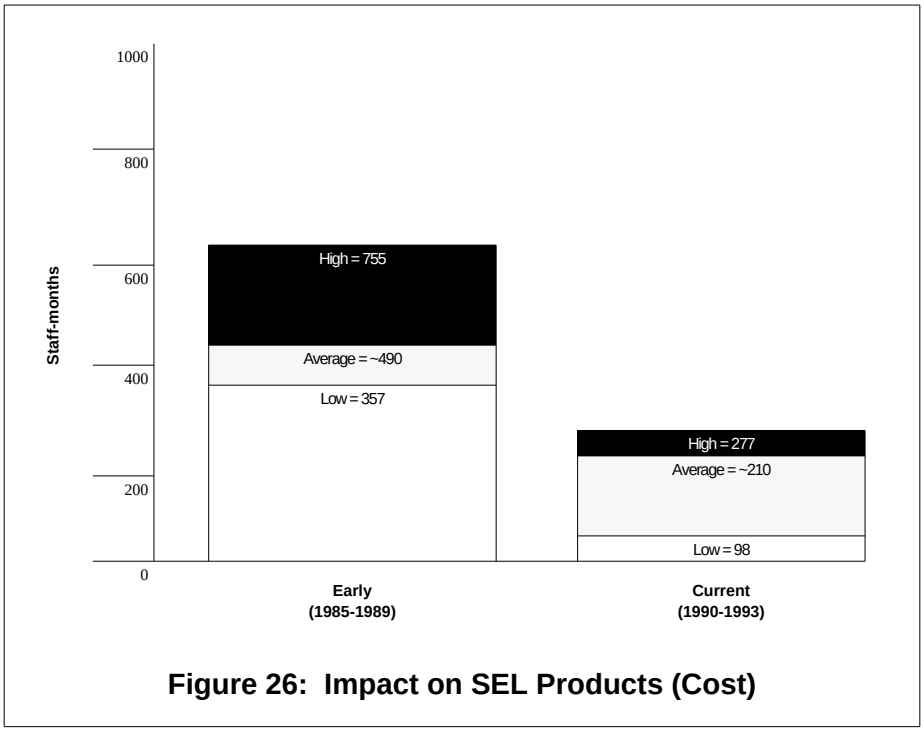


Figure 26: Impact on SEL Products (Cost)

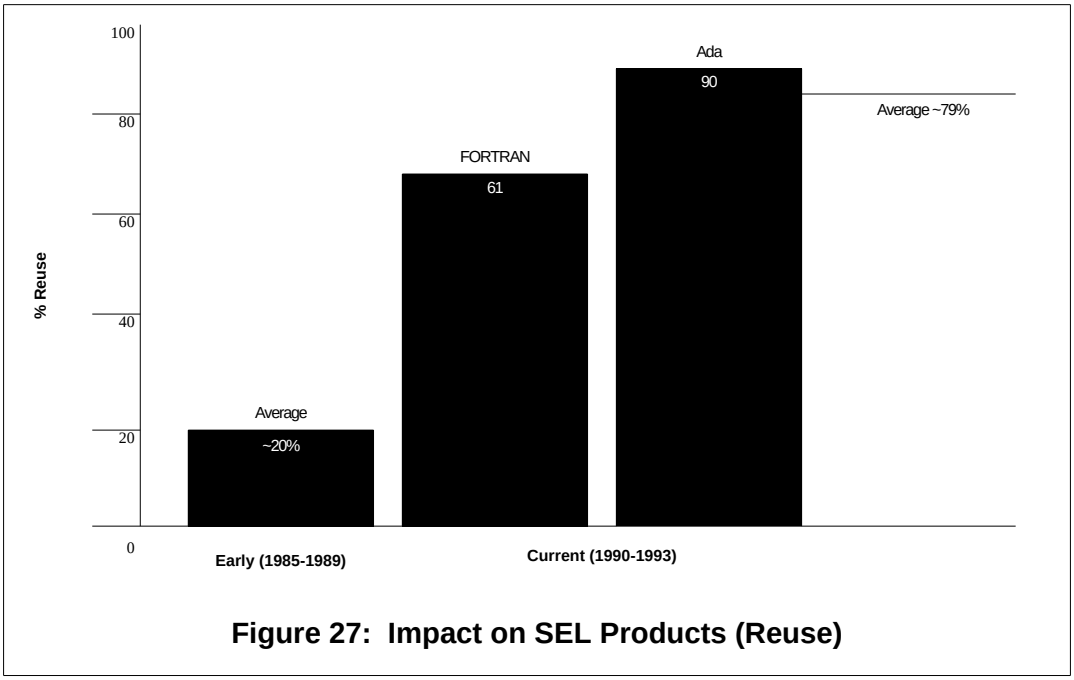
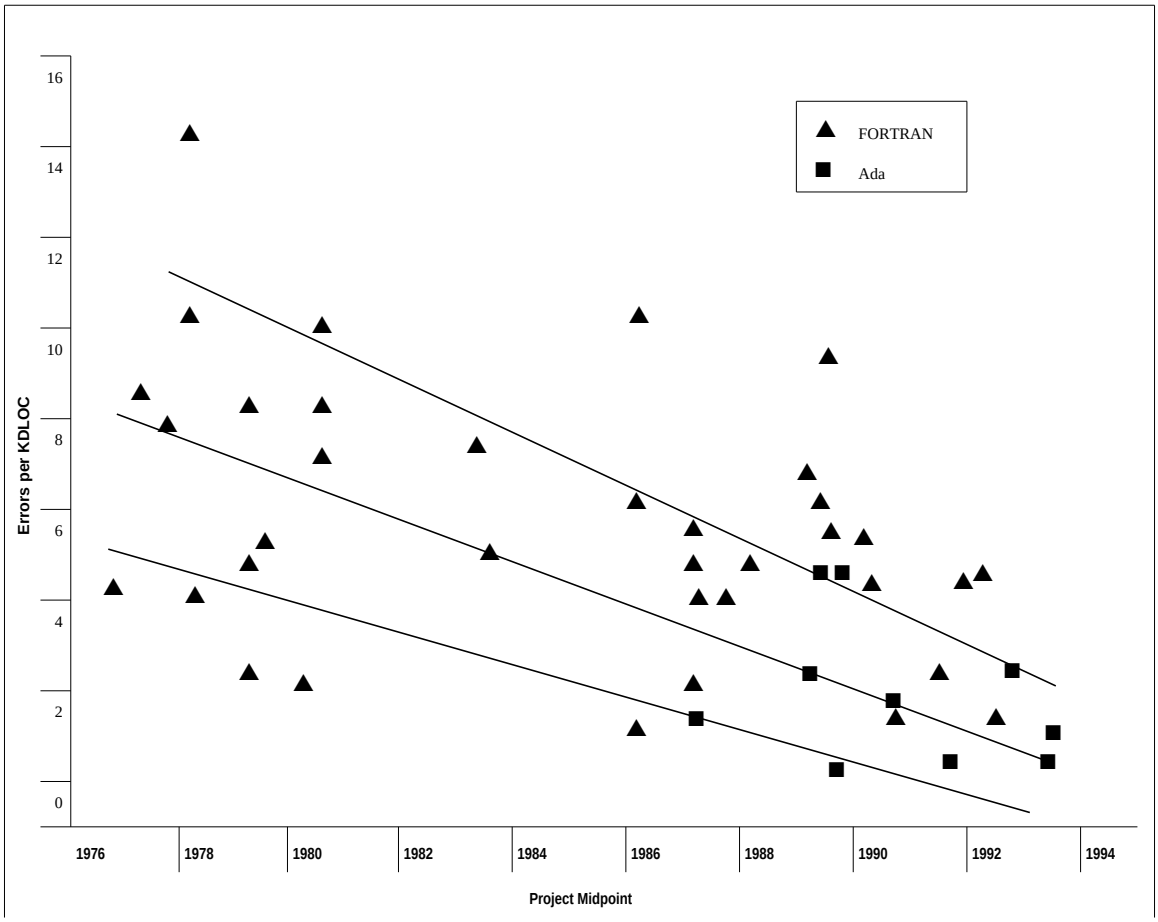


Figure 27: Impact on SEL Products (Reuse)

Through the experimentation and emphasis on the reuse of software in the SEL, detailed data have been tracked that characterize the trends in the reuse of software. Although code reuse represents only one measure of software reuse, it is one of the more measurable and more

easily understood, so the SEL uses it to measure reuse in its environment. Code reuse is defined as the total lines of application code in components (compilable units) that have been taken in their entirety from a previously completed system or application library. Commercial off-the-shelf products and multiple use of a module within the same system are not included in the computations.

In addition to examining the changes over recent years by comparing projects with similar characteristics, the long-term trends of reliability were examined for the full set of projects where accurate error data were available. Approximately 60 flight dynamics projects had accurate error data over the same phases of the life cycle. The error rate data were taken from these projects over the full lifetime of the SEL and were fit using a simple linear regression (shown in Figure 28). The data indicated that error rates decreased from approximately 7.5 errors per KSLOC to approximately 1 error per KSLOC—an improvement of over 75 percent.



6.2 Impact on Process

The SEL has reviewed in detail the process changes that have been tried and adopted over the lifetime of the improvement program. It would be satisfying to be able to point to a key technology or methodology change and to state that it had a direct, measurable link to a specific product improvement. However, it is difficult to isolate the impact of any one change in this environment. But the most significant changes that have been adopted can be identified by examining the policies, training programs, and development approaches that today constitute the SEL/FDD process. Although specific techniques or methodologies may have measurable impact on a class of projects, the significant improvement to the software development process is viewed at a higher level where the sustained, continuous incorporation of detailed techniques into higher level processes is the more significant payoff. The most significant process attributes that distinguish the current SEL production environment from the environment of a decade ago include the following:

- *Process change has been infused as a standard business practice.*
  The standards, policies, and training material all now contain elements of the continuous improvement approach to experimentation that has been promoted by the SEL.

- *Measurement is now our way of doing business.*
  Measurement is no longer treated as an add-on to development. The measurement activity is as common a part of the software standards and policies as documenting software. It is expected, applied, and effective.

- *Change is now driven by **product** and **process**, not merely process alone.*
  As the process improvement program has matured over the years, there has developed an equal concern about product attributes as well as process attributes. A set of product goals is always defined before process change is infused. Because of this, measures of product are as important as (and probably more important than) those of process.

- *Change is now bottom-up.*
  Although process improvement analysts originally assumed that they could work independently from the developers, the years have brought the realization that change must be guided by development-project experience. Direct input from developers as well as measures extracted from development activities are key factors in change.

- *"People-oriented" technologies are emphasized rather than automation.*
  The process changes that have been most effective are those that leverage the thinking ability of the developers. These include reviews, inspections, Cleanroom techniques, management practices, and independent testing techniques, all of which are driven by disciplined activities of the programmers/managers. Automation techniques have sometimes provided improvement, but not to the extent that people-driven approaches have.

6.3 Cost of Change

The benefits of the process improvement efforts are well substantiated by looking at the measures of software cost, error rates, and cycle time—all goals of the organization as change was being implemented. Not only has the SEL traced the detailed software measures throughout the 17-year lifetime, but expenditures incurred due to the efforts of process change have also been tracked quite closely. The SEL investment in process change activities can be divided into 3 significant areas:

- Project overhead
- Data handling, archiving, and technical support
- Process analysis

The total investment that the SEL has made in the improvement effort has been approximately 10 percent of the total software development cost in the FDD. Project overhead represents costs incurred due to developers attending training (in new processes), completing data collection forms, participating in interviews, and providing detailed additional information requested by the analysts. This overhead for data collection and process change is extremely small. It is now nearly impossible to measure except in the cases of very large process changes, such as using a new language (longer training, meetings, etc.). For projects participating in the routine process improvement efforts, the impact is less than 1 percent of the total software cost. A successful process improvement program does not require a large perturbation or cost to the development organization.

Data archiving and repository activities require a larger investment. Not only must measures be collected from the developers, but there must be a smooth process of data quality assurance, archiving, and reporting. This function of the SEL has cost approximately three percent of the total development budget. That is, the database support element has cost approximately eight million dollars over the lifetime of the SEL. This figure includes purchase and design of database management systems and distribution of SEL literature as well.

The analysis activity has been the most costly of all the expenditures. The responsibilities of the analysts include setting goals, defining experiments, interpreting measurement data, training the development/maintenance staff, developing standards and policies, and tailoring processes for particular needs. The analysts must provide refined processes to the development organization along with rationale of why one process is more appropriate than another. They must design and then provide any required training to the development organization.

SEL activities have cost approximately $25 million in an organization that has spent approximately $250 million on software development and maintenance. There is no empirical evidence to indicate whether this 1:10 ratio would persist when process improvement is expanded to a larger organization. Very preliminary data from SEL experience in carrying out a broader NASA-wide program indicate that project overhead would remain low, and that the

data repository could function with 15 persons to support development organizations that are orders of magnitude larger than the FDD. Figure 29 shows the observed process improvement cost based on the experiences of the FDD organization.

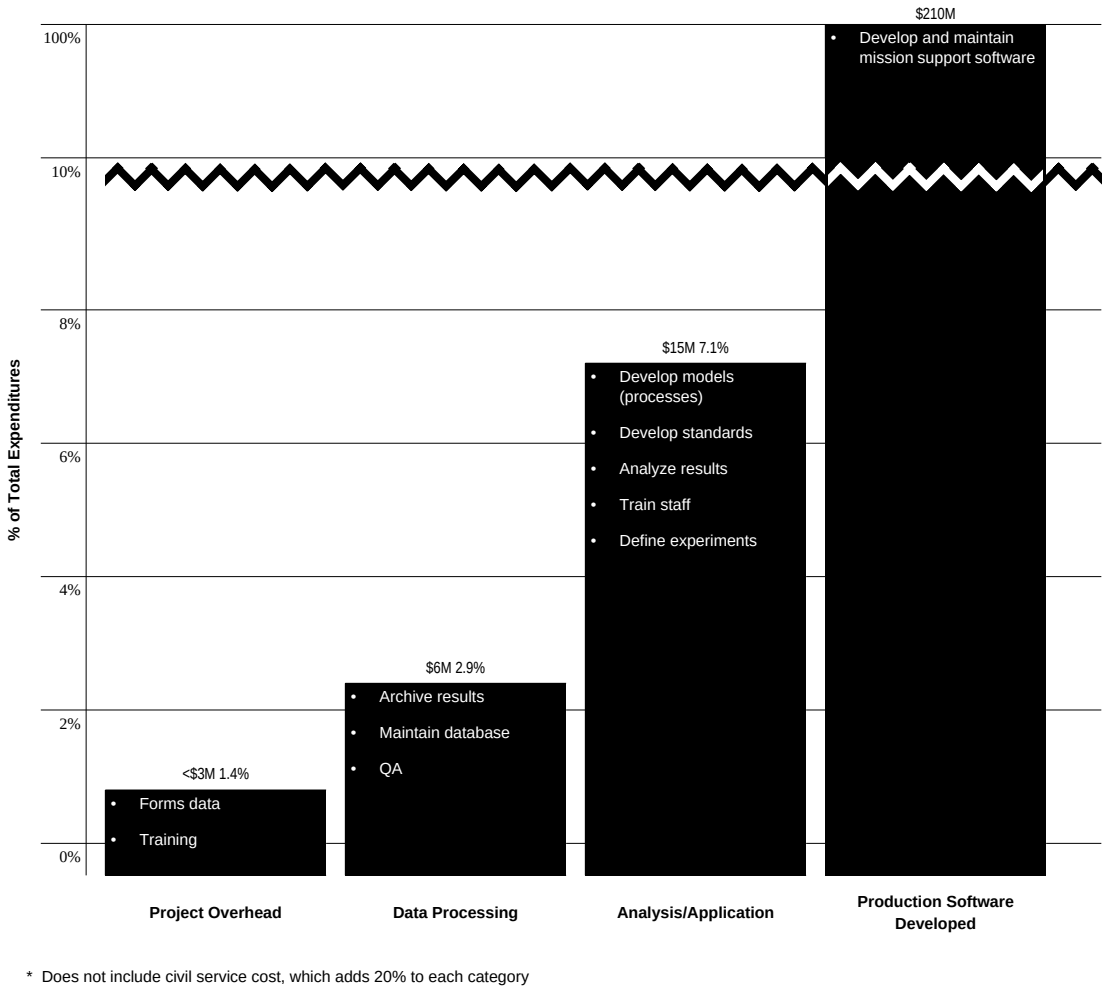


Figure 29: ROI for Process Improvement in the FDD

6.4 Impact on Other Organizations

The SEL has operated in one domain at NASA/GSFC for 17 years and the measures of change have been apparent. In addition to the impact of the process activities on this local organization, the overall concept of change and improvement has been adopted by other organizations at GSFC and across NASA.

Systems Engineering and Analysis Support (SEAS) Contract

CSC is the prime contractor on the SEAS contract (valued at $1 billion over 10 years) which supports NASA with a staff of nearly 1400 developers and analysts. This project has expanded the application of SEL concepts beyond the FDD. Drawing on the SEL's experience in documenting the methodology used in the flight dynamics environment and on CSC's general corporate methodology, SEAS has developed and documented a system development methodology for use across the entire contract and a set of standards and procedures to help staff members apply it.

Established guidelines for quantitative management are also used on the SEAS contract. Recognizing the importance of quantitative management, as shown by the SEL's success, SEAS has packaged its experiences in this area in a data collection, analysis, and reporting handbook to be used by all the managers.

The SEAS organization also offers a training program, modeled after the SEL's example. An established, required training program exists for all engineers, developers, testers, integrators, and managers working on the contract. This required training is designed to ensure consistent understanding and application of the SEAS System Development Methodology and of the quantitative management techniques used in the SEAS environment.

Earth Observing System (EOS) Core System Development Organization

The EOS Core System, one of GSFC's largest development efforts (totaling $760 million over 20 years), has adapted the SEL process improvement concept. The EOS program is managed by Hughes Applied Information System. SEL personnel are currently consulting with Hughes to help guide the development of a process improvement program for the EOS organization.

NASA Software Engineering Program

In 1991, a software engineering program modeled on the SEL's concepts was initiated by NASA Headquarters for application across the Agency. To date, GSFC and four other NASA centers have participated in the program. The first phase, baselining of all NASA software, is ongoing and scheduled for completion in December of 1994.

7 Summary

Over the course of its history, the SEL has pioneered a measurement-based paradigm for software process improvement. Long before it became fashionable to think of software development in terms of an “engineering discipline,” the founders of the SEL had the vision to recognize that software developers follow a process, and that if that process could be captured qualitatively and measured quantitatively, a baseline could be established against which improvements could be measured. What has evolved since that time is a unique “bottom-up” approach to software process improvement that is driven by a solid understanding of an organization’s environment (process and product characteristics), coupled with clearly defined organizational goals for process improvement. This approach eventually leads to a steady state that involves a continuous cycle of understanding the current baseline, introducing new methods and techniques in a controlled fashion and measuring their impact, refining those techniques deemed effective, and packaging the resulting process changes for infusion back into the organization’s standard process.

Over the years, the SEL has validated this paradigm and measured its impact in the FDD environment. It has executed experiments and studies ranging from small-scale comparisons of code reading and testing techniques to long-term evaluations of major technologies, such as Ada, object orientation, and Cleanroom. Each of these studies resulted in some modification to the standard flight dynamics software development process, usually in the incorporation of key concepts from the technologies studied tailored to maximize their leverage within the environment. As documented earlier in this report, the resulting impact on product measures has been significant, and when the costs of implementing the program are tallied, the return on investment is substantial.

The SEL is committed to sharing its findings as to the improvements it has witnessed in flight dynamics software development and what techniques have or have not made an impact. More importantly, however, the SEL is equally committed to sharing the process improvement paradigm it has forged, and all of the lessons it has learned along the way. Many of these results are published in software engineering journals and presented at major international conferences. In addition, the SEL sponsors an annual Software Engineering Workshop, with paper sessions, panels, and tutorials, that draws an audience of over 400 software engineering practitioners from around the world. The SEL has also captured its research results and process models in a series of documents, many of which have been sent to thousands of individuals and organizations. As discussed in Chapter 6, the SEL’s concepts have been adapted and adopted by other organizations, including Hughes Applied Information System, NASA Headquarters, and the SEL’s partner in industry, Computer Sciences Corporation. Thus, the SEL has had, and continues to have, a significant impact on the software engineering community.

Looking to the future, current efforts are just getting underway to establish SEL-based process improvement programs at various other U.S. Government agencies. There is also considerable interest from several sectors in examining the expansion issues related to applying the SEL paradigm beyond local environments to higher organizational levels. The SEL anticipates being actively involved in supporting these efforts. It also plans to continue supporting the process improvement goals of the FDD organization. Most importantly, perhaps, is the SEL's continued commitment to sharing its processes, findings, and lessons learned throughout the software community, and by so doing, advancing the state-of-the-practice in software engineering.

Appendix A Sample SEL Experiment Plan

SEL Representative Study Plan for SOHOTELS
October 11, 1993

A.1 Project Description

The Solar and Heliospheric Observatory Telemetry Simulator (SOHOTELS) software development project will provide simulated telemetry and engineering data for use in testing the SOHO Attitude Ground Support System (AGSS). SOHOTELS is being developed by a team of four GSFC personnel in Ada on the STL VAX 8820. The project is reusing design, code, and data files from several previous projects but primarily from the Solar, Anomalous, and Magnetospheric Particle Explorer Telemetry Simulator (SAMPEXTS).

The SOHOTELS team held a combined preliminary design review (PDR) and critical design review (CDR) in April 1993. In their detailed design document, the SOHOTELS team stated the following goals for the development effort:

- To maximize reuse of existing code.
- Where reuse is not possible, to develop code that will be as reusable as possible.
- To make sure performance does not suffer when code is reused.

A.2 Key Facts

SOHOTELS is being implemented in three builds so that it can be used to generate data for the early phases of the AGSS (which is a Cleanroom project). Build development and independent acceptance testing are being conducted in parallel. At present, the test team has finished testing SOHOTELS Build 1. The development team expects to complete Build 2 and deliver it to the independent test team by the end of the week.

SOHOTELS consists of six subsystems. As of June, the estimated total number of components was 435, of which 396 (91 percent) have currently been completed. Total SLOC for SOHOTELS was estimated at 67.6 KSLOC, with 46.6 KSLOC of code to be reused verbatim and 15.7 KSLOC to be reused with modifications. As of September 13, 1993, there were 65.4 KSLOC in the SOHOTELS system, or 97 percent of the estimated total.

The SOHOTELS task leader is currently re-estimating the size of the system because SOHOTELS will be more complex than was originally predicted. The new estimates will include SLOC for the schema files that are being developed.

The phase start dates for SOHOTELS are:

September 9, 1992	Requirements Definition
October 3, 1992	Design
May 1, 1993	Code and Unit Test
June 26, 1993	Acceptance Test
May 7, 1993	Cleanup

A.3 Goals of the Study

The study goals for SOHOTELS are

- To validate the SEL's recommended tailoring of the development life cycle for high-reuse Ada projects.
- To refine SEL models of high-reuse software development projects in Ada, specifically:
  - effort (per DLOC, by phase and by activity);
  - schedule (duration for telemetry simulators and by phase);
  - errors (number per KSLOC/DLOC);
  - classes of errors (e.g., initialization errors, data errors); and
  - growth in schedule estimates and size estimates (from initial estimates to completion and from PDR/CDR to completion).

A.4 Approach

The following steps will be taken to accomplish the study goals:

- Understand which of the standard development processes are being followed and which have been tailored for the SOHOTELS project. Ensure that information is entered into the SEL database that will allow SOHOTELS data to be correctly interpreted in light of this tailoring.
- Analyze project/build characteristics, effort and schedule estimates, effort and schedule actuals, and error data on a monthly basis while development is ongoing.
- At project completion, plot the effort, schedule, error rate, and estimate data. Compare these plots with current SEL models and with plots from other high-reuse projects in Ada. Compare and contrast the error-class data with data from FORTRAN projects, from Ada projects with low reuse, and from other high-reuse Ada projects.

A.5 Data Collection

To address these study goals, the following standard set of SEL data for Ada projects will be collected:

- Size, effort, and schedule estimates (Project Estimates Forms).
- Weekly development effort (Personnel Resources Forms).
- Growth data (Component Origination Forms and SEL librarians).
- Change and error data (Change Report Forms and SEL librarians).

Appendix B FDD Projects

The following tables are taken from the *SEL Cost and Schedule Estimation Study Report* [Condon 93]. They are included here to provide an overview of the basic characteristics of the many FDD projects studied by the SEL over the years. Table B-2 summarizes system size and reuse rates for the same projects.

Project	Type	Development Language	Duration Period[1]	Weeks	Technical and Management[6] Hours
PASA	AGSS	F	05/76 — 09/77	69	15760
ISEEB	AGSS	F	10/76 — 09/77	50	15262
AEM	AGSS	F	02/77 — 03/78	57	12588
SEASAT	AGSS	F	04/77 — 04/78	54	14508
ISEEC	AGSS	F	08/77 — 05/78	38	5792
SMM	AGSS	F	04/78 — 10/79	76	14371
MAGSAT	AGSS	F	06/78 — 08/79	62	15122
FOXPRO	AGSS	F	02/79 — 10/79	36	2521
DEA	AGSS	F	09/79 — 06/81	89	19475
DEB	AGSS	F	09/79 — 05/81	83	17997
DESIM	TS	F	09/79 — 10/80	56	4466
ERBS	AGSS	F	05/82 — 04/84	97	49476
DERBY	DS	F	07/82 — 11/83	72	18352
GROSS	DS	F	12/84 — 10/87	145	15334
GRODY	DS	A	09/85 — 10/88	160	23244
COBEDS	DS	F	12/84 — 01/87	105	12005
ASP	AGSS	F	01/85 — 09/86	87	17057
GROAGSS	AGSS	F	08/85 — 03/89	188	54755
GROSIM	TS	F	08/85 — 08/87	100	11463
COBSIM	TS	F	01/86 — 08/87	82	6106
COBEAGSS	AGSS	F	06/86 — 09/88	116	49931
GOADA	DS	A	06/87 — 04/90	149	28056
GOFOR	DS	F	06/87 — 09/89	119	12804
GOESAGSS	AGSS	F	08/87 — 11/89	115	37806
GOESIM	TS	A	09/87 — 07/89	99	13658
UARSAGSS	AGSS[2]	F	11/87 — 09/90	147	89514
ACME	AGSS[2]	F	01/88 — 09/90	137	7965
UARS_2	AGSS[2]	F	N/A	N/A	97479
UARSDSIM	DS	F	01/88 — 06/90	128	17976
UARSTELS	TS	A	02/88 — 12/89	94	11526
EUVEAGSS	AGSS	F	10/88 — 09/90	102	21658
EUVE_2[3]	AGSS	F	N/A	N/A	21658
EUVETELS	TS	A	10/88 — 05/90	83	4727
EUVEDSIM	DS	A	10/88 — 09/90	121[4]	20775[4]
SAMPEXTS	TS	A	03/90 — 03/91	48	2516
SAMPEX	AGSS[5]	F	03/90 — 11/91	85	4598
SAMPEXTP	AGSS[5]	F	03/90 — 11/91	87	6772
SAMPEX_2	AGSS[5]	F	N/A	N/A	11370
POWITS	TS	A	03/90 — 05/92	111	11695

Abbreviations: **A** **Ada** **AGSS** **Attitude Ground Support System** **DS** **Dynamics simulataor** **F** **FORTRAN simulator** **TS** **Telemetry simulator**

[1] Design phase through acceptance test phase.

[2] The AGSS for the UARS satellite was developed as two projects. One project, containing the majority of the AGSS code and functionality, was called simply UARSAGSS and was developed by CSC. The other project, containing two utilities (CFADS and STARID), was called ACME and was developed in-house by GSFC. When referring to the total size or effort of the two combined projects, this study uses the name UARS_2.

[3] The EUVE AGSS was developed as a single project, and the EUVEAGSS account in the SEL database includes all hours spent on this AGSS. In recording the lines of code in the EUVEAGSS account, however, the SEL database did not include the ACME lines of code, all of which were borrowed from the ACME project and reused verbatim in the EUVE AGSS. When referring to the size or productivity of the total EUVE AGSS, this study uses the name EUVE_2. The values for effort and schedule duration do not vary between EUVE AGSS and EUVE_2.

[4] Duration adjusted by +15% and Effort adjusted by +10% because EUVEDSIM did not have an acceptance test phase. These values are consistent with those of the *Ada Size Study Report.*

[5] The AGSS for the SAMPEX satellite was developed as two projects. The telemetry processor part, called SAMPEXTP, was developed in-house by GSFC. The other project, containing the majority of the AGSS code and functionality, was called simply SAMPEX and was developed by CSC. When referring to the total size or effort of the two combined projects this study uses the name SAMPEX_2.

[6] Includes technical staff and technical management hours for preproject through cleanup phases. Does not include support staff-hours (project management, librarians, secretaries, technical publications).

Table B-1: Projects Studied

Project	Newly Written	Extensively Modified	Slightly Modified	Reused Verbatim
PAS	84729	0	20041	7098
ISEEB	43955	0	3506	7776
AEM	45345	0	4673	893
SEASAT	49316	0	4252	21825
ISEEC	20075	0	6727	48618
SMM	76883	0	5652	2834
MAGSAT	61950	0	14297	13266
FOXPRO	5354	0	1323	2449
DEA	45004	0	9705	12616
DEB	44644	0	8606	13016
DESIM	14873	0	0	385
ERBS	137739	0	5767	15635
DERBY	37137	0	3901	4549
GROSS	33196	3493	8574	6441
GRODY	123935	1143	3037	146
COBEDS	26986	0	7363	2556
ASP	70951	0	0	10483
GROAGSS	194169	9982	18133	14109
GROSIM	31775	0	4294	2881
COBSIM	45825	1342	1156	4494
COBEAGSS	141084	16017	13647	7934
GOADA	109807	12496	41750	7049
GOFOR	22175	2867	6671	5330
GOESAGSS	106834	6377	9779	5869
GOESIM	59783	5784	15078	11450
UARSAGSS	260382	9340	21536	11868
ACME	34902	0	0	0
UARS_2	295284	9340	21536	11868
UARSDSIM	63861	17476	20710	4399
UARSTELS	38327	6114	12163	11544
EUVEAGSS	41552	13597	14844	179016
EUVE_2	41552	13597	14844	213918
EUVETELS	2161	371	5573	58591
EUVEDSIM	20859	36248	87415	39495
SAMPEXTS	0	3301	6120	52026
SAMPEX	10590	1631	1282	141006
SAMPEXTP	15899	1920	1777	36
SAMPEX_2	26489	3551	3059	141042
POWITS	12974	7980	20878	26275

Table B-2: Detailed Line-of-Code Data

References

[Basili 84] Basili, V. R., and D. M. Weiss, “A Methodology for Collecting Valid Software Engineering Data,” *IEEE Transactions on Software Engineering*, November 1984.

[Basili 87] Basili, V. R., and R. W. Selby, “Comparing the Effectiveness of Testing Strategies,” *IEEE Transactions on Software Engineering*, December 1987.

[Basili 92] Basili, V. R., G. Caldiera, F. McGarry, R. Pajerski, G. Page, and S. Waligora, “The Software Engineering Laboratory — An Operational Software Experience Factory,” *Proceedings of the Fourteenth International Conference on Software Engineering*, Melbourne, Australia, May 1992.

[Basili 94] Basili, V. R., and S. Green, “Software Process Evolution at the SEL,” *IEEE Software*, July 1994.

[Bassman 94] Bassman, M., F. McGarry, and R. Pajerski, *Software Measurement Guidebook* (SEL-94-002), Software Engineering Laboratory, July 1994.

[Card 85] Card, D. N., G. Page, and F. E. McGarry, “Criteria for Software Modularization,” *Proceedings of the Eighth International Conference on Software Engineering*, London, 1985.

[Condon 93] Condon, S., M. Regardie, M. Stark, and S. Waligora, *Cost and Schedule Estimation Study Report* (SEL-93-002), Software Engineering Laboratory, November 1993.

[Green 90] Green, S., *The Cleanroom Case Study in the Software Engineering Laboratory: Project Description and Early Analysis* (SEL-90-002), Software Engineering Laboratory, March 1990.

[Heller 92] Heller, G. H., J. Valett, and M. Wild, *Data Collection Procedures for the Software Engineering Laboratory Database* (SEL-92-002), Software Engineering Laboratory, March 1992.

[Hendrick 92] Hendrick, R., D. Kistler, and J. Valett, *Software Management Environment (SME) Concepts and Architecture (Revision 1)* (SEL-89-103), Software Engineering Laboratory, September 1992.

[Landis 90] Landis, L., F. E. McGarry, S. Waligora, et al., *Manager's Handbook for Software Development (Revision 1)* (SEL-84-101), Software Engineering Laboratory, November 1990.

[Landis 92] Landis, L., S. Waligora, F. E. McGarry, et al., *Recommended Approach to Software Development (Revision 3)* (SEL-81-305), Software Engineering Laboratory, June 1992.

[McGarry 94] McGarry, F. E., and M. Thomas, "Top-Down vs. Bottom-Up Process Improvement," *IEEE Software*, July 1994.

[Morusiewicz 93] Morusiewicz, L., and J. Valett, *Annotated Bibliography of Software Engineering Laboratory Literature* (SEL-82-1206), Software Engineering Laboratory, November 1993.

[Page 84] Page, G., F. E. McGarry, and D. Card, "A Practical Experience with Independent Verification and Validation," *Proceedings of the Eighth International Computer Software and Applications Conference*, IEEE Computer Society Press, 1984.

[Paulk 93] Paulk, M., B. Curtis, M. Chrissis, and C. Weber, *Capability Maturity Model for Software, Version 1.1* (CMU/SEI-93-TR-24, ADA 263403). Pittsburgh, Pa: Software Engineering Institute, Carnegie Mellon University, February 1993.

[Seidewitz 86] Seidewitz, E., and M. Stark, *General Object-Oriented Software Development* (SEL-86-002), Software Engineering Laboratory, August 1986.

[Stephens 74] Stephens, W. P., G. J. Meyers, and L. L. Constantine, “Structured Design,” *IBM Systems Journal*, Volume 3, Number 2, 1974.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
None

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release
Distribution Unlimited

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
CMU/SEI-94-TR-22

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ESC-TR-94-022

6a. NAME OF PERFORMING ORGANIZATION
Software Engineering Institute

6b. OFFICE SYMBOL (if applicable)
SEI

7a. NAME OF MONITORING ORGANIZATION
SEI Joint Program Office

6c. ADDRESS (city, state, and zip code)
Carnegie Mellon University
Pittsburgh PA 15213

7b. ADDRESS (city, state, and zip code)
HQ ESC/ENS
5 Eglin Street
Hanscom AFB, MA 01731-2116

8a. NAME OFFUNDING/SPONSORING ORGANIZATION
SEI Joint Program Office

8b. OFFICE SYMBOL (if applicable)
ESC/ENS

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F1962890C0003

8c. ADDRESS (city, state, and zip code))
Carnegie Mellon University
Pittsburgh PA 15213

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.
63756E	N/A	N/A	N/A

11. TITLE (Include Security Classification)
Software Process Improvement in the NASA Software Engineering Laboratory

12. PERSONAL AUTHOR(S)
Frank McGarry

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (year, month, day)	15. PAGE COUNT
Final	FROM TO	December 1994	80

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)
software process improvement, software engineering, Software Engineering Laboratory, software processes, process achievement award

19. ABSTRACT (continue on reverse if necessary and identify by block number)

The Software Engineering Laboratory (SEL) was established in 1976 for the purpose of studying and measuring software processes with the intent of identifying improvements that could be applied to the production of ground support software within the Flight Dynamics Division (FDD) at the National Aeronautics and Space Administration (NASA)/Goddard Space Flight Center (GSFC). The SEL has three member organizations: NASA/GSFC, the University of Maryland, and Computer Sciences Corporation (CSC). The concept of process improvement within the SEL focuses on the continual understanding of both process and product as well as goal-driven experimentation and analysis of process change within a production environment.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ■ SAME AS RPT □ DTIC USERS ■

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL
Thomas R. Miller, Lt Col, USAF

22b. TELEPHONE NUMBER (include area code)
(412) 268-7631

22c. OFFICE SYMBOL
ESC/ENS (SEI)

ABSTRACT — continued from page one, block 19