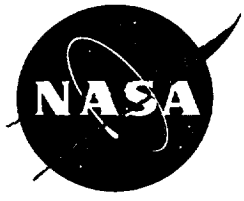
CR-189410

SOFTWARE ENGINEERING LABORATORY SERIES

SEL-94-005

AN OVERVIEW OF THE SOFTWARE ENGINEERING LABORATORY

DECEMBER 1994



(NASA-CR-189410) AN OVERVIEW OF THE SOFTWARE ENGINEERING LABORATORY (NASA. Goddard Space Flight Center) 64 p

N95-28714

Unclas

G3/61 0053149

National Aeronautics and Space Administration

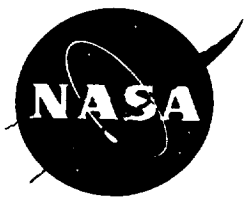
Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES

SEL-94-005

AN OVERVIEW OF THE SOFTWARE ENGINEERING LABORATORY

DECEMBER 1994



National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

Foreword

The **Software Engineering Laboratory** (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created to investigate the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1976 and has three primary organizational members:

NASA/GSFC, Software Engineering Branch

University of Maryland, Department of Computer Science

Computer Sciences Corporation, Software Engineering Operation

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on the process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

The major contributors to this document are

Frank McGarry and **Rose Pajerski**, NASA/Goddard Space Flight Center

Gerald Page and **Sharon Waligora**, Computer Sciences Corporation

Victor Basili and **Marvin Zelkowitz**, University of Maryland

The SEL is accessible on the World Wide Web at

http://groucho.gsfc.nasa.gov/Code_550/SEL_hp.html

Single copies of this document can be obtained by writing to

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

Abstract

This report describes the background and structure of the SEL organization, the SEL process improvement approach, and its 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.

Contents

Illustrations

Figures

Tables

Introduction

Since its inception, the Software Engineering Laboratory (SEL) has conducted experiments on approximately 120 Flight Dynamics Division (FDD) production software projects at NASA/Goddard Space Flight Center (GSFC), 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. This effort has been driven throughout by the goals of achieving significant overall improvement in three product measures:

- Reduction in **defect rate** of delivered software
- Reduction in **cost** of software to support similar missions
- Reduction in average **cycle time** to produce mission support software

The continual experimentation with software process has yielded an extensive set of empirical studies that has guided the evolution of standards, management practices, technologies, and training within the organization. Additionally, the SEL has produced over 200 reports that describe experiences from the experimentation process and its overall software process improvement approach.

Section 1. Background

1.1 SEL History

The SEL was created in 1976 at NASA/GSFC for the purpose of understanding and improving the overall software process and products that were being created within the 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 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.

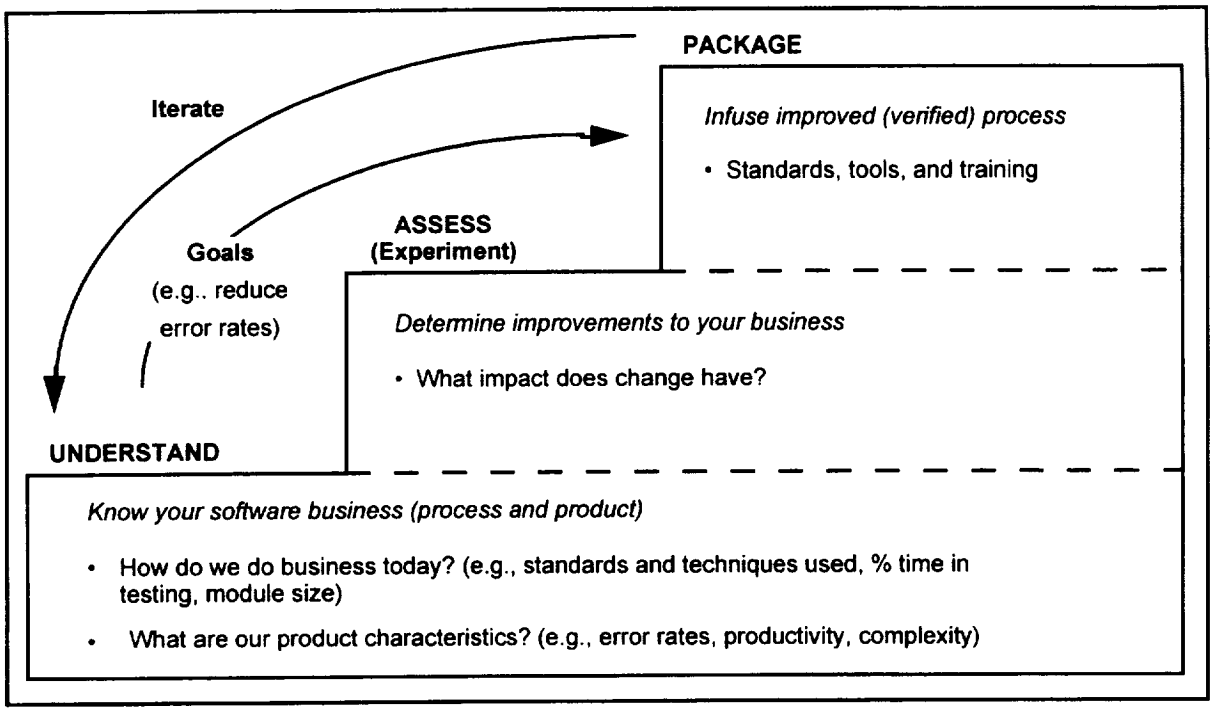


Figure 1. The SEL Process Improvement Paradigm

This includes the development and enhancement of standards, tools, and training.

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. A unique strength of the SEL's process improvement approach is that it was developed and has evolved based on scientific method. Over the years, its key concepts, briefly described below, have been captured and formalized in the open literature:

- Process evolution: the Quality Improvement Paradigm (Reference 1)
- Measurement and control: the Goal/ Question/Metric method (Reference 2)
- Structure and organization: the Experience Factory (Reference 3)

The **Quality Improvement Paradigm** is a two-loop feedback process (project and organization loops) that is a variation of the scientific method. It consists of the following steps:

- Characterization: Understand the environment based upon available models, data, intuition, etc., so that similarities among projects can be recognized.
- Planning: Based on this characterization, set quantifiable goals for successful project and organization performance and improvement and choose the appropriate processes and supporting methods and tools to achieve the improvement goals in the given environment.

- Execution: Construct the products using the selected processes and provide real-time project feedback based on the goal achievement data.
- Packaging: At the end of each specific project, analyze the data and the information gathered to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements. Then, package the experience gained in the form of updated and refined models and other forms of structured knowledge based on this and prior project experience. Finally, store the packages in an experience base so they are available for future use.

The **Goal/Question/Metric** (GQM) method is used to define measurement on the software project, process, and product in such a way that

- Resulting metrics are tailored to the organization and its goal.
- Resulting measurement data play a constructive and instructive role in the organization.
- Metrics and their interpretation reflect the values and the viewpoints of the different groups affected (e.g., developers, users, operators).

GQM defines a measurement model on three levels:

- Conceptual level (goal): A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, and relative to a particular environment.
- Operational level (question): A set of questions is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal.
- Quantitative level (metric): A set of metrics, based on the models, is associated with every question in order to answer it in a measurable way.

Although originally used to define and evaluate a particular project in a particular environment, GQM can also be used for control and improvement of a single project within an organization running several projects (References 4 and 5).

The **Experience Factory** organizational concept was introduced to institutionalize the collective learning of the organization that is at the root of continual improvement and competitive advantage. It establishes a separate organizational element that supports reuse of experience and collective learning by developing, updating, and delivering experience packages to the project organization which is responsible for developing and maintaining software. This structure creates a symbiotic relationship where the

- Project organization offers to the experience factory its products, the plans used in its development, and the data gathered during development and operation.
- Experience packagers transform these objects into reusable units and supply them to the project organization, together with specific support that includes monitoring and consulting.

As an operational experience factory, the SEL has been facilitating software process improvement within the FDD at NASA/GSFC for 18 years (Reference 6). All SEL experiments have been conducted in this production environment, which consists of approximately 250 engineers developing and maintaining systems that 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 developers and analysts.

The SEL's pioneering work in the practical application of software process improvement concepts in the FDD has been recognized throughout the software engineering community. In 1994, the SEL was chosen as the inaugural recipient of the IEEE Computer Society Award for Software Process Achievement. This award recognizes not only process achievement, but leadership in the field and outstanding contribution to the state-of-the-practice in software engineering. The SEL has been in continuous operation since 1976, and will continue to operate as long as process and product improvement remain a priority within its software domain.

Section 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 three functional components, not necessarily across organizational lines. The three functional areas are

- Software development/maintenance
- Process/product analysis
- Data base support

The three components (developers, process analysts, and data base 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 chapters that follow. Figure 2 provides a graphic overview of their function and size, and Table 1 depicts the difference in focus among the three groups.

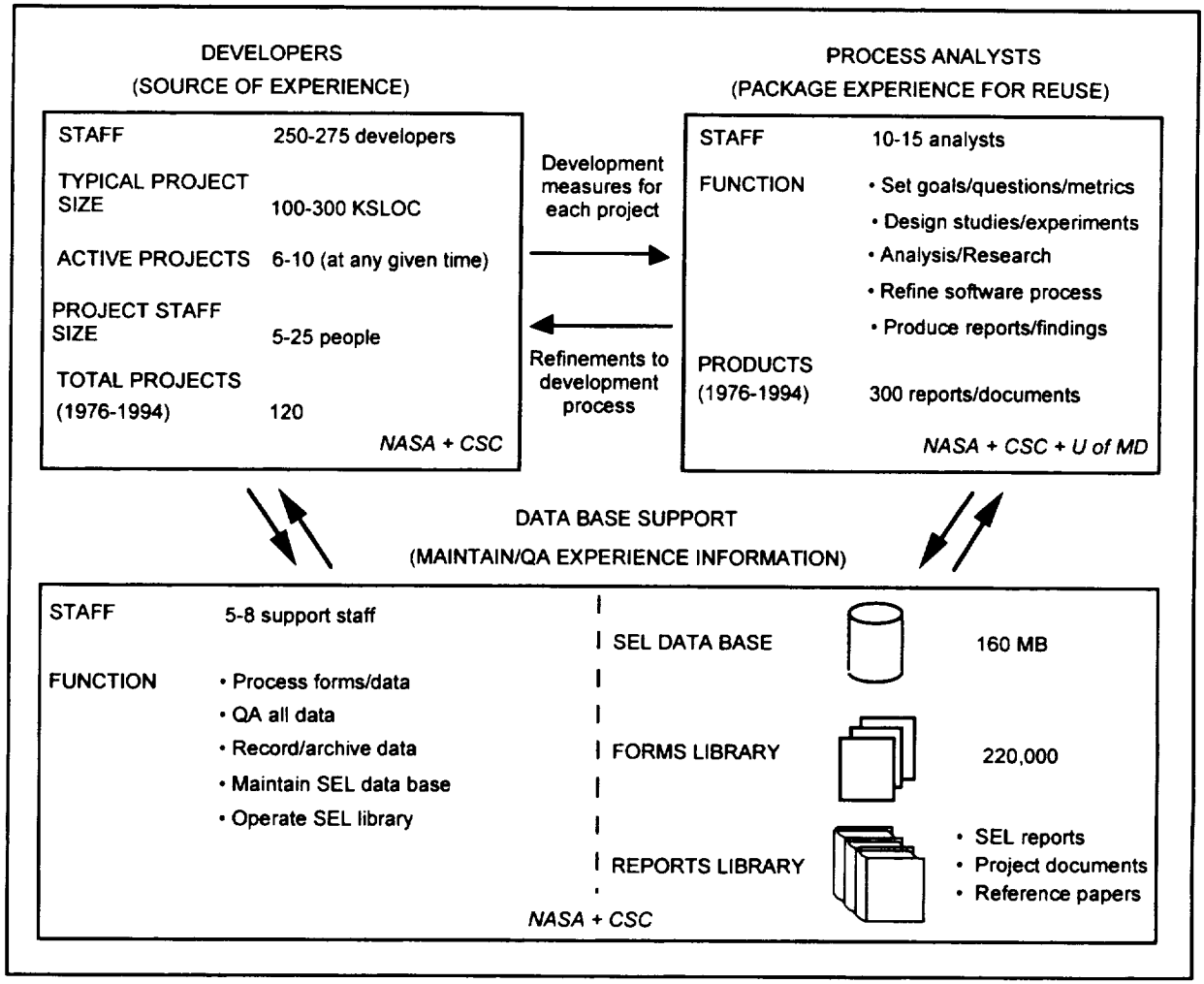


Figure 2. SEL Structure

Table 1. Focus of SEL Organizational Components

	DEVELOPERS	PROCESS ANALYSTS	DATA BASE 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 • Experiment with new techniques with the analysts' support	• Assess the impact of specific technologies • Produce models, standards, and training materials	• Maintain a library of experiences, models, and standards
Measure of success	• Validation and verification of software products	• Packaging and reuse of empirical software experience • Improved software products	• Efficient processes for information retrieval (data, models, reports)

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 non-real time, non-embedded, ground-based applications, and there are usually four or five projects in development at any one time. Traditionally, most of the software has been written in FORTRAN, although the organization is currently evolving to using C, C++, and Ada for new systems. 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 packaging 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 three 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 Data Base Support

The third function within the SEL is the data processing and archiving of the projects' experiences in the SEL's measurement data base. This is supported by approximately three full-time people at NASA/GSFC and approximately five full-time people at CSC. The data base 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 data base; 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 data base, 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 data base 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.

Section 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, which is formalized in the Experience Factory model, can be described as a "bottom-up" software improvement approach (Reference 7), where the process is defined and improved based on corporate knowledge that is extracted from the experiences of projects at the lowest level or bottom of the organization. The SEL approach 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 provide guidance 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) (Reference 8), developed by the Software Engineering Institute (SEI).

The SEL approach assumes that changes must be driven from the bottom up, 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 an organization would 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.

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 data base 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 information and data on which all process change is based. Process/product analysts and data base 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/data base 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.

Section 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 data base support staff who add it to the data base for access by the analysts conducting the experiment. These SEL activities are described in the sections that follow.

4.1 Defining Experiments

Based on organizational goals and process weaknesses identified in the understanding step, SEL analysts identify software process modifications that they hypothesize are likely to improve the resultant product. To do this, analysts review literature looking for candidate new technologies that address the particular needs of their organization. In cases where the candidate technologies are closer to the state-of-the-art than the state-of-the-practice, university studies are conducted on test beds before an experiment is undertaken in the production environment. Analysts also consult developers who have insight into the problem area and who may suggest promising process changes to pursue.

For each process modification selected, the analysts 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 data base support group so that project names, subsystem names, personnel participating, and forms expected can be logged, and the data base 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 was provided to the first development team using it in this environment (Reference 9).

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 3 and 4 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* (Reference 10). The full set of data collection forms and procedures can be found in the *Data Collection Procedures for the SEL Database* (Reference 11).

As the developers/maintainers complete the forms, they submit them to the data base 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. Data base support staff then enter the data in a central data base and perform a second quality-assurance step by checking for data entry errors by comparing the data base 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 information automatically such as source code characteristics (e.g., size, amount of reuse, complexity, module characteristics) or changes and growth of source code during development. Data base support personnel execute the tools to gather these additional measures, which are then entered in the SEL data base.

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 Reference 10 for details on these measures.)

Figure 5 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.

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.

Personnel Resources Form

Name: _____

Project: _____ Date: (Friday): _____

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 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, or code, or documentation. These are not caused by required changes or errors in the system. ☐

Documenting: Hours spent on any documentation on the system. 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. 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: _____

5150g(21)-38

Figure 3. 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_____

	month	day	year
Need for change determined on:			
Change completed (incorporated into system):			

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

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

Section B – All Changes

Type of Change (Check one)

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

Effects of Change

Y	N	
☐	☐	Was the change or correction to one and only one component? (Must match Effect in Section A)
☐	☐	Did you look at any other component? (Must match Effort in Section A)
☐	☐	Did you have to be aware of parameters passed explicitly or implicitly (e.g., COMMON blocks) to or from the changed components?

Section C – For Error Corrections Only

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 (Check Y or N for all)

Y	N	
☐	☐	Omission error (e.g., something was left out)
☐	☐	Commission error (e.g., something incorrect was included)
☐	☐	Error was created by transcription (clerical)

For Librarian's Use Only

Number:_____ _____

Date:_____ _____

Entered by:_____ _____

Checked by:_____ _____

6201g(13)-09

NOVEMBER 1991

Figure 4. Defect/Change Data Collection Form

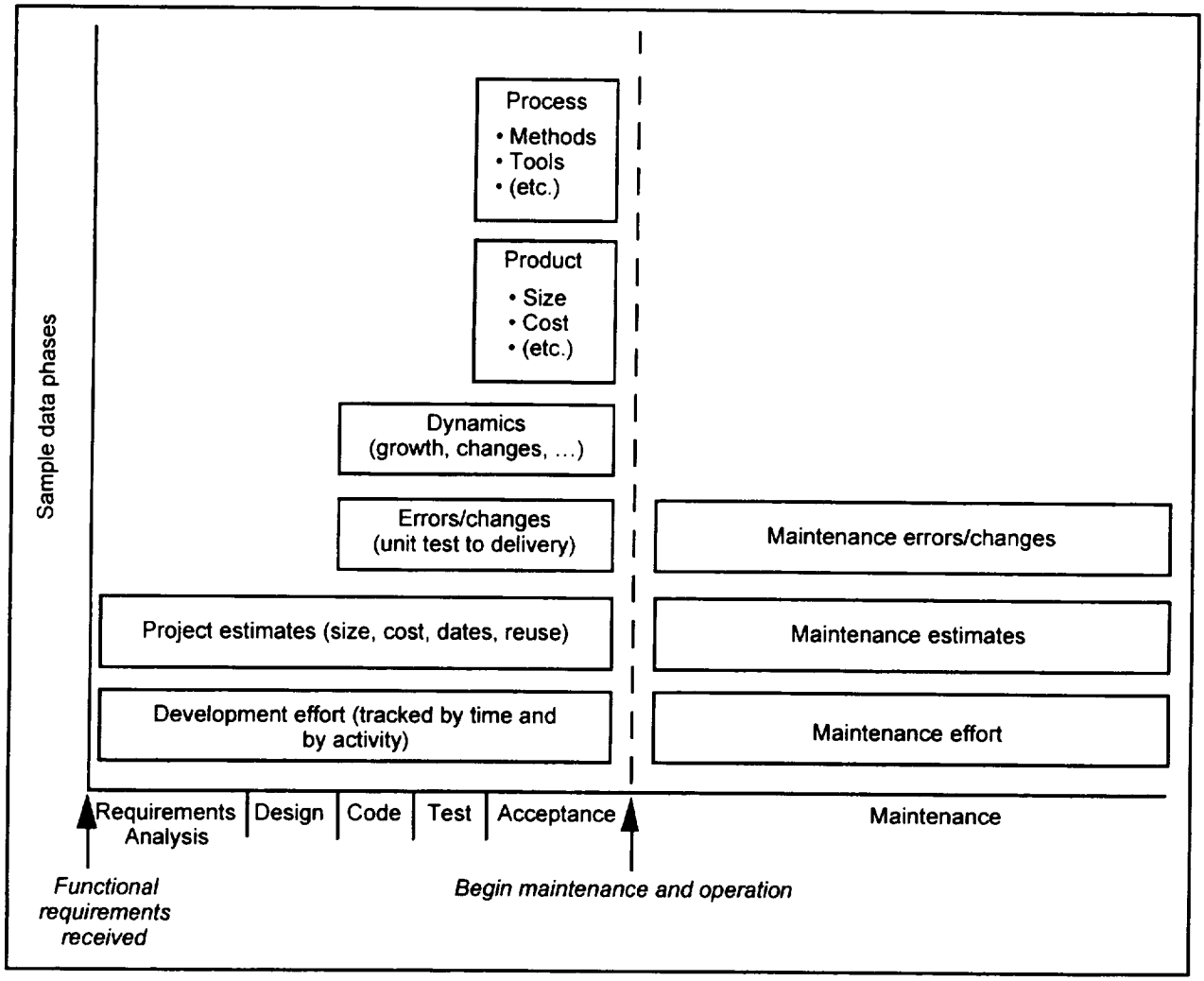


Figure 5. SEL Core Measures

Having learned from its extensive data analysis experience over the years, the SEL now follows these key rules (Reference 10):

- 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 must 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 data base 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, 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 of the extensive amount of analysis and reporting. 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 management, design, implementation, testing,

and the inspection process, it may turn out that only the implementation and testing 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. 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.)

Section 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 thousand source lines of code (KSLOC) to 1.5 million SLOC, with the majority falling in the 100–250 KSLOC range. All of the information extracted from these development and maintenance projects is stored in the SEL data base and used by the analysts who study the projects and produce reports, updated standards, tools, 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

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, based on which change and meaningful comparison can be made.

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. Table 2 lists 11 projects active between 1985 and 1990 that were included in the early SEL baseline.

Table 2. SEL Baseline (1985-1990)

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

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 3 for a sample effort data collection form.)

Figure 6 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 25 percent 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 this value 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 6, 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.

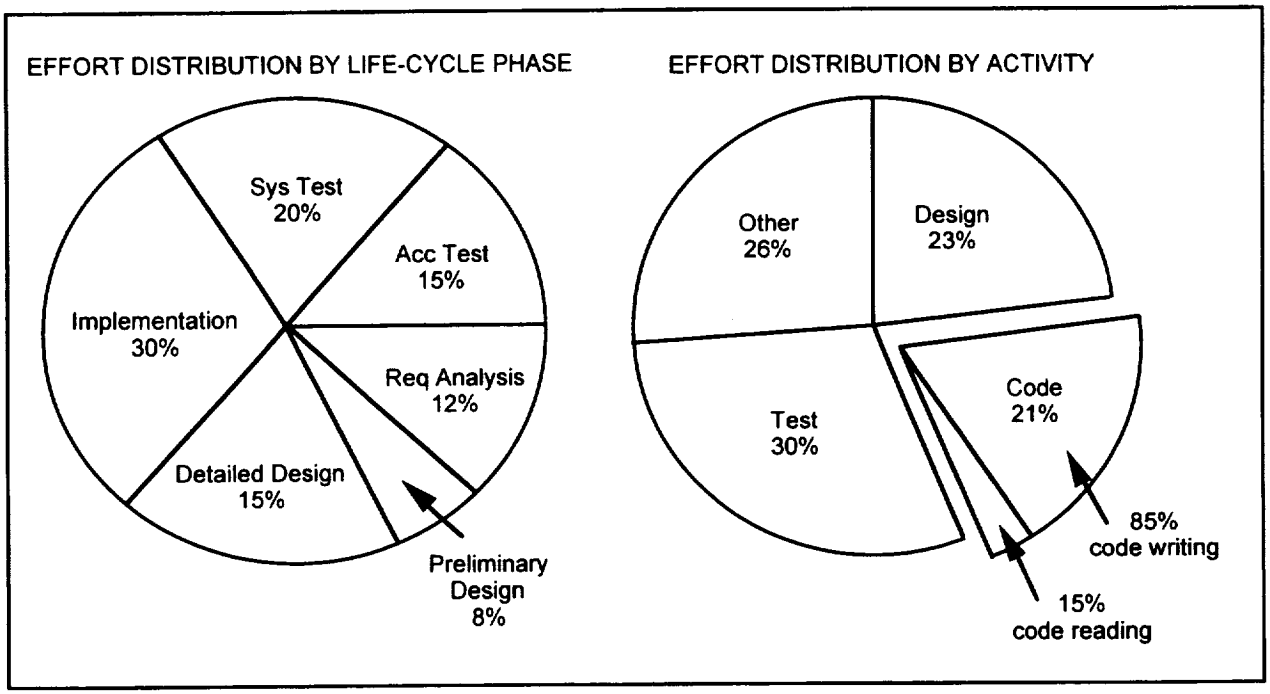


Figure 6. Effort Distribution by Phase and Activity

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.
- Initialization errors—improper settings of the initial value of variables.
- Logic/control errors—errors in flow control in a program, such as incorrect branches as the result of evaluating an if-statement 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.
- 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 continually understands 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 7.

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 8). More recent studies show that the step function is still present, although the error rates have decreased significantly.

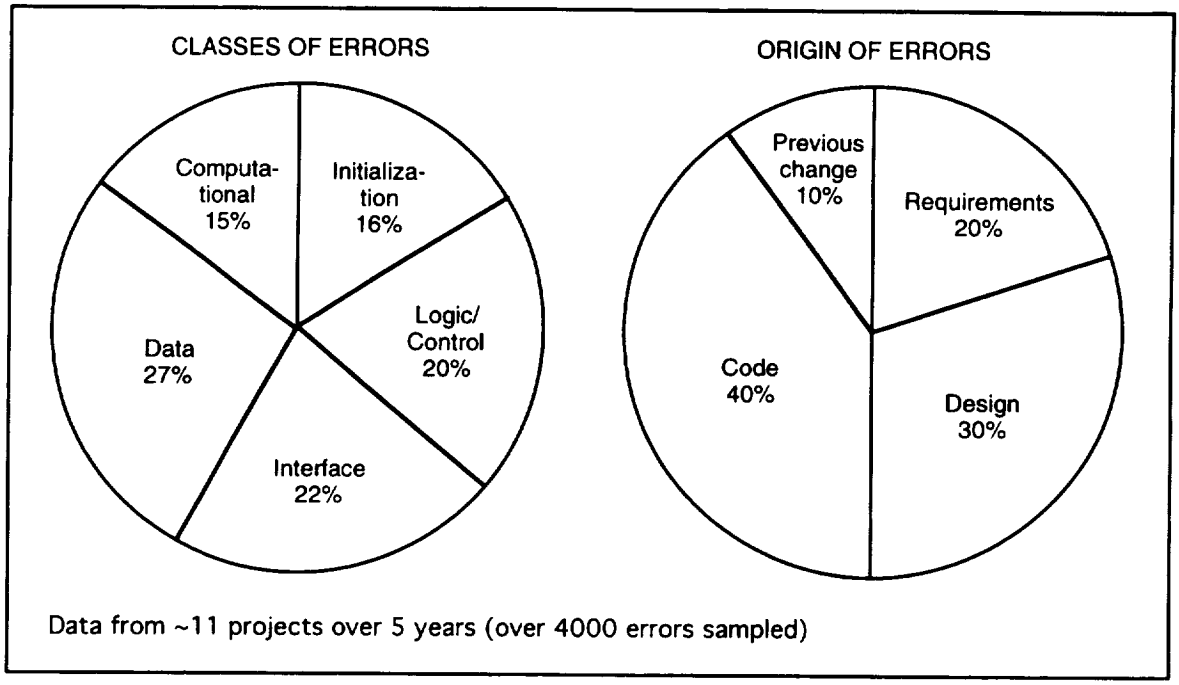


Figure 7. SEL Error Characteristics

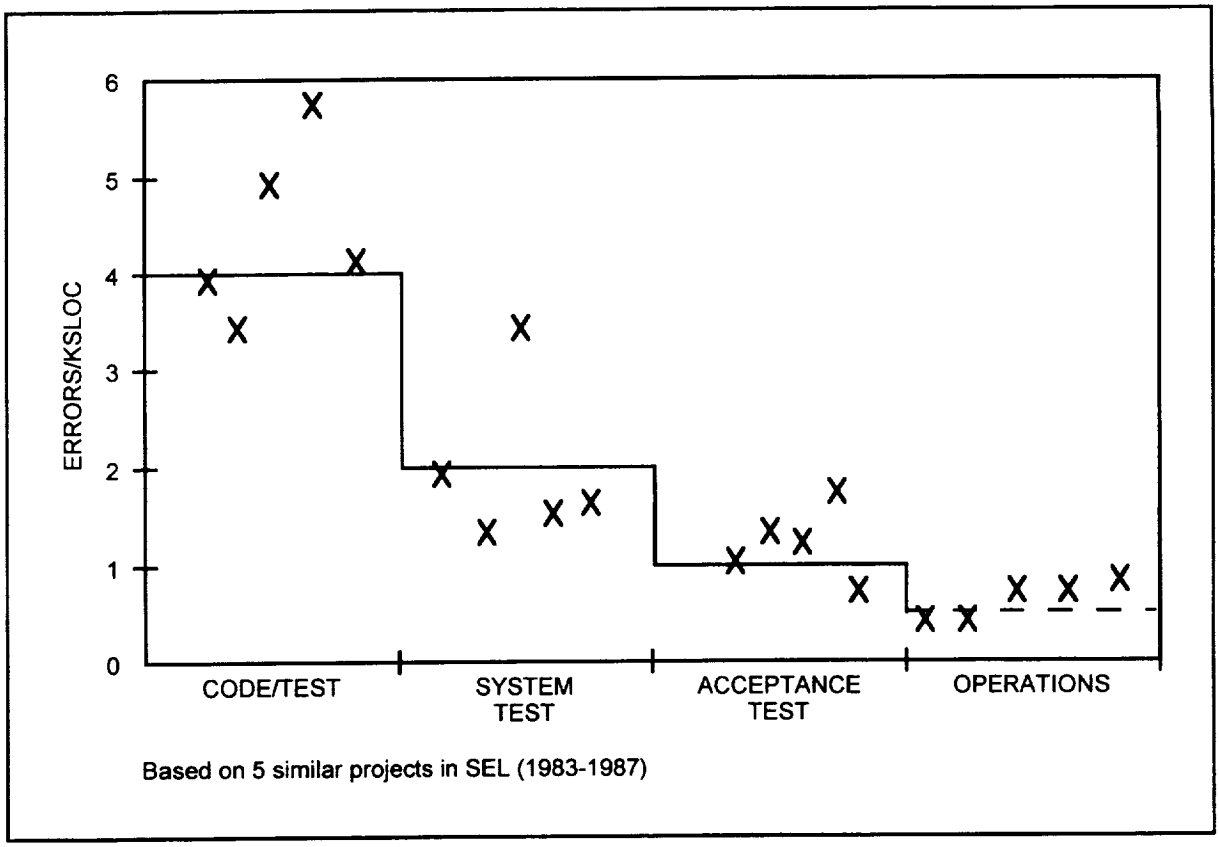


Figure 8. Error Detection Rate Model

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 leading 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 Table 3). 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. Table 4 describes the characteristics of another set of software projects active during the late 1980s and early 1990s. The differences between this and the earlier models/relations are surprisingly small, but there is change.

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.

Table 3. Initial SEL Models/Relations

Productivity	code rate = 26 new lines per day	
Effort distribution	**Date**	**Activity**
Design	26%	23%
Code	38%	21%
Test	36%	30%
Other		26%
Pages of documentation	doc = 34.7 (KSLOC.93)	
Maintenance cost	~12% development cost per year	
Reuse cost		
FORTRAN	20% of new	
Ada	30% of new	
Software size estimate growth		40%

Source: *SEL Relationship, Models and Management Rules*, 1991

Table 4. More Recent SEL Software Characteristics (late 1980s)

Productivity		
FORTRAN	code rate = 26 new lines per day	
Ada	code rate = 36 new lines per day	
Effort distribution	**Date**	**Activity**
Low reuse		
Design	24%	21%
Code	45%	26%
Test	31%	25%
Other		28%
*High reuse**		
Design	26%	17%
Code	38%	17%
Test	36%	32%
Other		24%
Reuse cost		
FORTRAN	20% of new	
Ada	30% of new	
Software size estimate growth		
Low reuse	40%	
High reuse	20%	

*High reuse = >70% reuse

Source: *Cost and Schedule Estimation Study Report*, 1993

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.

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 (Reference 12) 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 International Conference on Software Engineering (Reference 13).

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 ($\leq$ 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 9) 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 module size has little effect on error ratio.

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.

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 three 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 three programs using a different testing technique on each program.

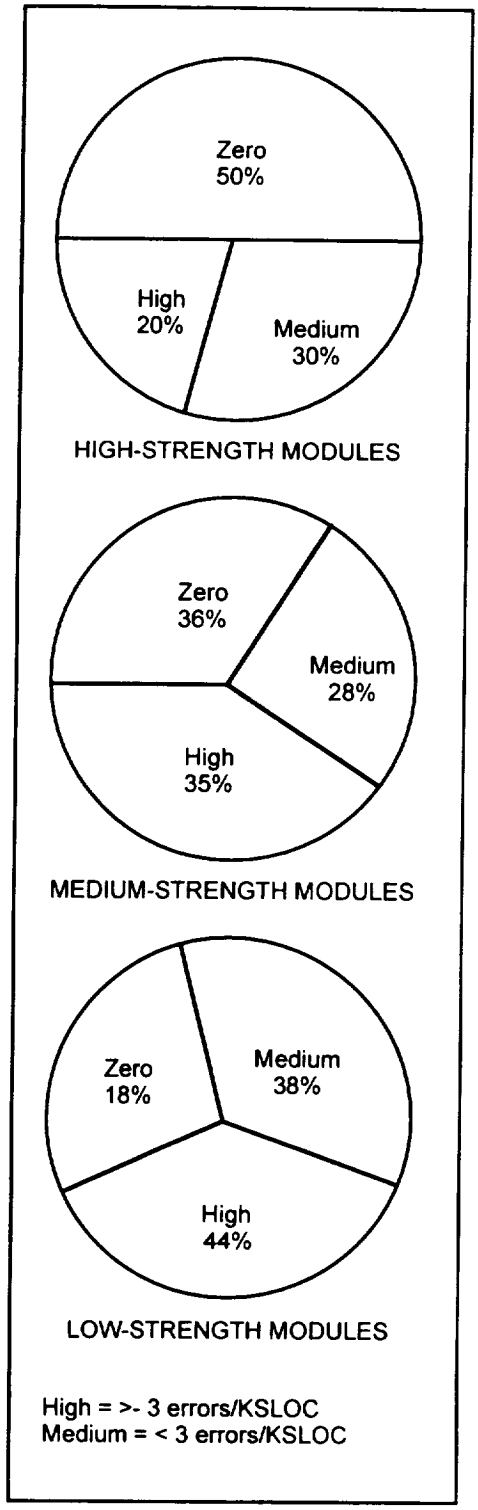


Figure 9. Fault Rate for Classes of Module Strength

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 10).
- Code reading was more cost effective than either functional testing or structural testing in number of errors found per unit of time (See Figure 11). Structural testing and functional testing had about the same costs.

The study also produced some interesting results concerning programmer expertise and the discovery of faults. Space does not permit a full explanation here (see Reference 14 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.

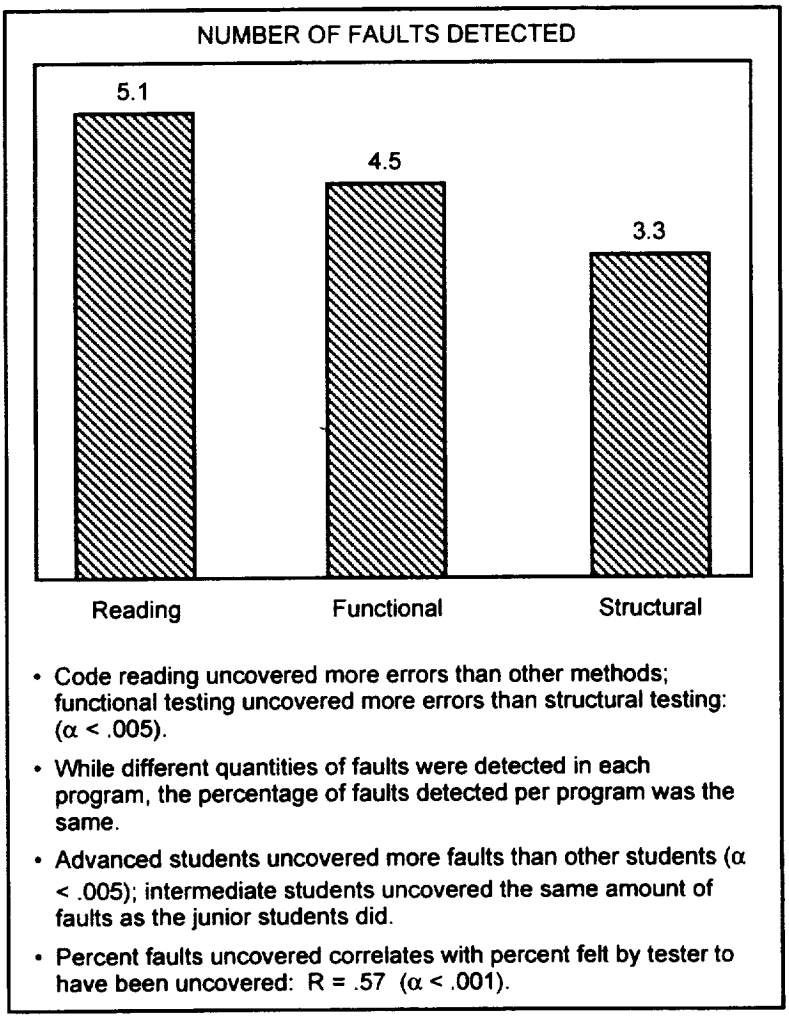


Figure 10. Fault Detection Rate by Testing Method

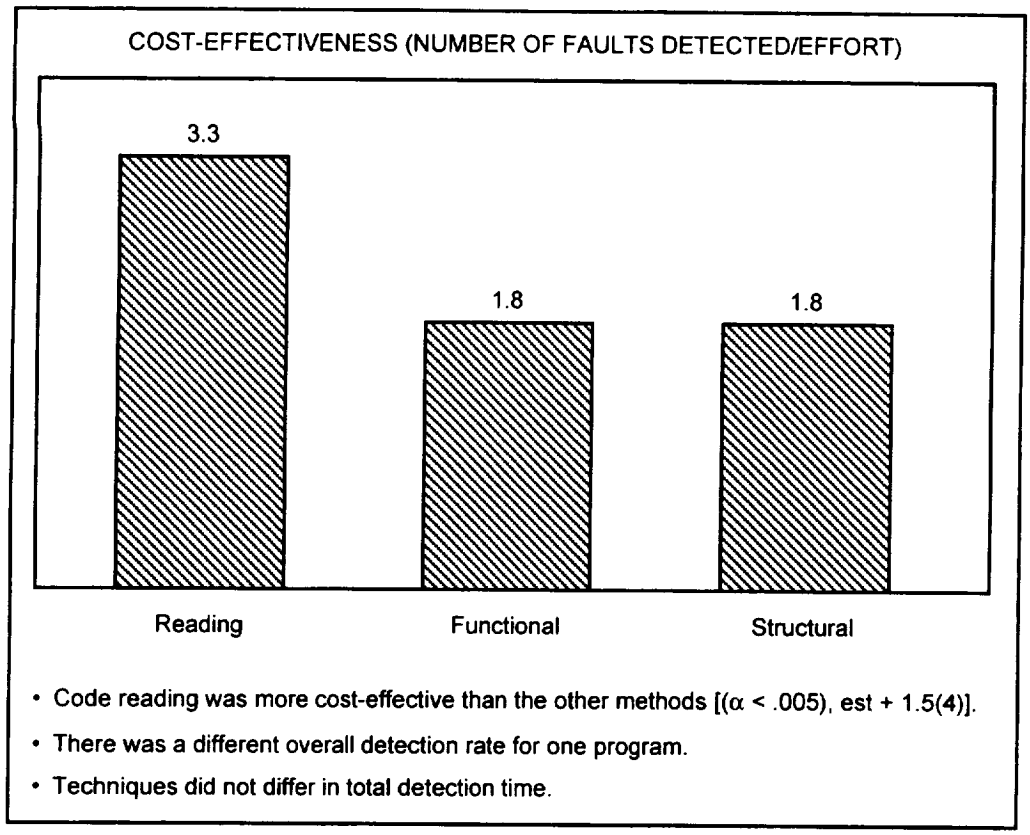


Figure 11. Cost of Fault Detection by Testing Method

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 SEL pursued it because results of the testing study and an earlier environment/tools study pointed to techniques that strengthen discipline as high-leverage candidates.

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.
- Function understanding and verification 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 was 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 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 12). Design time and code reading grew significantly, while code writing and testing times all dropped. Defect rates improved (Figure 13) although productivity remained about the same using this new technology. The results of these studies are reported in more detail in Reference 15.

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.

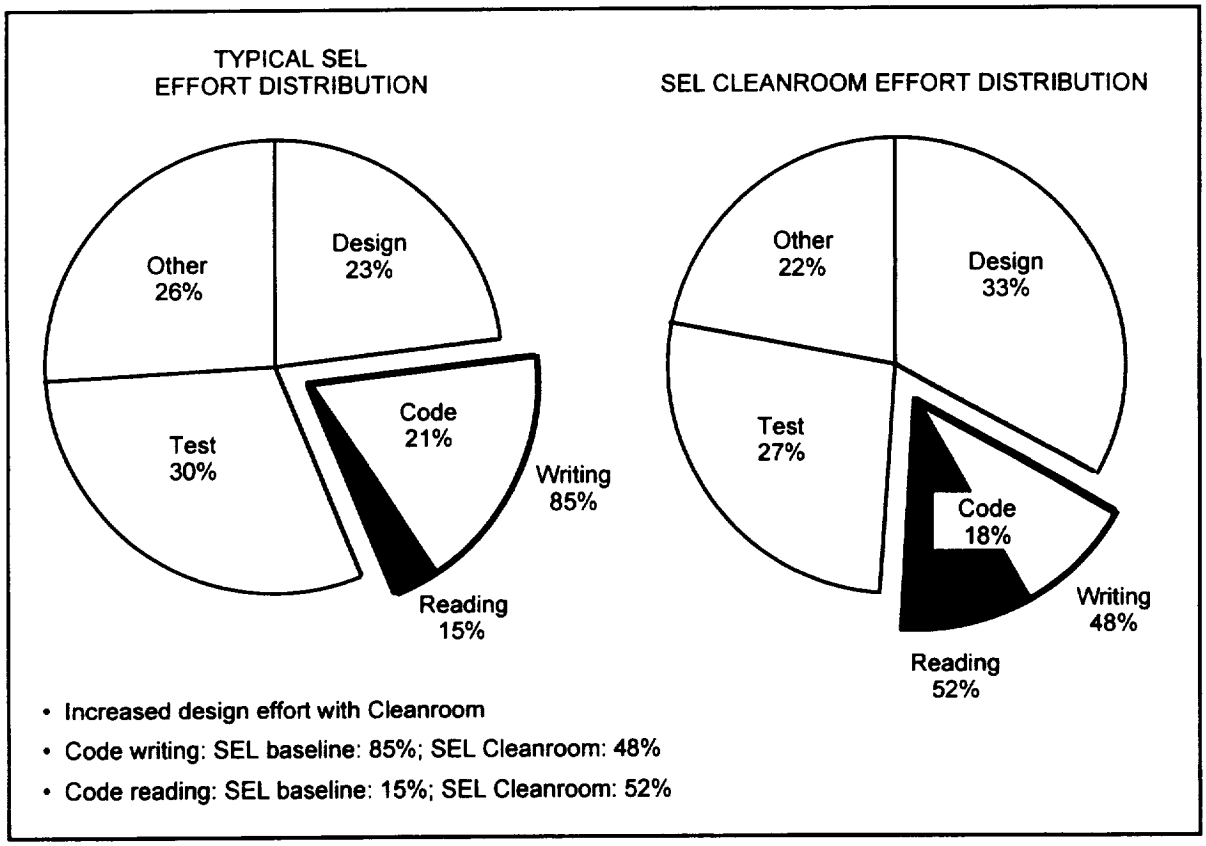


Figure 12. Results of Cleanroom Experiment

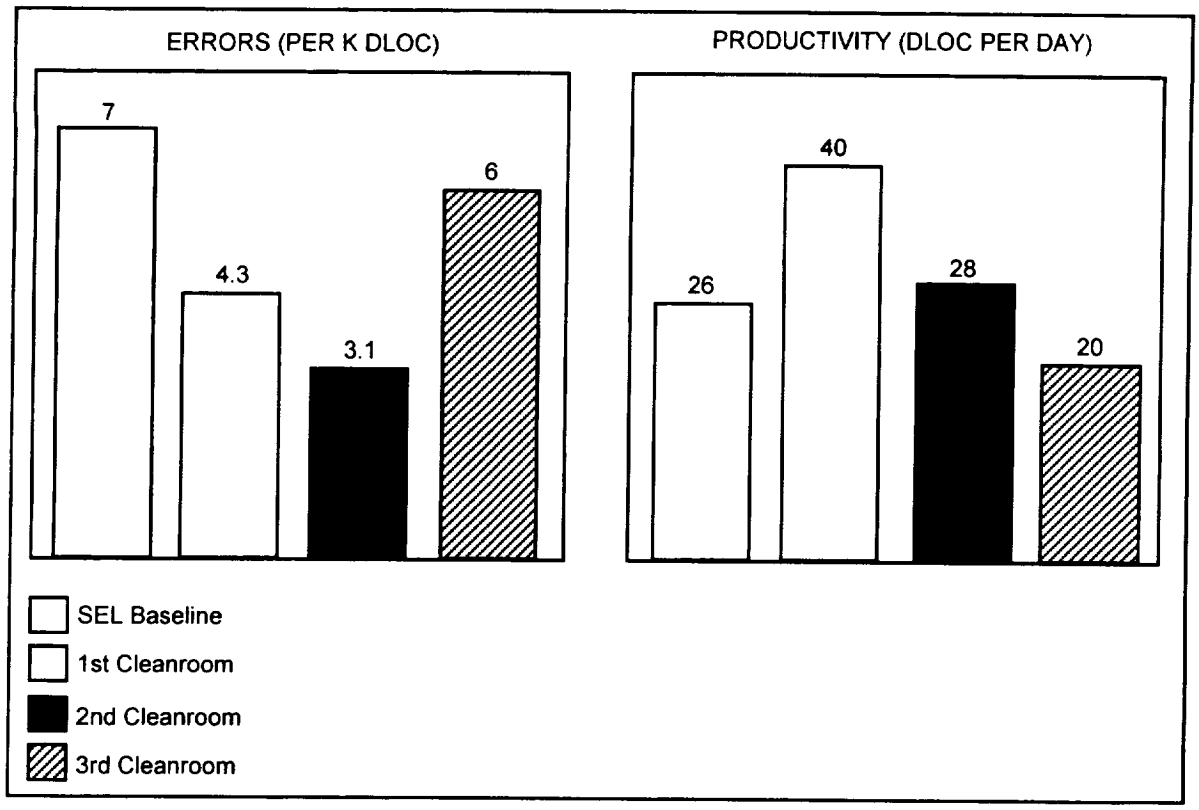


Figure 13. Assessing Cleanroom Against Goals and Expectations

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 14). As the SEL learned about Ada, and the programming staff became more familiar with the features of the language, 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 (Figure 15).

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.

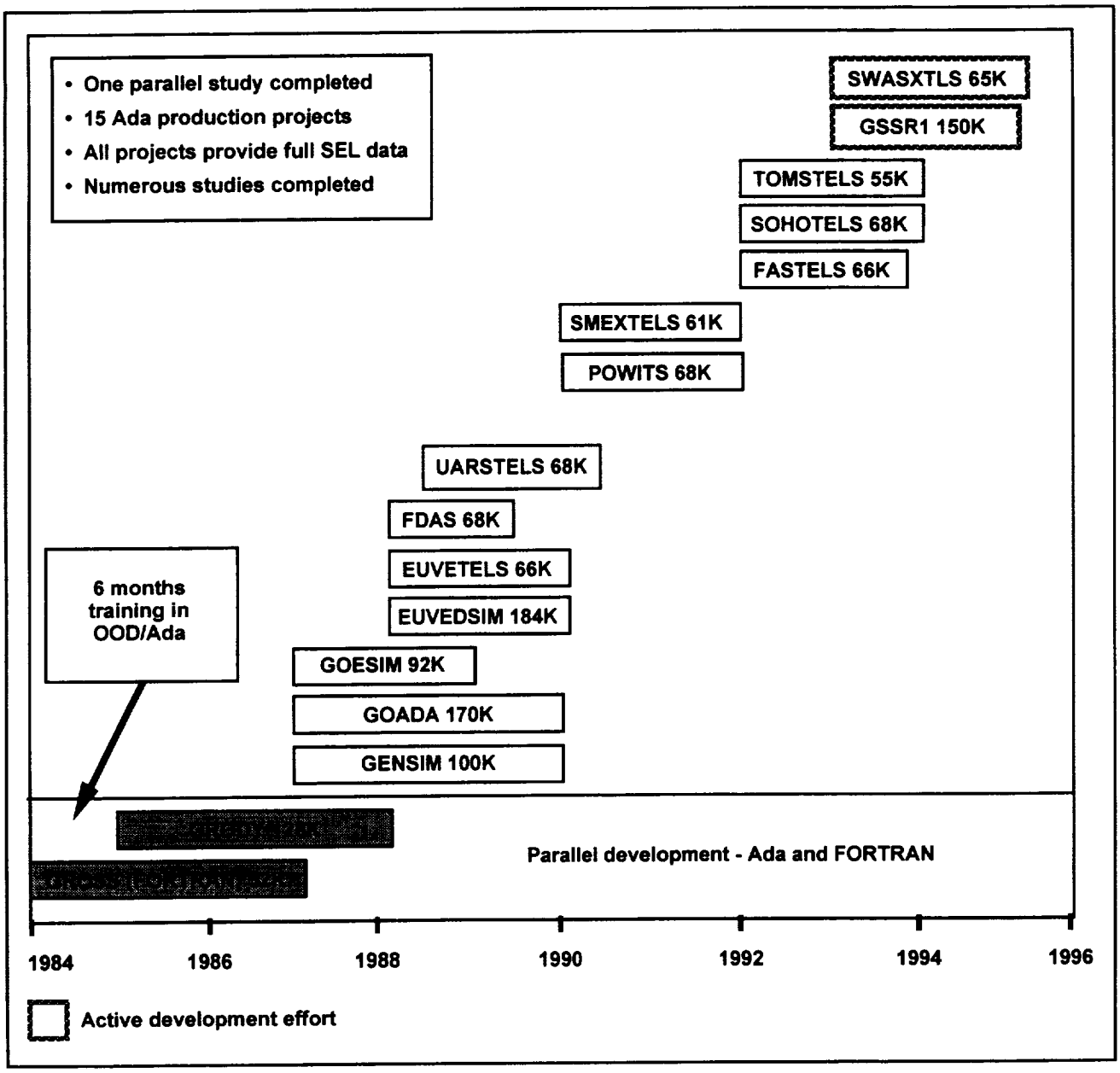


Figure 14. SEL Ada/OOT Projects

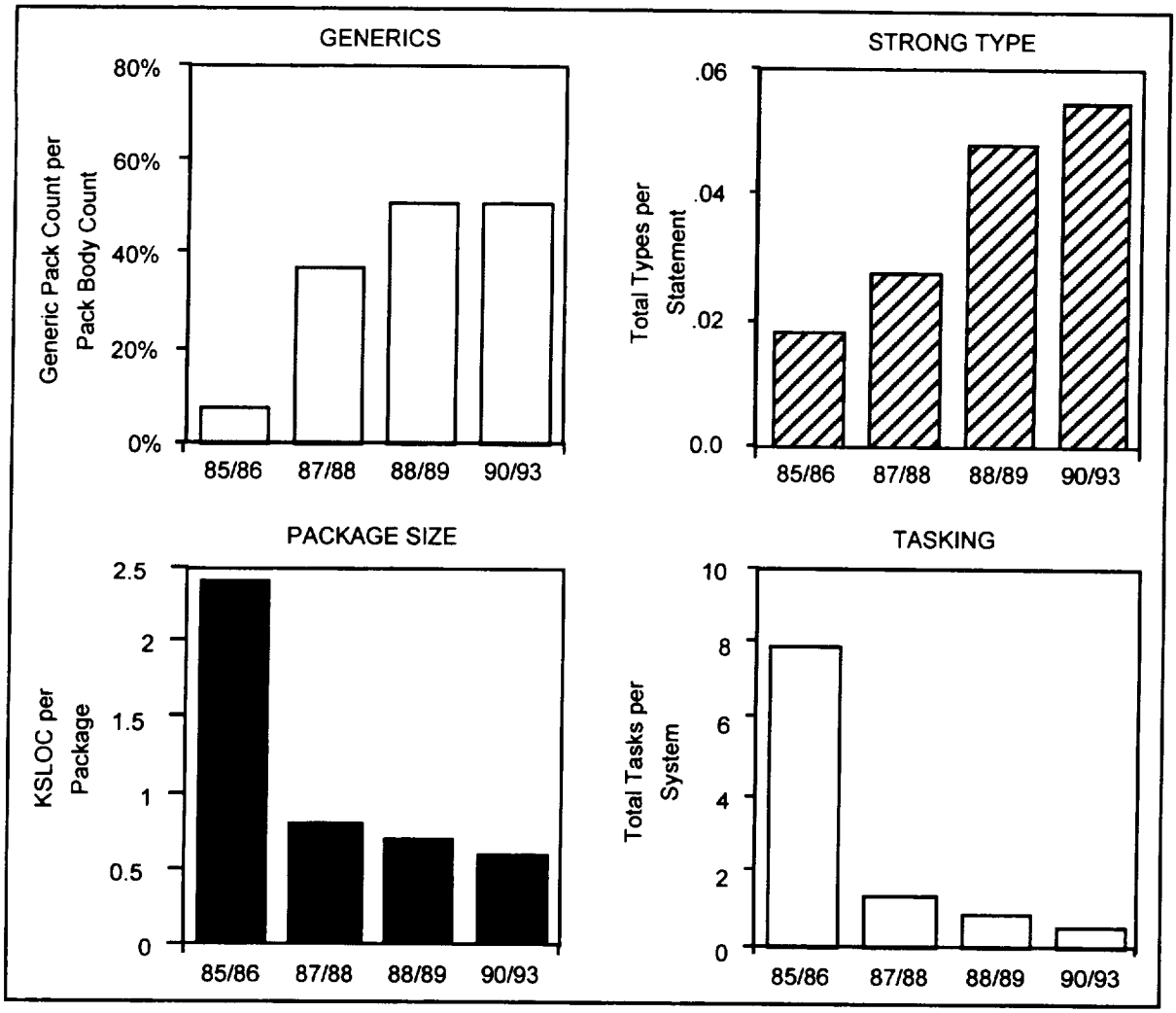


Figure 15. Maturing Use of Ada

Some of the attributes in Figure 15 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* (Reference 16) 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 16 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.

5.2.5 Studies with Independent Verification and Validation (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.

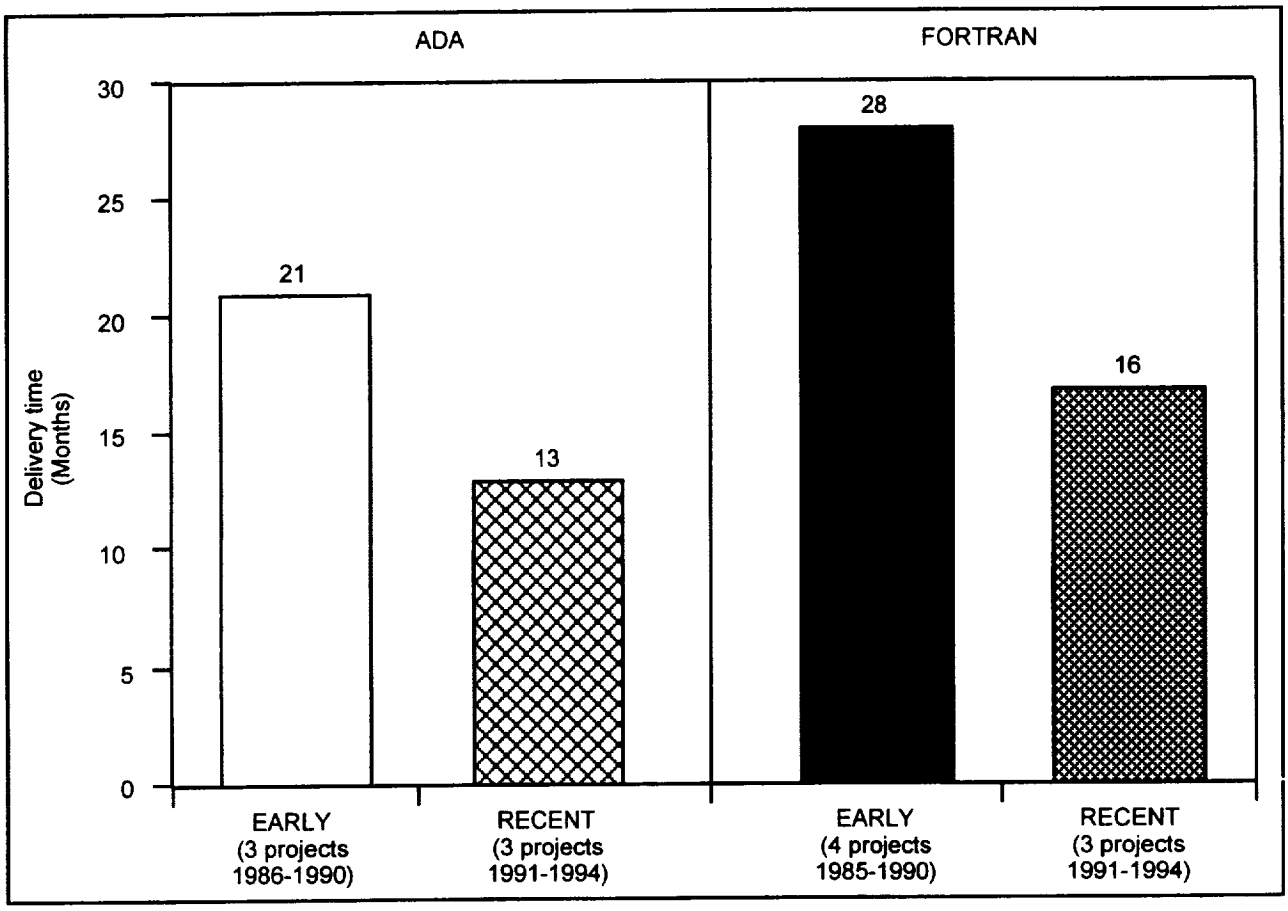


Figure 16. Reuse Shortened Project Duration

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 Reference 17.) 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 three full-time programmers and each project took approximately 16 months from design through acceptance. The initial expectations for these projects were

- Earlier discovery of defects and increased 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 17 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.

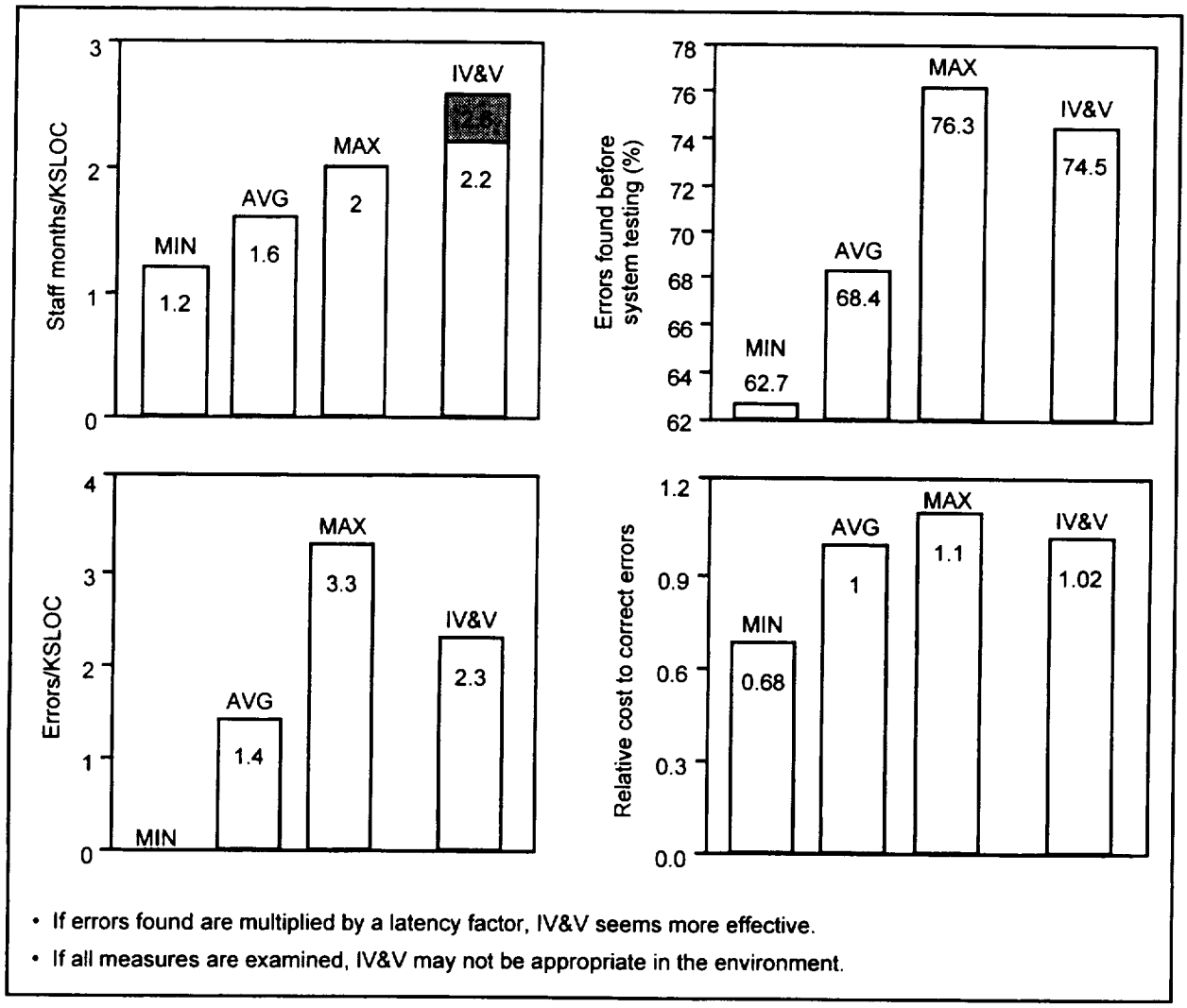


Figure 17. A Look at IV&V Methodology

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* (Reference 18).

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 "experience packages" which the SEL uses within the local development organization, and also shares with outside organizations. The primary products of the packaging step are standards, tools, and training that give practical guidance on how to apply the new techniques in the context of the local process. Here, 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. Additionally, the SEL produces interim packages that are used during experimentation while tailoring of the subject technology is being refined for local use.

5.3.1 Interim Packages

Often when the SEL is experimenting with a major software engineering technology that affects a large part of the life-cycle, multiple experiments must be conducted. During these experiments the technology is tailored iteratively to determine its most effective use in the local environment. In these cases, experience from the completed experiments is distilled to produce a custom-tailored process for the next experiment. For example, the results of the initial experiment with the Cleanroom methodology led to the generation of the *Software Engineering Laboratory Cleanroom Process Model* (Reference 19), because the technology radically affected the project organization and the distribution of life-cycle activities. This process was applied on subsequent Cleanroom experiments, and became a standard after successful use.

Sometimes interim packages fill a gap when a technology has not matured sufficiently for direct application locally. For example, when the SEL could not find an object-oriented approach that addressed the full life cycle, SEL analysts developed the *General Object-Oriented Development (GOOD) Methodology* (Reference 16) for use on the early Ada experiments. They also developed an Ada style guide to augment industry standards. Typically, interim packages are integrated into the next release of the baseline standards once their effectiveness is confirmed on a successful experiment. In some cases, however, the interim packages are dropped after experimentation because an acceptable industry-wide standard becomes available, as was the case with the *Ada Style Guide.*

5.3.2 Technology Reports

For each study conducted, the SEL analysts generate a technology report of results and conclusions. The reports may be papers for professional conferences, internal reports, or technical reports. Typically, a final technology assessment report is produced at the end of the experimentation phase, summarizing the SEL's experience with a particular technology. These reports have two purposes: first, to archive the experience and, secondly, to share the SEL's experience with other organizations. These publications are available to the public at no charge and are used as the foundation for extending studies within the SEL. See Reference 18 for a complete list of SEL-published and SEL-related literature.

In addition 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, 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 has packaged its process improvement experience (methods) in the form of guidebooks, such as the *Software Measurement*

Guidebook, that are designed to be used outside, as well as inside, the SEL.

To facilitate the sharing of software engineering experiences among practitioners, 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 regularly presents its latest advances in software process improvement methods and results from its ongoing experiments at this conference, which has been rated as the best conference for software practitioners.

5.3.3 Standards, Tools, and Training

Although the technology 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, tools, and training classes.

Standards

The SEL development organization uses a standard set of policies that is updated on a periodic basis to reflect new experimentation results. It comprises a set of guidebooks that describe the SEL's baseline methodology and several guidebooks that define major tailoring instances of the baseline process.

Baseline Standards:

- *Manager's Handbook for Software Development* (Reference 20)—presents the process that the managers use on flight dynamics systems. This handbook contains the models, guidelines, and acceptable processes expected to be applied on each of the development efforts. It provides specific guidance for using planning and performance models to successfully manage software engineering projects.
- *Recommended Approach to Software Development* (Reference 21)—presents guidelines and standards for developing software in the flight dynamics environment. It is intended for developers and technical managers of software development projects. It describes methods and practices for each phase of a software development life cycle including key activities, products, measures, methods, and tools.
- *Operational Software Maintenance Procedures*—presents the procedures for correcting, adapting, and enhancing operational flight dynamics software.
- *Cost and Schedule Estimation Study Report* (Reference 22)—presents planning models for cost and schedule estimation and the analysis of empirical data on which they are based. The planning parameters are built into planning spreadsheet tools for use by project managers and are updated yearly based on ongoing analysis.
- *Data Collection Procedures for the SEL Database* (Reference 11)—presents the detailed procedures and mechanisms for collecting software measurements. It contains instructions to the developers regarding the content, frequency, and format of the data to be provided.

Tailored Standards:

- *Ada Developer's Supplement to the Recommended Approach*—presents a collection of guidelines for programmers and managers who are developing flight dynamics software in Ada. It is intended to be used in conjunction with the *Recommended Approach to Software Development*. It provides additional detail on topics such as reuse and object-oriented analysis and design.
- *C Style Guide*—presents the recommended practices and style for programmers using the C language in the flight dynamics environment. The guidelines are based on generally recommended software engineering techniques, industry resources, and local convention. It offers preferred solutions to C programming issues and illustrates through examples of C code.
- *Cleanroom Process Model*—presents guidelines for using the Cleanroom methodology in the flight dynamics environment. It describes the Cleanroom life-cycle model and the specific activities performed in each life-cycle phase. It also addresses pertinent managerial issues and highlights the key differences and similarities of

the SEL Cleanroom process and the standard development approach.

The SEL has evolved its approach to standards over the years. The SEL has found that the baseline process is best presented at a medium level of detail; it is more important to communicate the rationale and guidance for applying the methods on projects rather than providing detailed procedures for them. This allows the detailed procedures to evolve as improvements are made and specific project needs change, without requiring waivers or continual updates to the formal standards. The SEL typically updates its baseline standards every 5 years.

The SEL has also discovered that a user-friendly format is important to creating standards that are actually used and consulted. The SEL guidebooks feature graphics to illustrate concepts and are designed to make information easy to find. They are also intended to be used primarily as references rather than one-time reading.

However, most important is the process by which the SEL gathers the information and ensures that the standards reflect the actual process. In the early stages of packaging standards, developers, maintainers, testers, and managers are interviewed to gather new and updated information. Facilitated workshops are then used to develop consensus on the process content. This information is further validated by analyzing empirical data. Then a small team of packagers with excellent communication skills is tasked with developing the final package.

Tools

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 data base 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 data base, the manager can predict future activities on the current project. (For more details on the SME, see Reference 23.) 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.

The SEL also provides tools to automate parts of the software measurement process. The SEL developed an automated tool for developers to use to complete data collection forms that require simple transcription (e.g., computer usage and component attributes) rather than thoughtful completion (e.g., change reports and effort allocation).

Training

As part of the packaging process, the SEL has developed a training program, which is outlined in a detailed training plan (Reference 24). The program consists of a standard set of courses designed to provide all of the developers, managers, analysts, and data base 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 data base 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.

Section 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).

6.1 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 its 17-year lifetime, but it also has tracked quite closely expenditures for process change efforts. The SEL investment in process change activities can be divided into three 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 11 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 approximately 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 3 percent of the total development budget. This figure includes purchase and design of data base management systems and distribution of SEL literature as well.

The analysis activity has been the most costly of all the expenditures in the SEL, averaging about 7 percent of development budgets. The responsibilities of the analysts include setting goals, defining experiments, interpreting measurement data, training the development/maintenance staff, developing standards, 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. Investment in analysis is a variable expense, depending on the experiments and technologies being researched and the amount of improvement payoff the organization is seeking at any time.

Over time, the SEL investment in process improvement has averaged 7 percent in research and analysis and 3–4 percent in data collection and data base support combined. While these numbers vary depending on the complexity of experimentation and the scope of the technologies being studied at any time, a local rule of thumb is to maintain data collection and data base support at no more than half of the investment in research and analysis.

6.2 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 also has been an ongoing goal, but this is a more subjective measure that 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 are routinely made between groups of projects developed at different times. For example, 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 is to assess the impact of process change on product characteristics. This was measured as improvement in the end product in the three key measures: defects, cost, and cycle time.

The early baseline comprises eight projects completed between 1985 and 1989 (see Table 5). 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 seven projects completed between 1990-1993 (see Table 6). 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.

Table 5. Early SEL Baseline (1985-1989)

PROJECT (No. & name)	% REUSE	COST* (Staff mos)	RELIABILITY (Error/KDLOC)
1. GROAGSS	14	381	4.42
2. COBEAGSS	12	348	5.22
3. GOESAGSS	12	261	5.18
4. UARSAGSS	10	675	2.81
5. GROSIM	18	79	8.91
6. COBSIM	11	39	4.45
7. GOESIM	29	96	1.72
8. UARSTELS	35	80	2.96

* Mission cost = cost of telemetry simulator + cost of AGSS (GRO = projects 1+5, COBE = 2+6, GOES = 3+7, UARS = 4+8).

Table 6. Current SEL Baseline (1990-1993)

PROJECT (No. & name)	% REUSE	COST [1] (Staff mos)	RELIABILITY (Error/KDLOC)
1. EUVEAGSS	81	155	1.22
2. SAMPEX	83	77	.76
3. WINDPOLR	18	476	n/a [2]
4. EUVETELS	96	36	.41
5. SAMPEXTS	95	21	.48
6. POWITS	69	77	2.39
7. TOMSTELS	97	n/a [3]	.23
8. FASTELS	92	n/a [3]	.69

1 Mission cost = cost of telemetry simulator + cost of AGSS (GRO = projects 1+5, COBE = 2+6, GOES = 3+7, UARS = 4+8).

2 Excluded since it used the Cleanroom development methodology where errors are counted differently.

3 Total mission cost for TOMS and FAST cannot be calculated since AGSSs are incomplete (they are not included in the cost baseline).

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 18).

Software cost was also compared between the two 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 two 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 19 shows the comparison of the cost data. The significant decrease in cost can be attributed to increases in both productivity and code reuse (Figure 20). This comparison shows that the average cost per mission has decreased by over 50 percent over the 8-year period.

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.

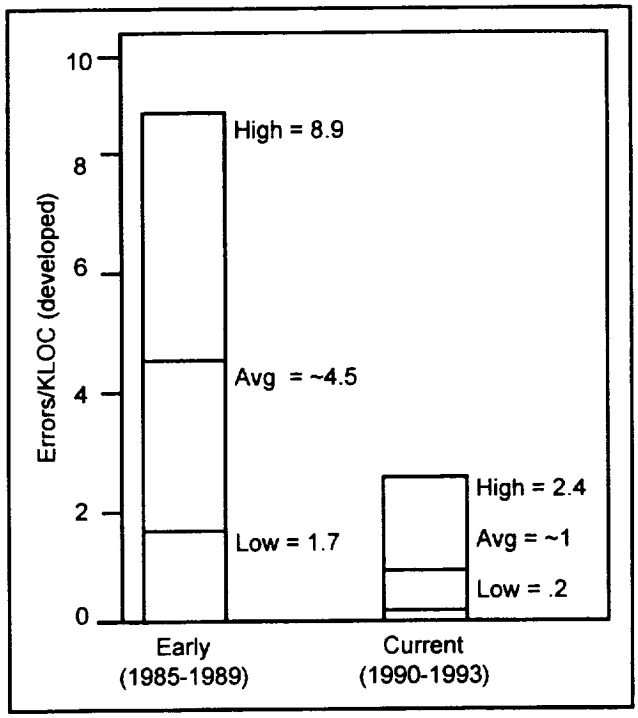


Figure 18. Impact on SEL Products (Reliability)

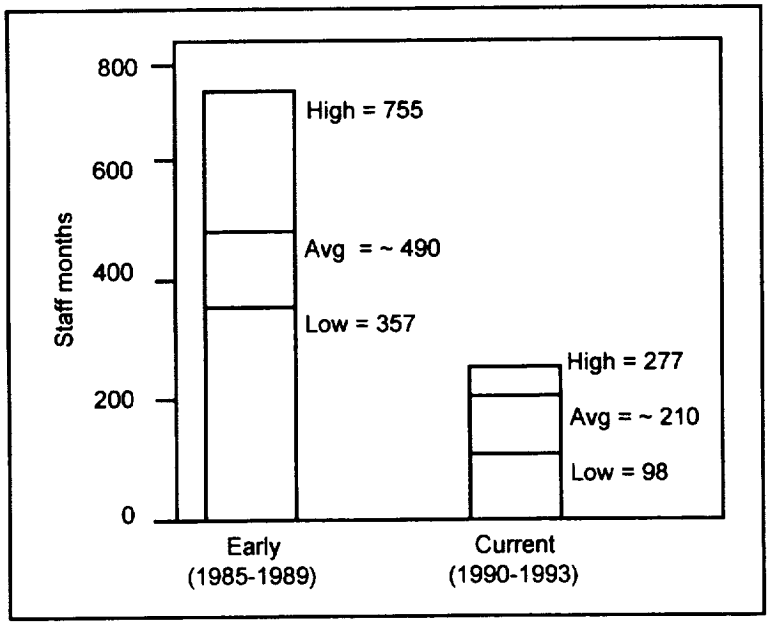


Figure 19. Impact on SEL Products (Cost)

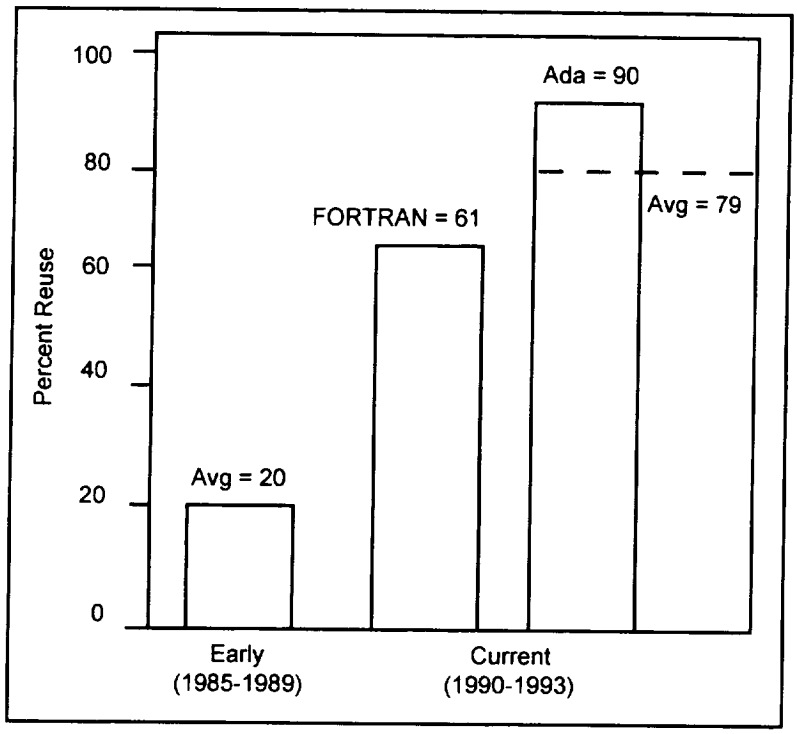


Figure 20. Impact on SEL Products (Reuse)

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 21). 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.

6.3 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 standards, 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, significant improvement to the software development process occurs where the sustained, continuous incorporation of detailed techniques into higher level organizational processes effects an overall change in the environment. The most significant process attributes that distinguish the current SEL production environment from the environment of a decade ago include:

- **Process change** has been infused as a standard business practice.

  All standards and training material 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 as documentation. It is expected, applied, and effective.

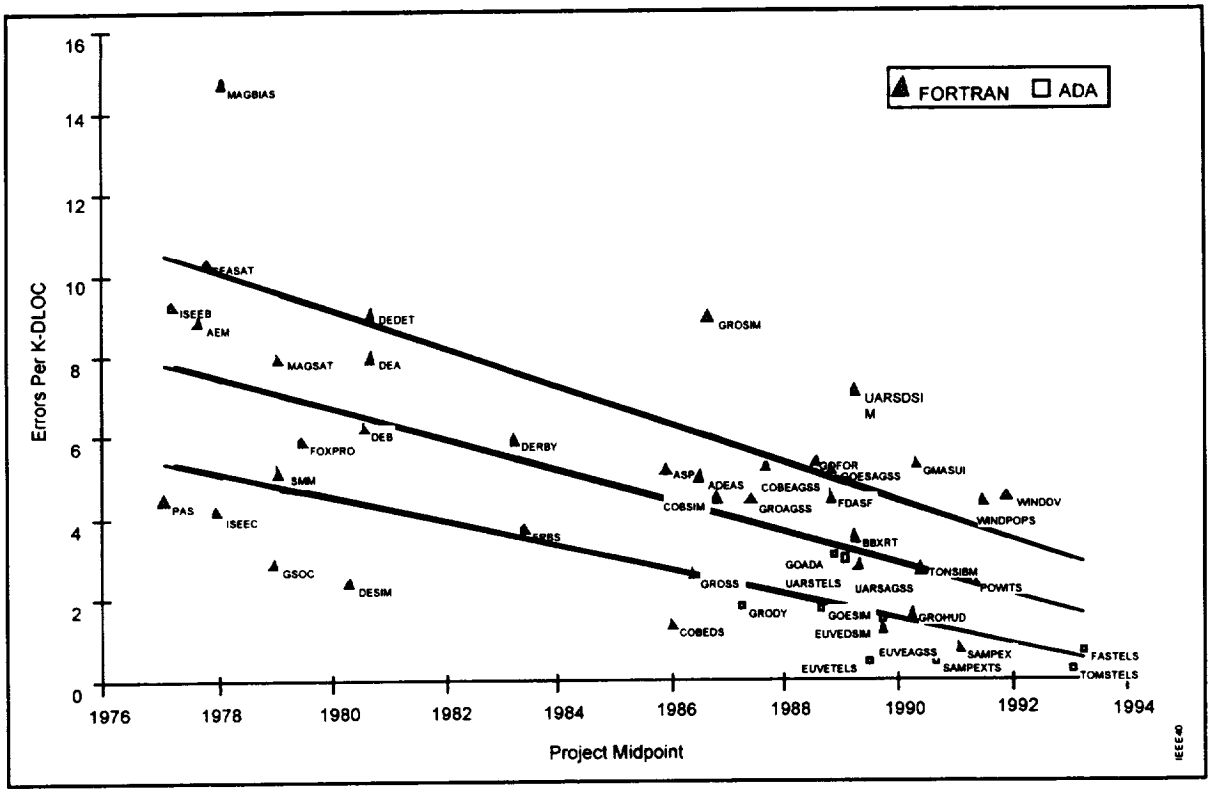


Figure 21. Development Error Rates (1977-1994)

- **Change is now driven by *product* and *process*,** not merely process alone.

  As the process improvement program has matured over the years, an equal concern has developed for 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 most effective process changes 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 people-driven approaches have had farther reaching effects.

The improvements in product characteristics and the changes to the standard process in this environment illustrate the impact of the FDD's investment in the SEL improvement program. Today, software developers in this organization are building better software more efficiently using many techniques and methods considered experimental only a few years ago. Their progress has been facilitated throughout by the SEL, whose focus on defining organizational goals, expanding domain understanding, and judiciously applying new technology has enabled the FDD to maximize the lessons learned from local experience.

Appendix A - Sample SEL Experiment Plan

SEL Representative Study Plan for SOHOTELS

October 11, 1993

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

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

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)
  - Growth in schedule estimates and size estimates (from initial estimates to completion and from PDR/CDR to completion)

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 data base 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.

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)

References

1. Basili, V. R., "Quantitative Evaluation of a Software Engineering Methodology," *Proceedings of the First Pan Pacific Computer Conference,* Melbourne, Australia, September 1985

2. Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering,* November 1984

3. Basili, V. R., "Software Development: A Paradigm for the Future (Keynote Address)," *Proceedings COMPSAC '89,* Orlando, Florida, September 1989

4. Basili, V. R., and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," *Proceedings of the Ninth International Conference on Software Engineering,* Monterey, California, April 1987

5. Basili, V. R., and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering,* Vol. 14, No. 6, June 1988

6. Basili, V. R., et al., "The Software Engineering Laboratory–An Operational Software Experience Factory," *Proceedings of the Fourteenth International Conference on Software Engineering,* Melbourne, Australia, May 1992

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

8. Paulk, M., B. Curtis, M. Chrissis, and C. Weber, *Capability Maturity Model for Software, Version 1.1,* Software Engineering Institute, Carnegie Mellon University, CMU/SEI-93-TR-24, February 1993

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

10. Bassman, M., F. McGarry, and R. Pajerski, *Software Measurement Guidebook,* Software Engineering Laboratory, SEL-94-003, July 1994

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

12. Stephens, W. P., G. J. Meyers, and L. L. Constantine, "Structured Design," *IBM Systems Journal,* Vol. 3, No. 2, 1974

13. Card, D. N., G. Page, and F. E. McGarry, "Criteria for Software Modularization," *Proceedings of the Eighth International Conference on Software Engineering,* London, 1985

14. Basili, V. R., and R. W. Selby, "Comparing the Effectiveness of Testing Strategies," *IEEE Transactions on Software Engineering,* December 1987

15. Basili, V. R., and S. Green, "Software Process Evolution at the SEL," *IEEE Software,* July 1994

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

17. 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

18. Morusiewicz, L., and J. Valett, *Annotated Bibliography of Software Engineering Laboratory Literature,* Software Engineering Laboratory, SEL-82-1206, November 1993

19. Green, S., *Software Engineering Laboratory Cleanroom Process Model,* Software Engineering Laboratory, SEL-91-004, November 1991

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

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

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

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

24. Doland, J. T., R. Pajerski, and S. Waligora, *Software Engineering Laboratory Training Plan,* Software Engineering Laboratory, SEL-93-TP1, September 1993

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, *Proceedings From the First Summer Software Engineering Workshop*, August 1976

SEL-77-002, *Proceedings From the Second Summer Software Engineering Workshop*, September 1977

SEL-78-005, *Proceedings From the Third Summer Software Engineering Workshop*, September 1978

SEL-78-006, *GSFC Software Engineering Research Requirements Analysis Study*, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, *Applicability of the Rayleigh Curve to the SEL Environment*, T. E. Mapp, December 1978

SEL-78-302, *FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3)*, W. J. Decker, W. A. Taylor, et al., July 1986

SEL-79-002, *The Software Engineering Laboratory: Relationship Equations*, K. Freburger and V. R. Basili, May 1979

SEL-79-004, *Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment*, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, *Proceedings From the Fourth Summer Software Engineering Workshop*, November 1979

SEL-80-002, *Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation*, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-005, *A Study of the Musa Reliability Model*, A. M. Miller, November 1980

SEL-80-006, *Proceedings From the Fifth Annual Software Engineering Workshop*, November 1980

SEL-80-007, *An Appraisal of Selected Cost/Resource Estimation Models for Software Systems*, J. F. Cook and F. E. McGarry, December 1980

SEL-80-008, *Tutorial on Models and Metrics for Software Management and Engineering*, V. R. Basili, 1980

SEL-81-011, *Evaluating Software Development by Analysis of Change Data*, D. M. Weiss, November 1981

SEL-81-012, *The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems*, G. O. Picasso, December 1981

SEL-81-013, *Proceedings of the Sixth Annual Software Engineering Workshop*, December 1981

SEL-81-014, *Automated Collection of Software Engineering Data in the Software Engineering Laboratory* (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, *Guide to Data Collection*, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-104, *The Software Engineering Laboratory*, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

SEL-81-110, *Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics*, G. Page, F. E. McGarry, and D. N. Card, June 1985

SEL-81-305, *Recommended Approach to Software Development*, L. Landis, S. Waligora, F. E. McGarry, et al., June 1992

SEL-81-305SP1, *Ada Developers' Supplement to the Recommended Approach*, R. Kester and L. Landis, November 1993

SEL-82-001, *Evaluation of Management Measures of Software Development*, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-004, *Collected Software Engineering Papers*: Volume 1, July 1982

SEL-82-007, *Proceedings of the Seventh Annual Software Engineering Workshop*, December 1982

SEL-82-008, *Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory*, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, *FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1)*, W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, *Glossary of Software Engineering Laboratory Terms*, T. A. Babst, M. G. Rohleder, and F. E. McGarry, October 1983

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

SEL-83-001, *An Approach to Software Cost Estimation*, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, *Measures and Metrics for Software Development*, D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, *Collected Software Engineering Papers*: Volume II, November 1983

SEL-83-007, *Proceedings of the Eighth Annual Software Engineering Workshop*, November 1983

SEL-83-106, *Monitoring Software Development Through Dynamic Variables (Revision 1)*, C. W. Doerflinger, November 1989

SEL-84-003, *Investigation of Specification Measures for the Software Engineering Laboratory (SEL)*, W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, *Proceedings of the Ninth Annual Software Engineering Workshop*, November 1984

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

SEL-85-001, *A Comparison of Software Verification Techniques*, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

SEL-85-002, *Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team*, R. Murphy and M. Stark, October 1985

SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985

SEL-85-004, *Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics*, R. W. Selby, Jr., and V. R. Basili, May 1985

SEL-85-005, *Software Verification and Testing*, D. N. Card, E. Edwards, F. McGarry, and C. Antle, December 1985

SEL-85-006, *Proceedings of the Tenth Annual Software Engineering Workshop*, December 1985

SEL-86-001, *Programmer's Handbook for Flight Dynamics Software Development*, R. Wood and E. Edwards, March 1986

SEL-86-002, *General Object-Oriented Software Development*, E. Seidewitz and M. Stark, August 1986

SEL-86-003, *Flight Dynamics System Software Development Environment (FDS/SDE) Tutorial*, J. Buell and P. Myers, July 1986

SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986

SEL-86-005, *Measuring Software Design*, D. N. Card et al., November 1986

SEL-86-006, *Proceedings of the Eleventh Annual Software Engineering Workshop*, December 1986

SEL-87-001, *Product Assurance Policies and Procedures for Flight Dynamics Software Development*, S. Perry et al., March 1987

SEL-87-002, *Ada® Style Guide (Version 1.1)*, E. Seidewitz et al., May 1987

SEL-87-003, *Guidelines for Applying the Composite Specification Model (CSM)*, W. W. Agresti, June 1987

SEL-87-004, *Assessing the Ada® Design Process and Its Implications: A Case Study*, S. Godfrey, C. Brophy, et al., July 1987

SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987

SEL-87-010, *Proceedings of the Twelfth Annual Software Engineering Workshop*, December 1987

SEL-88-001, *System Testing of a Production Ada Project: The GRODY Study*, J. Seigle, L. Esker, and Y. Shi, November 1988

SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988

SEL-88-003, *Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis*, K. Quimby and L. Esker, December 1988

SEL-88-004, *Proceedings of the Thirteenth Annual Software Engineering Workshop*, November 1988

SEL-88-005, *Proceedings of the First NASA Ada User's Symposium*, December 1988

SEL-89-002, *Implementation of a Production Ada Project: The GRODY Study*, S. Godfrey and C. Brophy, September 1989

SEL-89-004, *Evolution of Ada Technology in the Flight Dynamics Area: Implementation/Testing Phase Analysis*, K. Quimby, L. Esker, L. Smith, M. Stark, and F. McGarry, November 1989

SEL-89-005, *Lessons Learned in the Transition to Ada From FORTRAN at NASA/Goddard*, C. Brophy, November 1989

SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989

SEL-89-007, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, November 1989

SEL-89-008, *Proceedings of the Second NASA Ada Users' Symposium*, November 1989

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

SEL-89-301, *Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 3)*, L. Morusiewicz, December 1993

SEL-90-001, *Database Access Manager for the Software Engineering Laboratory (DAMSEL) User's Guide*, M. Buhler, K. Pumphrey, and D. Spiegel, March 1990

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

SEL-90-003, *A Study of the Portability of an Ada System in the Software Engineering Laboratory (SEL)*, L. O. Jun and S. R. Valett, June 1990

SEL-90-004, *Gamma Ray Observatory Dynamics Simulator in Ada (GRODY) Experiment Summary*, T. McDermott and M. Stark, September 1990

SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990

SEL-90-006, *Proceedings of the Fifteenth Annual Software Engineering Workshop*, November 1990

SEL-91-001, *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules*, W. Decker, R. Hendrick, and J. Valett, February 1991

SEL-91-003, *Software Engineering Laboratory (SEL) Ada Performance Study Report*, E. W. Booth and M. E. Stark, July 1991

SEL-91-004, *Software Engineering Laboratory (SEL) Cleanroom Process Model*, S. Green, November 1991

SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991

SEL-91-006, *Proceedings of the Sixteenth Annual Software Engineering Workshop*, December 1991

SEL-91-102, *Software Engineering Laboratory (SEL) Data and Information Policy (Revision 1)*, F. McGarry, August 1991

SEL-92-001, *Software Management Environment (SME) Installation Guide*, D. Kistler and K. Jeletic, January 1992

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

SEL-92-003, *Collected Software Engineering Papers: Volume X*, November 1992

SEL-92-004, *Proceedings of the Seventeenth Annual Software Engineering Workshop*, December 1992

SEL-93-001, *Collected Software Engineering Papers: Volume XI*, November 1993

SEL-93-002, *Cost and Schedule Estimation Study Report*, S. Condon, M. Regardie, M. Stark, et al., November 1993

SEL-93-003, *Proceedings of the Eighteenth Annual Software Engineering Workshop*, December 1993

SEL-94-001, *Software Management Environment (SME) Components and Algorithms*, R. Hendrick, D. Kistler, and J. Valett, February 1994

SEL-94-002, *Software Measurement Guidebook*, M. Bassman, F. McGarry, R. Pajerski, July 1994

SEL-94-003, *C Style Guide*, J. Doland and J. Valett, August 1994

SEL-94-004, *Collected Software Engineering Papers: Volume XII*, November 1994

SEL-94-005, *An Overview of the Software Engineering Laboratory*, F. McGarry, G. Page, V. Basili, et al., December 1994

SEL-RELATED LITERATURE

[10]Abd-El-Hafiz, S. K., V. R. Basili, and G. Caldiera, "Towards Automated Support for Extraction of Reusable Components," *Proceedings of the IEEE Conference on Software Maintenance-1991 (CSM 91)*, October 1991

[4]Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

[2]Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," *Program Transformation and Programming Environments*. New York: Springer-Verlag, 1984

[1]Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," *Proceedings of the Fifth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1981

[8]Bailey, J. W., and V. R. Basili, "Software Reclamation: Improving Post-Development Reusability," *Proceedings of the Eighth Annual National Conference on Ada Technology*, March 1990

[10]Bailey, J. W., and V. R. Basili, "The Software-Cycle Model for Re-Engineering and Reuse," *Proceedings of the ACM Tri-Ada 91 Conference*, October 1991

[1]Basili, V. R., "Models and Metrics for Software Management and Engineering," *ASME Advances in Computer Technology*, January 1980, vol. 1

Basili, V. R., *Tutorial on Models and Metrics for Software Management and Engineering*. New York: IEEE Computer Society Press, 1980 (also designated SEL-80-008)

[3]Basili, V. R., "Quantitative Evaluation of Software Methodology," *Proceedings of the First Pan-Pacific Computer Conference*, September 1985

[7]Basili, V. R., *Maintenance = Reuse-Oriented Software Development*, University of Maryland, Technical Report TR-2244, May 1989

[7]Basili, V. R., *Software Development: A Paradigm for the Future*, University of Maryland, Technical Report TR-2263, June 1989

[8]Basili, V. R., "Viewing Maintenance of Reuse-Oriented Software Development," *IEEE Software*, January 1990

[1]Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

[9]Basili, V. R., G. Caldiera, and G. Cantone, "A Reference Architecture for the Component Factory," *ACM Transactions on Software Engineering and Methodology*, January 1992

[10]Basili, V., G. Caldiera, F. McGarry, et al., "The Software Engineering Laboratory—An Operational Software Experience Factory," *Proceedings of the Fourteenth International Conference on Software Engineering (ICSE 92)*, May 1992

[1]Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

[12]Basili, V., and S. Green, "Software Process Evolution at the SEL," *IEEE Software*, July 1994

[3]Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," *Proceedings of the International Computer Software and Applications Conference*, October 1985

[4]Basili, V. R., and D. Patnaik, *A Study on Fault Prediction and Reliability Assessment in the SEL Environment*, University of Maryland, Technical Report TR-1699, August 1986

[2]Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM*, January 1984, vol. 27, no. 1

[1]Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," *Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics*, March 1981

[3]Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P—A Prototype Expert System for Software Engineering Management," *Proceedings of the IEEE/MITRE Expert Systems in Government Symposium*, October 1985

Basili, V. R., and J. Ramsey, *Structural Coverage of Functional Testing*, University of Maryland, Technical Report TR-1442, September 1984

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," *Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost*. New York: IEEE Computer Society Press, 1979

[5]Basili, V. R., and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," *Proceedings of the 9th International Conference on Software Engineering*, March 1987

[5]Basili, V. R., and H. D. Rombach, "TAME: Tailoring an Ada Measurement Environment," *Proceedings of the Joint Ada Conference*, March 1987

[5]Basili, V. R., and H. D. Rombach, "TAME: Integrating Measurement Into Software Environments," University of Maryland, Technical Report TR-1764, June 1987

[6]Basili, V. R., and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, June 1988

[7]Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment*, University of Maryland, Technical Report TR-2158, December 1988

[8]Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: Model-Based Reuse Characterization Schemes*, University of Maryland, Technical Report TR-2446, April 1990

[9]Basili, V. R., and H. D. Rombach, "Support for Comprehensive Reuse," *Software Engineering Journal*, September 1991

[3]Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environment's Characteristic Software Metric Set," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

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

[3]Basili, V. R., and R. W. Selby, Jr., "Four Applications of a Software Data Collection and Analysis Methodology," *Proceedings of the NATO Advanced Study Institute*, August 1985

[5]Basili, V. R., and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering*, December 1987

[9]Basili, V. R., and R. W. Selby, "Paradigms for Experimentation and Empirical Studies in Software Engineering," *Reliability Engineering and System Safety*, January 1991

[4]Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, July 1986

[2]Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," *IEEE Transactions on Software Engineering*, November 1983

[2]Basili, V. R., and D. M. Weiss, *A Methodology for Collecting Valid Software Engineering Data*, University of Maryland, Technical Report TR-1235, December 1982

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

[1]Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," *Proceedings of the Fifteenth Annual Conference on Computer Personnel Research*, August 1977

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," *Proceedings of the Software Life Cycle Management Workshop*, September 1977

[1]Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," *Proceedings of the Second Software Life Cycle Management Workshop*, August 1978

[1]Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," *Computers and Structures*, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," *Proceedings of the Third International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1978

Bassman, M. J., F. McGarry, and R. Pajerski, *Software Measurement Guidebook*, NASA-GB-001-94, Software Engineering Program, July 1994

[9]Booth, E. W., and M. E. Stark, "Designing Configurable Software: COMPASS Implementation Concepts," *Proceedings of Tri-Ada 1991*, October 1991

[10]Booth, E. W., and M. E. Stark, "Software Engineering Laboratory Ada Performance Study—Results and Implications," *Proceedings of the Fourth Annual NASA Ada User's Symposium*, April 1992

[10]Briand, L. C., and V. R. Basili, "A Classification Procedure for the Effective Management of Changes During the Maintenance Process," *Proceedings of the 1992 IEEE Conference on Software Maintenance (CSM 92)*, November 1992

[10]Briand, L. C., V. R. Basili, and C. J. Hetmanski, "Providing an Empirical Basis for Optimizing the Verification and Testing Phases of Software Development," *Proceedings of the Third IEEE International Symposium on Software Reliability Engineering (ISSRE 92)*, October 1992

[11]Briand, L. C., V. R. Basili, and C. J. Hetmanski, *Developing Interpretable Models with Optimized Set Reduction for Identifying High Risk Software Components*, University of Maryland, Technical Report TR-3048, March 1993

[12]Briand, L. C., V. R. Basili, Y. Kim, and D. R. Squire, "A Change Analysis Process to Characterize Software Maintenance Projects", *Proceedings of the International Conference on Software Maintenance*, September 1994

[9]Briand, L. C., V. R. Basili, and W. M. Thomas, *A Pattern Recognition Approach for Software Engineering Data Analysis*, University of Maryland, Technical Report TR-2672, May 1991

[11]Briand, L. C., S. Morasca, and V. R. Basili, "Measuring and Assessing Maintainability at the End of High Level Design," *Proceedings of the 1993 IEEE Conference on Software Maintenance (CSM 93)*, November 1993

[12]Briand, L., S. Morasca, and V. R. Basili, *Defining and Validationg High-Level Design Metrics*, University of Maryland, Technical Report TR-3301, June 1994

[11]Briand, L. C., W. M. Thomas, and C. J. Hetmanski, "Modeling and Managing Risk Early in Software Development," *Proceedings of the Fifteenth International Conference on Software Engineering (ICSE 93)*, May 1993

[5]Brophy, C. E., W. W. Agresti, and V. R. Basili, "Lessons Learned in Use of Ada-Oriented Design Methods," *Proceedings of the Joint Ada Conference*, March 1987

[6]Brophy, C. E., S. Godfrey, W. W. Agresti, and V. R. Basili, "Lessons Learned in the Implementation Phase of a Large Ada Project," *Proceedings of the Washington Ada Technical Conference*, March 1988

[2]Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

[2]Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

[3]Card, D. N., "A Software Technology Evaluation Program," *Annais do XVIII Congresso Nacional de Informatica*, October 1985

[5]Card, D. N., and W. W. Agresti, "Resolving the Software Science Anomaly," *Journal of Systems and Software*, 1987

[6]Card, D. N., and W. W. Agresti, "Measuring Software Design Complexity," *Journal of Systems and Software*, June 1988

[4]Card, D. N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," *IEEE Transactions on Software Engineering*, February 1986

Card, D. N., V. E. Church, W. W. Agresti, and Q. L. Jordan, "A Software Engineering View of Flight Dynamics Analysis System," Parts I and II, Computer Sciences Corporation, Technical Memorandum, February 1984

Card, D. N., Q. L. Jordan, and V. E. Church, "Characteristics of FORTRAN Modules," Computer Sciences Corporation, Technical Memorandum, June 1984

[5]Card, D. N., F. E. McGarry, and G. T. Page, "Evaluating Software Engineering Technologies," *IEEE Transactions on Software Engineering*, July 1987

[3]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

[1]Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," *Proceedings of the Fifth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1981

[4]Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Prototypes," *ACM Software Engineering Notes*, July 1986

[2]Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," *Proceedings of the Seventh International Computer Software and Applications Conference*. New York: IEEE Computer Society Press, 1983

Doubleday, D., *ASAP: An Ada Static Source Code Analyzer Program*, University of Maryland, Technical Report TR-1895, August 1987 (NOTE: 100 pages long)

[6]Godfrey, S., and C. Brophy, "Experiences in the Implementation of a Large Ada Project," *Proceedings of the 1988 Washington Ada Symposium*, June 1988

[5]Jeffery, D. R., and V. Basili, *Characterizing Resource Data: A Model for Logical Association of Software Data*, University of Maryland, Technical Report TR-1848, May 1987

[6]Jeffery, D. R., and V. R. Basili, "Validating the TAME Resource Data Model," *Proceedings of the Tenth International Conference on Software Engineering*, April 1988

[11]Li, N. R., and M. V. Zelkowitz, "An Information Model for Use in Software Management Estimation and Prediction," *Proceedings of the Second International Conference on Information Knowledge Management*, November 1993

[5]Mark, L., and H. D. Rombach, *A Meta Information Base for Software Engineering*, University of Maryland, Technical Report TR-1765, July 1987

[6]Mark, L., and H. D. Rombach, "Generating Customized Software Engineering Information Bases From Software Process and Product Specifications," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

[5]McGarry, F. E., and W. W. Agresti, "Measuring Ada for Software Development in the Software Engineering Laboratory (SEL)," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

[7]McGarry, F., L. Esker, and K. Quimby, "Evolution of Ada Technology in a Production Software Environment," *Proceedings of the Sixth Washington Ada Symposium (WADAS)*, June 1989

[3]McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," *Proceedings of the Hawaiian International Conference on System Sciences*, January 1985

[3]Page, G., F. E. McGarry, and D. N. Card, "A Practical Experience With Independent Verification and Validation," *Proceedings of the Eighth International Computer Software and Applications Conference*, November 1984

[12]Porter, A. A., L. G. Votta, Jr., and V. R. Basili, *Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment*, University of Maryland, Technical Report TR-3327, July 1994

[5]Ramsey, C. L., and V. R. Basili, "An Evaluation of Expert Systems for Software Engineering Management," *IEEE Transactions on Software Engineering*, June 1989

[3]Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

[5]Rombach, H. D., "A Controlled Experiment on the Impact of Software Structure on Maintainability," *IEEE Transactions on Software Engineering*, March 1987

[8]Rombach, H. D., "Design Measurement: Some Lessons Learned," *IEEE Software*, March 1990

[9]Rombach, H. D., "Software Reuse: A Key to the Maintenance Problem," *Butterworth Journal of Information and Software Technology*, January/February 1991

[6]Rombach, H. D., and V. R. Basili, "Quantitative Assessment of Maintenance: An Industrial Case Study," *Proceedings From the Conference on Software Maintenance*, September 1987

[6]Rombach, H. D., and L. Mark, "Software Process and Product Specifications: A Basis for Generating Customized SE Information Bases," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

[7]Rombach, H. D., and B. T. Ulery, *Establishing a Measurement Based Maintenance Improvement Program: Lessons Learned in the SEL*, University of Maryland, Technical Report TR-2252, May 1989

[10]Rombach, H. D., B. T. Ulery, and J. D. Valett, "Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL," *Journal of Systems and Software*, May 1992

[6]Seidewitz, E., "Object-Oriented Programming in Smalltalk and Ada," *Proceedings of the 1987 Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 1987

[5]Seidewitz, E., "General Object-Oriented Software Development: Background and Experience," *Proceedings of the 21st Hawaii International Conference on System Sciences*, January 1988

[6]Seidewitz, E., "General Object-Oriented Software Development with Ada: A Life Cycle Approach," *Proceedings of the CASE Technology Conference*, April 1988

[9]Seidewitz, E., "Object-Oriented Programming Through Type Extension in Ada 9X," *Ada Letters*, March/April 1991

[10]Seidewitz, E., "Object-Oriented Programming With Mixins in Ada," *Ada Letters*, March/April 1992

[12]Seidewitz, E., "Genericity versus Inheritance Reconsidered: Self-Reference Using Generics," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 1994

[4]Seidewitz, E., and M. Stark, "Towards a General Object-Oriented Software Development Methodology," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

[9]Seidewitz, E., and M. Stark, "An Object-Oriented Approach to Parameterized Software in Ada," *Proceedings of the Eighth Washington Ada Symposium*, June 1991

[8]Stark, M., "On Designing Parametrized Systems Using Ada," *Proceedings of the Seventh Washington Ada Symposium*, June 1990

[11]Stark, M., "Impacts of Object-Oriented Technologies: Seven Years of SEL Studies," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, September 1993*

[7]Stark, M. E. and E. W. Booth, "Using Ada to Maximize Verbatim Software Reuse," *Proceedings of TRI-Ada 1989*, October 1989

[5]Stark, M., and E. Seidewitz, "Towards a General Object-Oriented Ada Lifecycle," *Proceedings of the Joint Ada Conference*, March 1987

[10]Straub, P. A., and M. V. Zelkowitz, "On the Nature of Bias and Defects in the Software Specification Process," *Proceedings of the Sixteenth International Computer Software and Applications Conference (COMPSAC 92)*, September 1992

[8]Straub, P. A., and M. V. Zelkowitz, "PUC: A Functional Specification Language for Ada," *Proceedings of the Tenth International Conference of the Chilean Computer Science Society*, July 1990

[7]Sunazuka, T., and V. R. Basili, *Integrating Automated Support for a Software Management Cycle Into the TAME System*, University of Maryland, Technical Report TR-2289, July 1989

[10]Tian, J., A. Porter, and M. V. Zelkowitz, "An Improved Classification Tree Analysis of High Cost Modules Based Upon an Axiomatic Definition of Complexity," *Proceedings of the Third IEEE International Symposium on Software Reliability Engineering (ISSRE 92)*, October 1992

Turner, C., and G. Caron, *A Comparison of RADC and NASA/SEL Software Development Data*, Data and Analysis Center for Software, Special Publication, May 1981

[10]Valett, J. D., "Automated Support for Experience-Based Software Management," *Proceedings of the Second Irvine Software Symposium (ISS_92)*, March 1992

[5]Valett, J. D., and F. E. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

[3]Weiss, D. M., and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory," *IEEE Transactions on Software Engineering*, February 1985

[5]Wu, L., V. R. Basili, and K. Reed, "A Structure Coverage Tool for Ada Software Systems," *Proceedings of the Joint Ada Conference*, March 1987

[1]Zelkowitz, M. V., "Resource Estimation for Medium-Scale Software Projects," *Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science*. New York: IEEE Computer Society Press, 1979

[2]Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," *Empirical Foundations for Computer and Information Science* (Proceedings), November 1982

[6]Zelkowitz, M. V., "The Effectiveness of Software Prototyping: A Case Study," *Proceedings of the 26th Annual Technical Symposium of the Washington, D.C., Chapter of the ACM*, June 1987

[6]Zelkowitz, M. V., "Resource Utilization During Software Development," *Journal of Systems and Software*, 1988

[8]Zelkowitz, M. V., "Evolution Towards Specifications Environment: Experiences With Syntax Editors," *Information and Software Technology*, April 1990

NOTES:

[1]This article also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I*, July 1982.

[2]This article also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983.

[3]This article also appears in SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985.

[4]This article also appears in SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986.

[5]This article also appears in SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987.

[6]This article also appears in SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988.

[7]This article also appears in SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989.

[8]This article also appears in SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990.

[9]This article also appears in SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991.

[10]This article also appears in SEL-92-003, *Collected Software Engineering Papers: Volume X*, November 1992.

[11]This article also appears in SEL-93-001, *Collected Software Engineering Papers: Volume XI*, November 1993.

[12]This article also appears in SEL-94-004, *Collected Software Engineering Papers: Volume XII*, November 1994.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1994	Contractor Report

4. TITLE AND SUBTITLE

An Overview of the Software Engineering Laboratory

5. FUNDING NUMBERS

552

6. AUTHOR(S)

Software Engineering Laboratory

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

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

8. PERFORMING ORGANIZATION REPORT NUMBER

SEL-94-005

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

NASA Aeronautics and Space Administration
Washington, D.C. 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

CR-189410

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified-Unlimited
Subject Category: 61
Report is available from the NASA Center for AeroSpace Information,
800 Elkridge Landing Road, Linthicum Heights, MD 21090; (301) 621-0390.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes the background and structure of the SEL organization, the SEL process improvement approach, and its 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 total investment in process improvement with the measurable improvements seen in the organization's software product.

14. SUBJECT TERMS

Software Engineering, Software Measurement, Data Collection

15. NUMBER OF PAGES

60

16. PRICE CODE

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