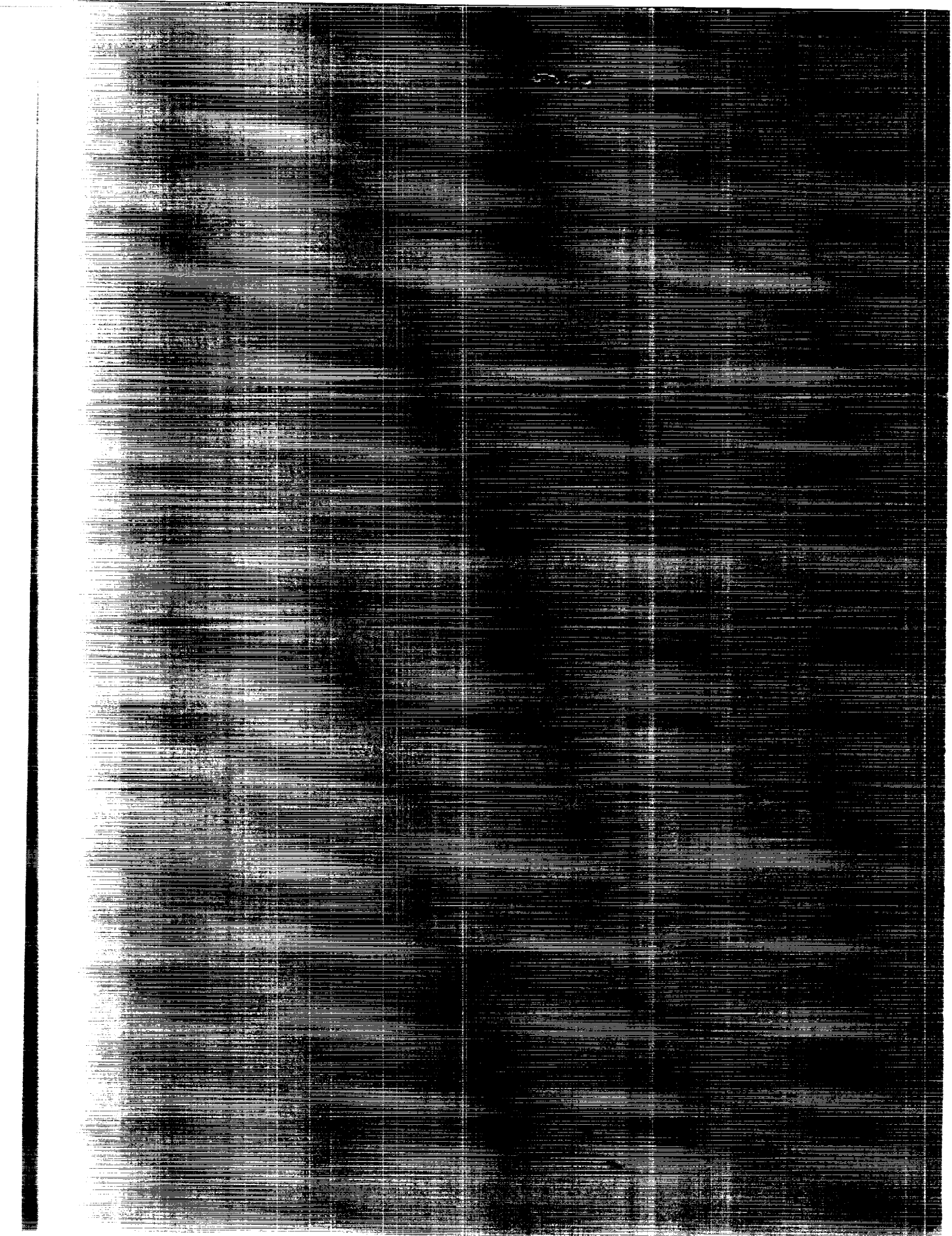
SERIES

SEL-90-002

ROOM CASE
E SOFTWARE
LABORATORY:
SCRIPTION AND
ANALYSIS

RCH 1990



SOFTWARE ENGINEERING LABORATORY SERIES SEL-90-002

THE CLEANROOM CASE STUDY IN THE SOFTWARE ENGINEERING LABORATORY: PROJECT DESCRIPTION AND EARLY ANALYSIS

MARCH 1990

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 for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC, Systems Development Branch
The University of Maryland, Computer Sciences Department
Computer Sciences Corporation, Systems Development 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 this 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 primary authors of this document are

Scott Green	(NASA/GSFC)
Ara Kouchakdjian	(University of Maryland)
Victor Basili	(University of Maryland)
David Weidow	(NASA/GSFC)

Additionally, the following persons contributed significantly:

Richard Burley	(NASA/GSFC)
James Jeletic	(NASA/GSFC)
Jon Valett	(NASA/GSFC)

Single copies of this document can be obtained by writing to

Systems Development Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

ACKNOWLEDGMENTS

The authors would like to thank Frank McGarry of NASA/GSFC for his assistance in making the Cleanroom case study possible. We would also like to thank Michael Dyer and F. Terry Baker of IBM Systems Integration Division (IBM–SID) for their time and advice, before and during the project.

ABSTRACT

This case study analyzes the application of the Cleanroom software development methodology to the development of production software at the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC). The Cleanroom methodology emphasizes human discipline in program verification to produce reliable software products that are "right the first time."

Preliminary analysis of the Cleanroom case study shows that the method can be applied successfully in the FDD environment and may increase staff productivity and product quality. Compared to typical Software Engineering Laboratory (SEL) activities, there is evidence of lower failure rates, a more complete and consistent set of inline code documentation, a different distribution of phase effort activity, and a different growth profile in terms of lines of code developed.

EXECUTIVE SUMMARY

E.1 INTRODUCTION

The Software Engineering Laboratory (SEL) at the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) investigates the effectiveness of software engineering technologies when applied to the development of applications software within GSFC's Flight Dynamics Division (FDD). One such technology currently being examined by the SEL is the Cleanroom software development methodology. There are several significant differences between the SEL Cleanroom model and the standard SEL development methodology, including the following items:

- Cleanroom testers and developers are on completely separate teams.
- Cleanroom developers have no access to the mainframe computer for compilation and testing purposes.
- Cleanroom developers rely on code reading instead of unit testing to verify correctness of the software prior to system testing.
- Cleanroom testers use a statistical testing approach.

E.2 OBJECTIVES AND SCOPE

This case study analyzes the application of the Cleanroom methodology on part of a large ground support system used in mission support of attitude determination requirements. The system selected for the study was the Coarse/Fine Attitude Determination Subsystem (CFADS) of the Upper Atmosphere Research Satellite (UARS). The completed system contained approximately 34,000 source lines of primarily FORTRAN code. The major goals of the study were to

- Assess the process used in the SEL Cleanroom model with respect to team structure, team activities, and effort distribution
- Analyze the products of the SEL Cleanroom model and determine the impact on measures of interest, including reliability, productivity, overall life-cycle costs, and software quality
- Analyze the residual products in the application of the SEL Cleanroom model, such as fault distribution, error characteristics, system growth, and computer usage

E.3 PRELIMINARY ANALYSIS

The following statements summarize this report's findings concerning preliminary analysis of the Cleanroom methodology and its application in the FDD:

- The project members were able to successfully apply a tailored version of the Cleanroom methodology.

- The Cleanroom methodology enhances a "team" development approach and minimizes individual programming styles.
- Preliminary analysis of key measures indicates an increase in productivity and reliability and a decrease in rework effort, as compared with typical SEL projects.
- The Cleanroom effort distribution shows a significant increase in design effort and decrease in coding effort.
- Informal design reviews held by the development team appeared to be an effective method of early fault detection.
- Less than one-third of the faults uncovered in the code-reading process were found by multiple code readers.
- The breakdown of effort spent in code writing versus code reading was approximately even, as compared with a typical SEL ratio of 6 to 1 in favor of code writing.
- All team members indicated a willingness to reapply the methodology on future projects.

E.4 CONCLUSIONS

Based on the findings indicated in the preliminary analysis of the project, the following conclusions have been drawn:

- The Cleanroom methodology can be applied successfully in the FDD environment, but additional tailoring is required.
- The methodology had a favorable impact on all key measures of interest.
- Early concerns, such as the team's experience level, the unstable specifications environment, and the psychological impact of the methodology on team members, appeared to have little or no impact on the application of the methodology.
- The separation of teams forced the developers to apply a more thorough effort in design and code verification.
- The impact of the testing approach cannot be analyzed fully until the project has completed its formal acceptance testing phase.
- Based on the favorable results found here, further studies are called for.

Table of Contents

List of Illustrations

List of Tables

SECTION 1—INTRODUCTION

This case study analyzes the application of the Cleanroom software development methodology to the development of production software at the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC). The case study involves the methodology's application on part of a large ground support system used in mission support of attitude determination requirements.

In addition to describing the Cleanroom methodology, this paper analyzes its application in comparison to the existing Software Engineering Laboratory (SEL) methodology used in GSFC's Flight Dynamics Division (FDD) (References 1 and 2). The analysis covers the phases from project planning through system test. Areas of analysis include the tailoring and use of the method, as well as effort, defect, and productivity statistics. The emphasis of this study is on understanding the methodology and its applicability to the SEL environment, rather than on a detailed assessment in relation to other development methodologies.

SECTION 2—THE CLEANROOM METHODOLOGY: HISTORY, DESCRIPTION, AND PRIOR EXPERIENCES

The Cleanroom software development methodology (References 3, 4, 5, 6, 7, and 8) was conceived in the early 1980s by Dr. Harlan Mills at IBM. The term "Cleanroom" originates in the integrated circuit (IC) production process, where ICs are assembled in dust-free "clean rooms" to prevent the destructive effects of dust. When applying the Cleanroom methodology to the software development process, the primary focus is on defect prevention rather than defect removal.

The goal of Cleanroom is to use a structured development process to build a product that is "right the first time," instead of using the test process to reach a desired level of reliability. The essential elements of Cleanroom include an emphasis on human discipline and a stress on the use of a structured development approach. These elements are enforced in a variety of ways. First, there is a complete separation of development and test activities. Second, there is a reliance on "reading" for correctness by the developers. Third, the purpose of testing is for quality assessment rather than for debugging or finding defects. Finally, a top-down development approach, with the use of stubs, is followed to allow for early assessment of product quality. Figure 2-1 illustrates the organization needed to follow the disciplined approach necessary for Cleanroom. These efforts result in a more structured development process, which appears to be the direction of software engineering.

In previous uses of Cleanroom (References 5, 6, 8, and 9), the actual development activities were based on the IBM Systems Integration Division (IBM-SID) model for software development, which has its foundation in structured programming (Reference 10). Using top-down design and development, a system was divided into increments, which allowed developers to concentrate on small parts of the system at any one time. The formal design process included the use of state machines to help modularize the system and to support the concept of information hiding. Development progressed through stepwise refinement, and each successive step was verified by stepwise abstraction to ensure correctness. In addition, review/inspection activities occurred at various milestones (Reference 11). The purpose of the verification and review activities was not to find defects but to confirm correctness. Since verification and reviews were perceived as constructive (positive) activities, team development was reinforced.

Testers for Cleanroom projects used a statistical testing approach (References 3, 4, 11, and 12). As with the development process, the purpose of the test process was to confirm correctness and project future reliability, rather than to find defects. For this reason, a black-box testing method—specifically, statistical testing—was preferred to a white-box testing approach (Reference 4). Data for test cases were selected according to the operational profile of the system. All inputs to the

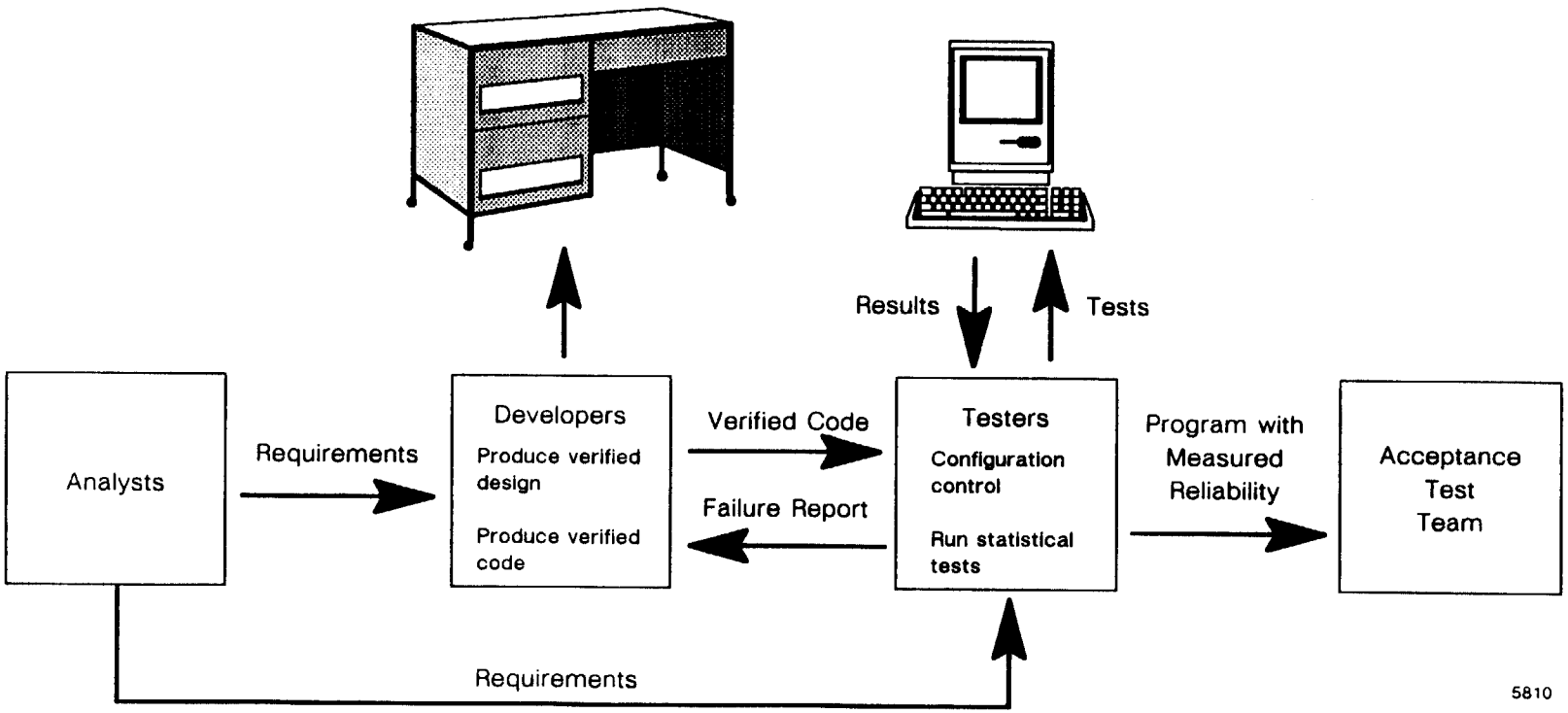


Figure 2-1. The Cleanroom Development Process

system were determined, and a probability distribution of possible values for each input was calculated. Test cases were then statistically generated. Top-down development allowed system test to begin once the first increment was submitted. In addition, since data were statistically generated, it was possible to determine and project the reliability of the system in terms of mean time to failure (MTTF).

As previously stated, this case study was not the first to use the Cleanroom methodology. Cleanroom has been used for a few projects at IBM-SID and in a controlled experiment at the University of Maryland. One of the Cleanroom projects developed at IBM was the COBOL Structuring Facility (COBOL/SF) (References 7 and 8). COBOL/SF is a language product consisting of 80,000 executable lines of PL/I. The developers were hired directly out of college; thus, Cleanroom was the first methodology any of the developers used in a corporate environment. During development, the goal was to write simple designs and small procedures, with walkthroughs used as a substitute for verification. The emphasis was on proving the correctness of design, rather than finding errors. All released versions of COBOL/SF were characterized by high quality and productivity.

At the University of Maryland, students in two graduate-level software engineering courses participated in a controlled experiment (Reference 9). Student teams were organized so that each team would have comparable experience, and the differences in the classroom instruction were negligible. A programming language unfamiliar to all students was used to prevent a bias toward a team that had more experience in a particular language, and to control unauthorized execution of programs by developers. The teams in one class used the Cleanroom methodology for development, while teams in the second class were given the same development methodology, but also were given the opportunity to test. The submitted code from all teams was tested by the teaching assistant, who executed identical test cases for each team. The Cleanroom teams were able to apply the development methodology more successfully. They passed more test cases, fulfilled requirements more completely, generated less complex code, and had more inline documentation than the non-Cleanroom teams. This experiment indicated that an extra piece of technology (developer testing) did not necessarily lead to added success. At the conclusion of the experiment, most of the Cleanroom developers said that they would feel comfortable using Cleanroom again, although they missed the satisfaction of testing their code. Since the teaching assistant handled all testing responsibilities for the Cleanroom teams, the satisfaction level of Cleanroom test teams was not part of the project assessment.

SECTION 3—THE CLEANROOM CASE STUDY

3.1 THE SOFTWARE ENGINEERING LABORATORY

The SEL is sponsored by NASA/GSFC to investigate the effectiveness of software engineering technologies when applied to the development of applications software (Reference 13). It was organized in 1977 with the following goals:

1. To understand the software development process in a particular environment
2. To measure the effects of various development techniques, models, and tools on this development process
3. To identify and apply improved methodologies in the GSFC environment

The principals of the SEL are the Systems Development Branch of the FDD of NASA/GSFC, the Computer Sciences Department of the University of Maryland, and the Systems Development Operation of Computer Sciences Corporation. Over the past 14 years, the SEL has investigated numerous techniques and methods over dozens of projects in order to understand and improve the software development process in the FDD environment (References 1, 2, 13, 14, 15, 16, and 17).

The Cleanroom methodology was selected to be studied in the SEL for a variety of reasons. The SEL was interested in optimizing the software development effort and improving the effectiveness of software testing, thereby reducing the rework effort, which encompasses a significant portion of the FDD development effort (Reference 18). Cleanroom also displayed the potential to increase software quality and reliability without impacting productivity, an area of interest in any software environment. This experiment represented an opportunity to increase the SEL's understanding of Cleanroom in the FDD production environment, rather than an academic environment, by a group other than the originators of the methodology.

3.2 THE PROJECT AND ITS ENVIRONMENT

Development was carried out in the standard FDD environment, which is well understood based upon a large number of prior studies. The development process begins upon delivery of the requirements and specifications document. The requirements and specifications are generated by an organization separate from the development organization, and they adhere to a standard format familiar to both groups. The functionally oriented specifications are highly algorithmic and considered to be of good quality. However, due to the continually evolving characteristics of the spacecraft hardware and system architecture, the requirements and specifications require frequent modifications. On typical ground support systems,

approximately 300 formal inquiries typically are generated by the development organization, and 150 to 200 formal specification updates are required over the development life cycle.

The project selected for the Cleanroom case study at GSFC was the Coarse/Fine Attitude Determination Subsystem (CFADS) of the Attitude Ground Support System (AGSS) for the Upper Atmosphere Research Satellite (UARS). The size of the CFADS system was initially projected to be about 22,000 FORTRAN source lines of code (SLOC), which was approximately 12 percent of the entire AGSS. The development environment was an IBM mainframe running the multiple virtual storage (MVS) operating system, and the remaining subsystems of the AGSS were developed using the standard SEL development methodology. There were numerous "design drivers," factors related to the project that limited design options. The most significant was the interactive graphics system, which limited the ways some data could be bound to procedures. Development for the AGSS began in November 1987 and system testing was completed in March 1990. The development of CFADS began in January 1988 and system testing was completed in January 1990. System milestones were somewhat different in these projects, as illustrated in Figure 3-1. In addition, the CFADS consisted of six builds, with one build for each subfunction of approximately 5 thousand source lines of code (KSLOC), as opposed to one build for every 60 to 80 KSLOC, as is traditionally done on AGSS development in the FDD environment. As seen in Figure 3-1, the CFADS did not have a separate acceptance test process, but was integrated into the AGSS before acceptance testing began in March 1990.

The project was staffed by a total of seven different NASA/GSFC personnel. The project began with four developers, but dropped to three during design when one team member left NASA. The test team was composed of two people, but staffing was briefly increased to three when other work commitments prevented the original testers from allocating the planned level of effort. In addition to testing the system, the test team also served as library managers. All personnel were also working on other projects simultaneously during the CFADS effort. These other responsibilities would often take time and attention from the case study. Additionally, these other projects used methodologies other than Cleanroom, so staff members would often need to use two development methodologies during the same day.

Figure 3-2 compares the average level of experience for the Cleanroom team members with that of a typical SEL project team member. Most of the Cleanroom team's experience was as members of team efforts, with some time also spent managing projects. All of their development experience was with the standard SEL methodology in the FDD.

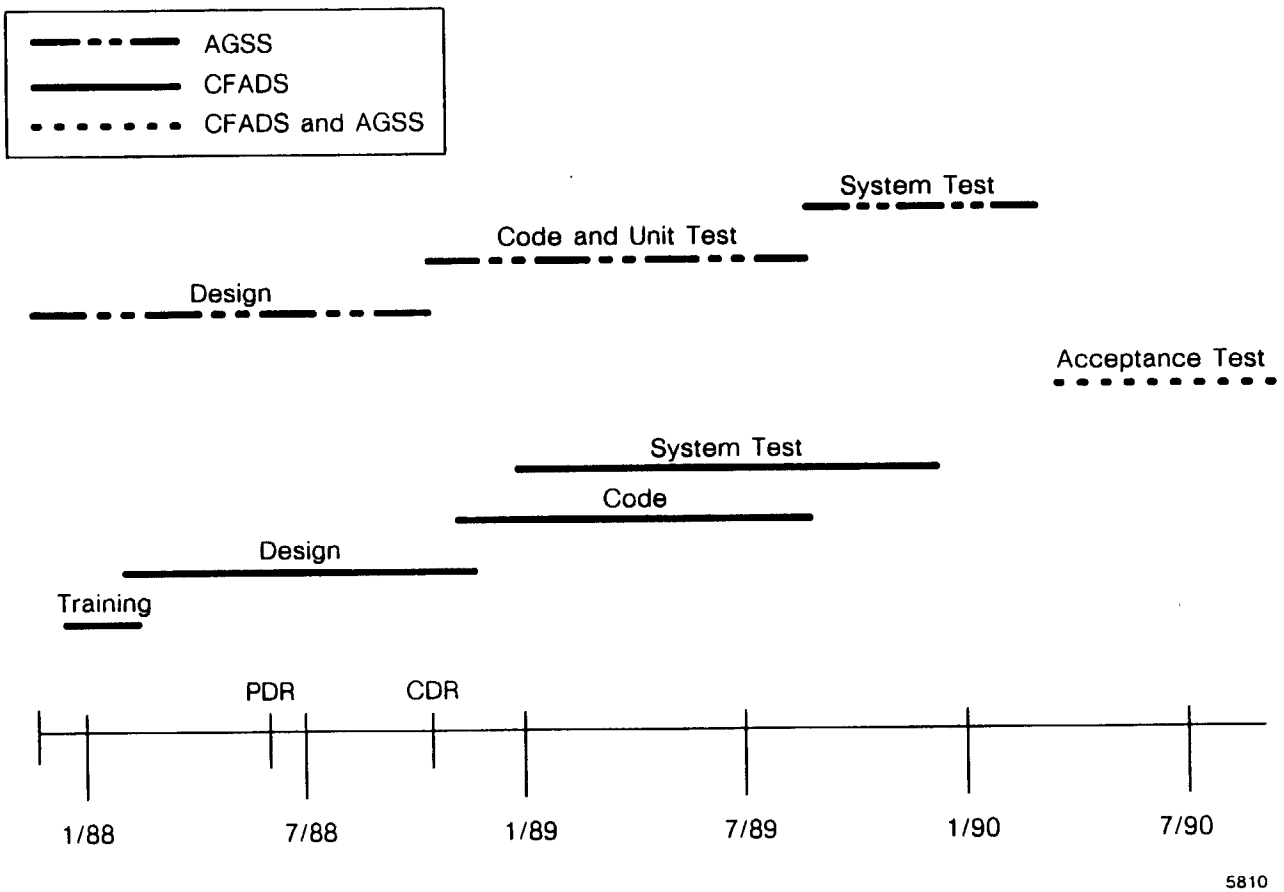


Figure 3-1. System Schedules for CFADS and AGSS

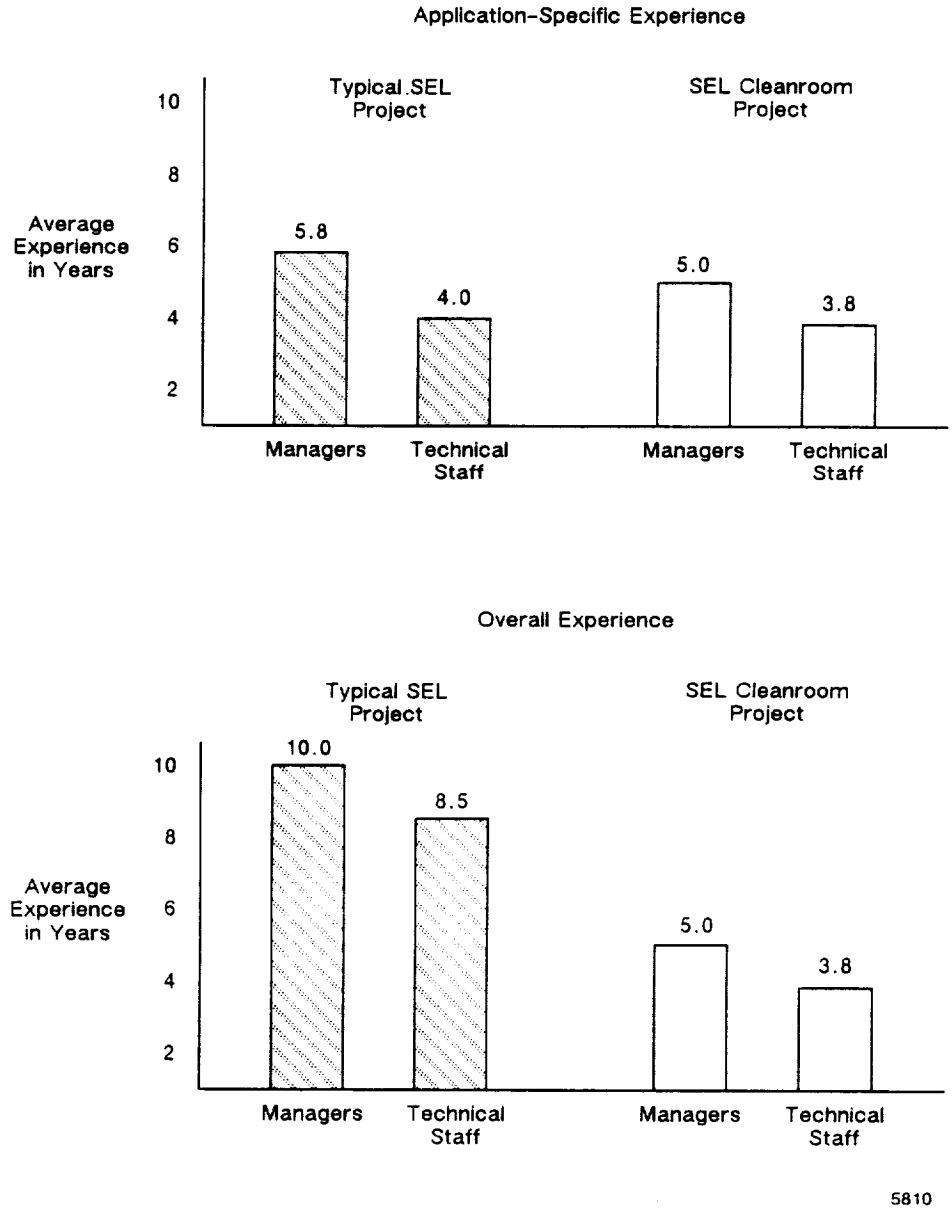


Figure 3-2. Experience Comparisons Between the SEL Cleanroom Project and Typical SEL Projects

3.3 THE GOALS

The major goals of the SEL experiment were as follows:

1. Assess the process used in the SEL Cleanroom model with respect to team structure, team activities, and effort distribution
2. Analyze the products of the SEL Cleanroom model and determine the impact on measures of interest, including reliability, productivity, overall life-cycle costs, and software quality
3. Analyze the residual products in the application of the SEL Cleanroom model, such as fault distribution, error characteristics, system growth, and computer usage

Additionally, other minor goals such as assessing the code-reading activity of the SEL Cleanroom model were defined during the life cycle.

3.4 EARLY CONCERNS

Four major concerns were expressed by management and development personnel very early in the project:

- The team's inexperience in the project's application area
- The impact of unstable requirements and specifications on the methodology
- Coordination with the main AGSS, which was developed without Cleanroom
- The psychological impact of the methodology on the team members

Despite the team's general experience in the FDD environment, this was the first time any of the project members had worked in the CFADS application domain. It was also the first time any member had used a methodology different than the typical SEL methodology.

As previously stated, the FDD environment is one in which modifications to the initial specifications are often necessary. This specification instability was of particular interest to the Cleanroom experiment in view of the emphasis on developing software "right the first time." There was also concern for the potential impact on rework effort and possible consequences on the timely delivery of builds for the test team. Frequently, the delivered specifications are not as detailed as might be desired, since assumptions are made based on past FDD projects, and items are negotiated between analysts and developers during early design.

Additionally, coordination between the CFADS and the remaining AGSS needed to be carefully planned, noting possible conflicts caused by using different

development methodologies. Formal reviews needed special consideration, as did common libraries used by both groups.

Finally, there was concern regarding the psychological impact of the team separation and limitation of each team's activities. The development team and testing team had specific tasks and guidelines to follow, and activities for each were carefully defined. There was concern that the developers would attain only minimal satisfaction from releasing a product that they would not be able to test or see executed. There was a parallel concern that the test team would experience a void by testing a product without having participated in the design and development.

3.5 DATA COLLECTION

Project data collection methods fell into four categories: forms, automated data collection, subjective data collection, and postdevelopment tools. The primary source of quantitative data was the set of standard SEL forms used in the FDD environment for all SEL projects (Reference 16). These forms address a wide variety of issues, from project estimation to personnel resources and change descriptions. In addition to the standard forms, a few new forms were designed to gather additional data and fulfill functions unique to this project. Information such as source code growth (system SLOC), source changes (module version updates), and computer usage (central processing unit, or CPU, hours) are automatically provided by several tools running in the host machine. To gather qualitative data, interviews and informal discussions were held. An observer from the University of Maryland familiar with the IBM-SID Cleanroom model was present for 10 to 15 hours a week at GSFC. The observer's task was to resolve questions pertaining to Cleanroom, tailor the methodology for the environment, ensure that the methodology was being used properly, and gather data and give real-time feedback to the developers and testers. At the conclusion of the project, standard tools were run to gather system statistics, including detailed component attributes and source code characteristics such as size and complexity.

SECTION 4—A TAILORED SEL CLEANROOM MODEL

The Cleanroom methodology had to be tailored to the FDD development environment, which is based on the waterfall model (References 1, 2, and 15). The tailored methodology needed to preserve the salient features of Cleanroom, but also needed to be easily adapted to the FDD environment. Some features of IBM-SID Cleanroom, which the developers and testers were not able to use because of a lack of experience in IBM-SID methods, needed to be reevaluated. Other characteristic SEL activities required modification to simulate IBM-SID Cleanroom features. Over a period of time, a version of Cleanroom was defined that seemed to best fit this environment. This period of time extended into development, as the initial Cleanroom description evolved to account for new problems encountered in the environment. The tailored methodology was referred to as the SEL Cleanroom model.

There were several significant differences between the SEL Cleanroom model and the standard SEL development methodology:

- Cleanroom testers and developers are on completely separate teams
- Cleanroom developers have no access to the mainframe computer for compilation and testing
- Cleanroom developers rely on code reading instead of unit testing to verify correctness of the software prior to system testing
- Cleanroom testers use a statistical testing approach

Table 4-1 highlights differences between the SEL Cleanroom model and the standard SEL development methodology. The SEL Cleanroom model used the standard SEL guidelines (References 1 and 2) for top-down design and development, but had more increments than a similar project using the standard methodology would have had. The project's requirements and specifications analysis process was strongly emphasized. The design process was a combination of SEL and IBM-SID activities, with attempts to use state machines, detailed program design language (PDL), and a generic design. High level designs were reviewed at various milestones. Team design reviews confirmed correctness during low level design, while redundant sequential code reading did the same during the coding phase. Sequential code reading differs from code reading by stepwise abstraction in that lines of code are read in physical order, not by functional hierarchy. The developers used sequential reading because it is commonly used in the FDD environment, and the training schedule did not offer sufficient time for skill development in reading by stepwise refinement.

All development was done at desks and on personal computers (PCs), with files transferred from the developers to the testers via floppy disks. The testers

Table 4-1. Process Comparisons Between the SEL Cleanroom and Standard SEL Development Models

	Organization	Design Quality Control	Code Quality Control	Testing Strategy
SEL Cleanroom	Separate development and test teams	Team design reviews	Sequential redundant code reading for verification	Statistical testing
Standard SEL	Single development and test team	PDL reading	Code reading and unit testing	Integration and system testing

5810

managed the libraries and uploaded code from the PC to the mainframe in order to build and test the system. This process ensured that the developers did not compile or unit-test their code, as they were not given any access to the mainframe. A statistical testing approach was to be used to validate the code, and failures were reported back to the development team. The developers then identified the error source and took the appropriate corrective action. Table 4-2 compares development and test team responsibilities in the SEL Cleanroom model.

4.1 TRAINING AND PREPARATION

Before project development began, both teams attended a 1-week tutorial on the Cleanroom methodology. The training was also attended by the project's supervisors. Lectures were given by Dr. Victor Basili of the University of Maryland, and Mr. Michael Dyer and Mr. F. Terry Baker, both of IBM-SID. Sessions discussed Cleanroom in general, and emphasized the IBM-SID method of software development. The classes consisted of lectures followed by question-and-answer sessions. Later, there were follow-up sessions by Mr. Dyer on statistical testing for the test team, and a presentation by Dr. Basili on verification by stepwise abstraction for the developers.

The purpose of the training sessions was twofold. First, it allowed the CFADS team the opportunity to learn as much as possible about the theory behind the methodology and how Cleanroom has been done in the past. Second, it served to reduce some of the misconceptions and apprehension of the team members. The cost of the training, in terms of CFADS team effort, was 4 percent of the total

Table 4-2. Comparison of SEL Cleanroom Team Responsibilities

Schedule	Project Phase	Development Team	Test Team
1/88	Training	Attend 1-week tutorial Attend additional sessions on design and code reading	Attend 1-week tutorial Attend additional sessions on test case generation
2/88–11/88	Design	Analyze requirements and specifications, submitting questions as needed Create high level design using abstract state machines Create low level design using PDL Review designs	None
	Pretest	None	Analyze requirements and specifications, submitting questions as needed Analyze specifications to understand functionality of the system Determine test items and passage criteria Determine system inputs and distributions
12/88–12/89	Code	Write code components Read code independently Submit code to testers	None
	Test	Respond to failures encountered by testers Correct and reverify code	Generate test cases Handle all configuration control activities Compile components Link components Execute test cases Validate results Return code to developers for correction

5810

hours expended on the project. Overall, the team members were happy with the tutorial, but felt that the addition of a laboratory exercise would have made the activity more effective. At the conclusion of the training sessions, two project members felt skeptical about proceeding with Cleanroom, two were cautiously confident concerning the methodology's application, and two members seemed confident that the methodology could be applied successfully in the FDD environment.

SECTION 5—APPLICATION OF THE CLEANROOM METHODOLOGY

5.1 PREDESIGN

The predesign phase, accounting for 30 percent of the total design effort, was given greater emphasis than on a typical FDD project. The major activity in predesign was requirements analysis, where the requirements and specifications were studied in order to understand the problem domain, resolve ambiguities, and make corrections. The FDD has a history of reusing requirements and specifications from previous projects, and these documents frequently assume an understanding of the systems developed for previous satellites as a base for describing the present project. Additionally, the documentation is often dynamic, and changes can occur during any phase of development. Since Cleanroom strives to develop software that is "right the first time," the requirements and specifications had to be more complete, so the requirements analysis activity needed to be more thorough. Team meetings were held to ensure that all development and test team personnel had a common understanding of the documents. Where ambiguities or errors were found, clarifications were requested. The result was an improved set of documentation for the developers and testers to work with.

5.2 HIGH LEVEL DESIGN

The remaining 70 percent of the design effort was divided into design creation (49 percent) and design review (21 percent). The high level design process attempted to emulate the high level design activities used at IBM-SID, encapsulating data and functions as state machines. Use of state machine concepts such as information hiding and data abstraction helped the developers in the design process, although there are few explicit signs of these concepts in the design and code of actual modules. One reason for the inability to actually implement state machines may have been the functional specifications, which strongly implied a function-oriented design. In addition, the developers' lack of experience in using state machines as a design representation may explain why they were able to use state machines conceptually but not concretely.

Because of problems in the early stage of development, such as the inability to build concrete state machines, some members of the development team again expressed concerns with the methodology. This may have been attributable to the fact that the methodology was still being tailored and was not in a clear, final form. This proved to be frustrating for some of the developers, as they could not see a complete life-cycle model and found it difficult to work on one step of the methodology when the next step potentially could be redefined. Many of the attitudes changed as the process became more complete and familiarity increased.

Tangible results also helped increase team confidence in the Cleanroom methodology.

Since the process was being tailored as the project progressed, it seemed logical to follow the IBM-SID Cleanroom model. Although the IBM-SID and SEL environments are similar in their use of disciplined life-cycle development activities, the level of formality employed during the activities is different and prevents a blanket use of the IBM-SID model. For example, the IBM-SID model bases its design review activities on the formal Fagan inspections (Reference 11), while the SEL Cleanroom model relied on informal peer review techniques familiar to the environment to verify design correctness.

The content and schedule of builds were also defined during this phase. The project leader, a member of the development team, worked together with the test team to determine the number of requisite builds and the functions that would be contained in each. This joint effort allowed the builds to be viewed from two perspectives and attempted to keep both teams sufficiently active during the entire implementation and testing phases. An effort was made to schedule builds so that the development team would complete a current build at approximately the same time as the test team finished with the previous one.

5.3 LOW LEVEL DESIGN

Since developers were not permitted to test their code, the developers inferred that the design had to be of high quality. This encouraged them to write more detailed PDL, and to make a strong effort to reduce their individual programming styles, opting for a common design to facilitate the design review process. Because developers used the state machine concept, the system was modularized differently than it normally might have been, although actual state machines were never implemented.

The team design review process was simple and successful. A few days before a scheduled review, the designs for a set of related modules were distributed to the other developers. The design for each module consisted of a standardized prolog, calling sequence information, and the PDL. The appropriate baseline diagrams also were distributed, along with any related COMMON blocks and general notes. The designs were studied individually by the other developers, which encompassed 15 percent of the design effort, and faults were noted. During the design reviews, which accounted for 6 percent of the design effort, faults were discussed and corrections suggested. A set of modules would remain in the design review process until the development team determined that there was no need to review the design again. All designs required at least two reviews.

As a result, the assessment of a representative sample of the CFADS design showed an average of 18.4 faults/KSLOC found during the design reviews. This is a lower bound, as the designer corrected other faults before and after reviews.

Since the SEL does not normally track design faults, it is impossible to evaluate this parameter in relation to the existing SEL methodology. The distribution of the faults by types along with raw count and percentage of faults found in each activity appears in column A of Table 5-1. Qualitatively, the developers were confident in the completeness and accuracy of the final designs, and felt they knew and understood the entire system better than they normally would have on other projects.

Table 5-1. Fault Distribution by Quality Control Activity

Fault Type	Activity				Totals
	A	B	C	D	
	Design Reviews	Code Reading	Compilation	Testing	
FORTRAN Syntax	0.0%	4.0%	100.0%	0.0%	7.0%
Control Flow	20.0%	8.0%	0.0%	12.0%	12.0%
Interface	24.0%	17.0%	0.0%	34.0%	20.0%
Data Initialization	1.0%	5.0%	0.0%	12.0%	4.0%
Data Declaration	45.0%	19.0%	0.0%	5.0%	25.0%
Data Use	0.0%	32.0%	0.0%	23.0%	19.0%
Computation	10.0%	9.0%	0.0%	11.0%	9.0%
Displays	0.0%	6.0%	0.0%	3.0%	4.0%
Total Number of Faults	542.0*	883.0	74.0	175.0	1,674.0
Percent of Total Faults Found	32.0%	53.0%	4.5%	10.5%	

5810

*Projected

5.4 CODING

The more thorough design process allowed the developers to concentrate solely on coding the system during the coding stage, without major impacts due to an incomplete design. Through the first few builds, slightly more than half the effort in the coding phase was spent writing code, with the rest in code reading. More time was shifted to reading code as failures were found by the testers and corrected by the developers, who were no longer writing as much new code. The final relationship

was that 48 percent of the total coding effort was expended in code writing, and 52 percent was spent in code reading.

The code review process was similar to the design review process. Strict coding guidelines were established and adhered to by the developers, although opinions regarding the content and flexibility of the guidelines varied among the development team. This standardization aided the code reading process by forcing all code to appear similar in style, and made the process of mapping the code to the PDL easier. Related modules were coded and listings given to the readers along with necessary references. The readers read the code independently in a sequential manner, marked faults, and suggested corrections. Code was returned to the developer, corrected, and given back to the readers to be reread. Code was ready to be delivered to the testers only when no faults were found by the readers. Most modules required two or three iterations of the code reading process.

Redundant sequential code reading resulted in significant numbers of faults being found and corrected before the code was actually tested. The readers found and corrected an average of 30 executable faults/KSLOC while code reading. An additional 10.4 nonexecutable faults/KSLOC were found and corrected. Nonexecutable faults are faults found in the commentary for the procedure, which could not have been found by the testers. These corrections helped to make prologs more consistent and complete in relation to the code, and should make maintenance efforts easier. As with the design reviews, these numbers are lower bounds, as each developer found and corrected additional faults before and after his code was read by the readers. Surprisingly, of the 30 executable faults/KSLOC found by the readers, only 28.5 percent were found by both readers, with no consistent pattern between readers in terms of relative effectiveness. This leads to the conclusion that for this project two readers were more effective than one would have been, and that sequential code reading allowed readers to read the code in different ways.

One of the early concerns for the project was the impact of specification changes, a common occurrence in the FDD environment. However, this concern was never any more of a factor than is typically found in other SEL projects. Subjectively, the developers felt making changes of any type was actually easier in the Cleanroom environment due to the detail and accuracy of the PDL. All software changes were reverified, and appropriate PDL updates were made and checked.

It should also be noted that all reused modules for this experiment were reused in their entirety. Any reusable module that required changes was rewritten to follow the design and coding standards and passed through the review process as a new routine.

The actual distribution of faults by type, along with the total number of faults and percentage per activity, is shown in Table 5-1, column B. Qualitatively, the

developers had high confidence in their code. These results also confirm the effectiveness of reading, as found elsewhere in the literature (References 3 and 9).

5.5 PRETEST

As was shown in Figure 3-1, the testing phase ran concurrently with the development phase. The testing process began with a requirements analysis. This activity was performed with the developers, to enable the specifications to be reviewed from two different perspectives and to promote a common understanding of the specifications by both groups. Following the requirements analysis, interaction between the testers and developers ended, and did not resume until the actual testing activities began. In addition to studying the requirements, pretest activities included identifying potential test items, determining system usage profiles, preparing test data, generating test cases, mapping test items to the test cases, and generating expected results for each test case. The pretest phase involved 32 percent of the total testing effort.

Test items defined the lowest level of functionality to be evaluated during the testing process. A test item consisted of one or more functions that mapped directly to the specifications. Identification of the specific test items was a subjective process that relied on the testers' impressions concerning the level of functionality that could be verified. Each test item was then mapped to the build where it would be available for validation. As previously stated, the test team was directly involved in determining the content of each build, so an attempt was made to reasonably spread test items throughout all builds.

These activities were simplified by the functional orientation of the specifications. Alterations to the specifications, however, which continued throughout the testing activity, resulted in changes to the test items identified. A great deal of care was needed to ensure a mapping of the test items to the specifications throughout the testing effort. While no interaction was permitted between the testers and developers during pretest activities, it was later determined that communication with the developers may have been useful in defining the type and level of diagnostic output desired to support the testing activities.

The manner in which the system was expected to be used by an operator (the usage profile) was determined by direct observation of operators executing similar systems and by previous experiences of the testers. This information, combined with specification outlines on user interaction with the system, was used in an attempt to develop a statistical profile of system inputs and options. These distributions were then used as the basis for generating test cases to closely simulate actual operator use of the system. While test cases were generated based on usage profiles, it was later determined that the variation from test case to test case was minimal. This was due to the limited number of inputs required by the system (few option selections), the types of inputs required (strong dependencies on data and results from outside interfaces), and the low level of user interaction required

by the system (the sequential operation of the system). Because of the limited variation in the test cases, many were later abandoned during the testing activities, as they would have given little additional insight about the overall system quality.

Once test cases were generated for the system, they were mapped to the subset of test items associated with the build being evaluated. Based on the total test item coverage by all test cases, the testers were free to generate test cases independent of the system usage profile. These test cases isolated critical test items not covered in the other test cases, or test items that may rarely be exercised but are nonetheless critical.

The final pretest activities included preparation of test data for the test cases and generation of expected results for the test items in each test case. Because the Cleanroom build schedule required data earlier than in traditional development efforts, the unavailability of the necessary data forced these activities to be postponed until the test cases were executed.

5.6 TEST

The remaining 68 percent of the testing effort comprised three distinct activities: configuration control, system integration, and test case execution/verification.

Traditional SEL configuration control practices were followed by maintaining a controlled library of all source code for the system on the host machine. However, because developers were not permitted to access the mainframe, additional configuration control responsibilities included the maintenance of a separate controlled library on a PC. The mainframe control library was used to integrate the system, while the PC control library was used to distribute components back to the developers, upon their request, for modification. After being placed into the PC control library, new or modified components were uploaded to the mainframe control library. The extra effort required to manage two libraries was minimal but did require careful procedures to ensure consistency between the two. The test team was also required to maintain a library of stubs on the mainframe for use in the system integration.

The system integration activity was viewed as a two-step process. The first step involved compiling all new and modified components as they were received and uploaded by the testers. The results of this compilation were the first opportunity to assess the effectiveness of the development method. Of the 101 FORTRAN subroutines, which averaged 258 SLOC, 62 percent compiled on their first attempt. Among those that failed, all but one compiled on the second try. Although all faults were obviously due to incorrect FORTRAN usage, over 90 percent were directly attributable to typographical mistakes such as missing commas or misspelled variable names. As might be expected, BLOCK DATA routines displayed greater success. Ninety-one percent compiled on the initial attempt, and all that failed the first time compiled on the second attempt. Overall, 70 percent of all

compilable units compiled successfully on the first try. This result compares favorably to the 63 percent figure of first compilation success for the first increment of COBOL/SF at IBM-SID. The second step of the system integration process was the rebuilding of the system to support test case execution. The need to completely rebuild the system from scratch each time, rather than linking in only the modified components, became evident early in the testing effort. This was due to a dependency on other development groups, separate from the CFADS effort, for common portions of the system. Communication was often inadequate and occasionally resulted in outdated versions of the common components being included in the CFADS integration process. To promote rapid turnaround when identifying, isolating, and correcting faults in the system, the load module was rebuilt frequently, often two or more times per week. There was also a greater dependence on system stubs, since the entire system was to be integrated and executed much earlier in the development process than would be required during a traditional development effort. This required the test team to maintain the stubs library on the host and include it in the system integration process.

Test case execution and verification accounted for the majority of the test team's efforts. Activities consisted of preparing test data, generating expected results, executing test cases, analyzing and reporting results, and supporting the development team's fault isolation efforts. As stated earlier, test data preparation and expected results generation were postponed until test activities began. For the generation of simulated test data, the test team relied heavily on an external system developed solely for that purpose. Because data were needed much earlier in the Cleanroom effort than in traditional SEL efforts, there were conflicts with the development schedule of the data simulation system. This resulted in requests for data which were frequently only partially satisfied or exhibited analytic inconsistencies throughout testing activities. The impact to the testers was most strongly felt in the difficulties caused in generating expected results prior to test case execution, and the inability to exercise all test items associated with a test case.

The test team executed the system based on a test case setup. During execution, a log was kept that identified the run, documented alterations to the test case, and described abnormalities observed. All outputs generated by the system, including copies of the interactive screens displayed, were collected and attached to the log for future analysis. Alterations to the test cases were required when there were deficiencies in the test data or in the functionality of the system, or when modifications to the specifications affected the mapping of test items. Functional deficiencies were primarily the result of software that was designed to provide a certain level of functionality but failed to do so because of source code errors or limitations with system interfaces. The limitations often were linked to conflicts with the schedule of the interfacing group responsible for developing those components. To expedite the testing process, the test team adjusted cases appropriately to test around known deficiencies. As these problems were addressed, the required functionality was added within the system. The net result for the testers was an

inability to execute test cases as specified and to exercise various test items associated with a test case. The final solution was to abandon many of the test cases in the interest of the project's schedule and resources (possibly justified, since there was little variation from test case to test case), and to postpone until the last build of the system the verification of many of the test items. In several instances, it was not until then that the required data became available, the required functionality was provided, or the modifications to the specifications were completely understood to allow for accurate assessment of the code.

Once a test case had been executed, the results were analyzed and failures reported to the developers. Analysis of the results required the generation of expected results for each test item and a comparison with the computed results. Generation of expected results was usually quite tedious and time consuming. The comparison of expected results with computed results occasionally resulted in requests from the testers to modify the diagnostic output in the system. A log kept during analysis activities documented any concerns, questions, and problems identified and provided a compilation of which test items passed, failed, or could not be verified. In addition, a test item status sheet was maintained to track the status of each test item. All failures were documented using a Software Failure Report (SFR) form, which identified the test case run and described the observed failure. To expedite the process, developers were often informed of a run's results informally and potential faults were investigated prior to the generation of the SFRs.

When a potential failure was identified, the developers attempted to isolate the problem and define a solution, with assistance from the testers as required. On occasion, developers requested additional runs of the system to assist in their fault isolation efforts, and on a single occasion it became necessary to build a test version of the system which was modified directly to isolate the cause of a failure. Several problems were also traced to the external interfacing groups.

The testing process uncovered 3.3 errors/KSLOC. Errors represent the number of software changes attributed to error corrections on data collection forms. Errors are tracked only for changes that occur after code is placed under configuration control. These errors resulted in a total of 175 faults, whose distribution by type are listed in column D of Table 5-1. Most faults were identified and corrected the day the failure was reported, although some took several days to correct, and a few took weeks to correct. Once a fault was isolated, the first correction implemented typically resulted in the removal of the failure. As mentioned earlier, while a fault remained in the system, an attempt was made to test around the problem.

After correction or modification was made to the system, test cases or variations of test cases were executed to verify the modification. In the case of a failure correction, the test case that caused the failure was reexecuted. For other modifications due to specification changes, test cases were usually created by the testers to exercise the specific test items affected. Testing activities continued until all test items were verified.

SECTION 6—PRELIMINARY ANALYSIS

The fundamental finding was that the Cleanroom methodology could be used successfully in the FDD environment, as the CFADS team demonstrated.

The completed system contained approximately 31 KSLOC divided into 101 FORTRAN subroutines, 33 BLOCK DATAs, 33 COMMON blocks, and 2 NAMELISTs. Combined with the 2.9 KSLOC for the graphic displays, which were not developed using Cleanroom, the entire CFADS totaled 34 KSLOC. Figure 6-1 compares the growth history of the CFADS with similar systems developed using the traditional SEL approach. As a result of the increased design effort, code appeared later and system growth progressed much more quickly. Because of the incremental development and code reading processes, which forced related modules to be read by the developers and submitted to the testers together, the SEL Cleanroom growth in Figure 6-1 appeared as a step function.

The personnel effort distribution during the development process for the Cleanroom project is somewhat different than the effort expended on other FDD projects. As Figure 6-2 shows, the Cleanroom effort spent more time in Design and Other (management, meetings, etc.) activities and less time in the Coding area. It should be noted again that almost one-third of the Testing effort was spent on pretest activities, which resulted in only 18 percent of the entire project's effort being spent actually executing test cases and finding and fixing defects.

Productivity for the project is approximately 4.9 SLOC per staff hour, from predesign through system test. This number compares favorably with the average of 2.9 SLOC per staff hour typically found in this environment.

During the testing phase, the error rate of the system was 3.3 errors/KSLOC, compared to the 6 errors/KSLOC typically found in the FDD environment. In actuality, the Cleanroom error figure is artificially high for comparison purposes, as some errors that would typically have been found during unit test when using another methodology are not encountered until the system test process when using Cleanroom. Typically, the SEL does not formally track errors found during unit testing.

In analyzing the number of faults found during the various activities of the life cycle, 85 percent were found and corrected before any code came under configuration control. Table 5-1 showed the fault type breakdown by activity. Additionally, nearly 90 percent of all faults were removed before the first test case was executed. The majority of the faults found throughout development and test were data declaration problems, followed in frequency by data use and interface issues.

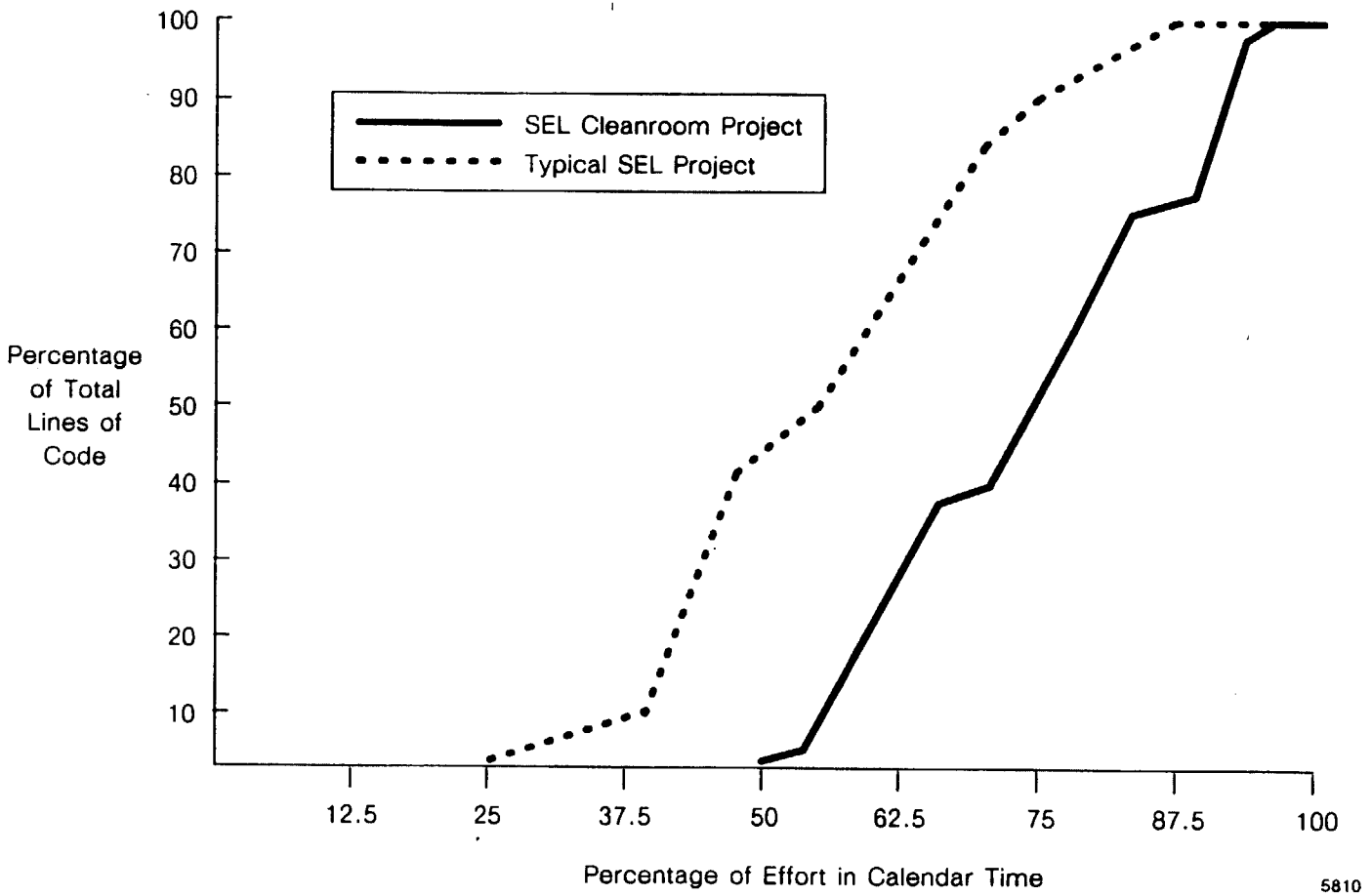


Figure 6-1. Growth of System in Calendar Time Through System Testing

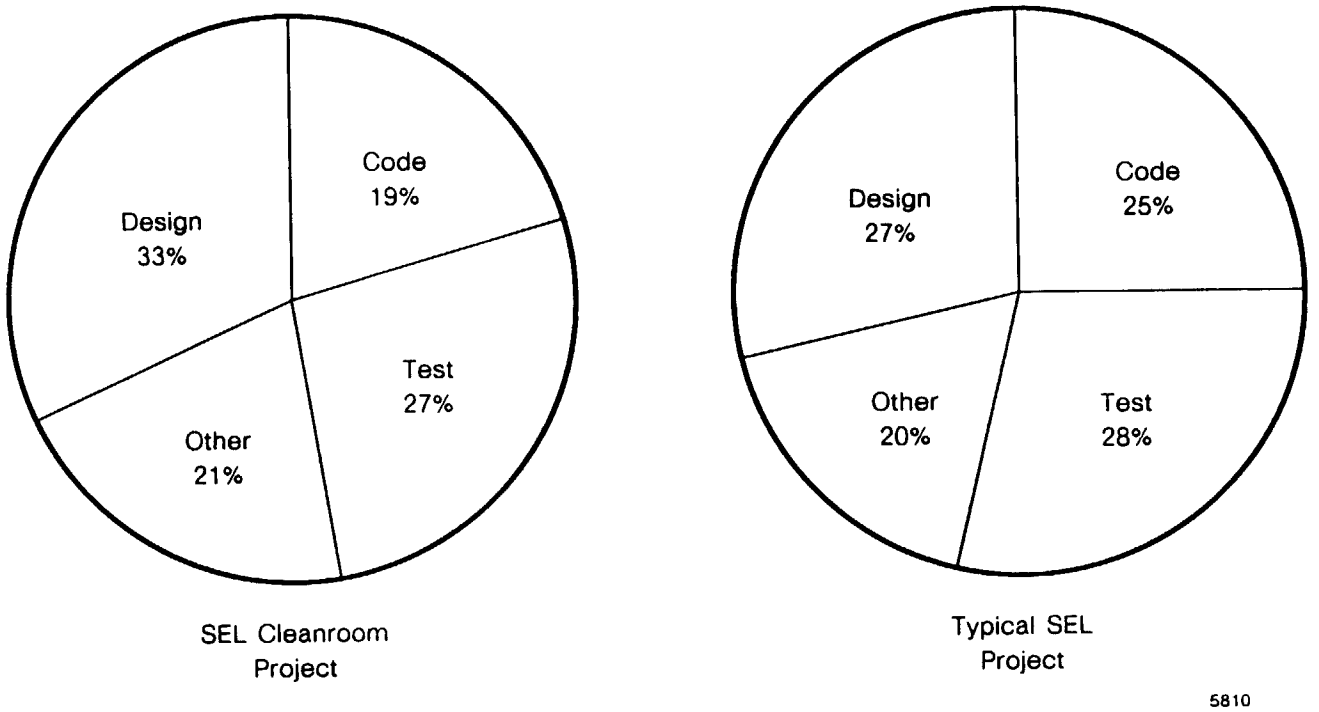


Figure 6-2. Effort Comparison Between the SEL Cleanroom Project and Typical SEL Projects Through System Testing

The early concerns about using Cleanroom on the project and in the FDD environment did not limit Cleanroom's applicability.

1. The team's inexperience in the application area was countered by a more complete Requirements Analysis process. The increased early emphasis enhanced the development and test teams' familiarization with the project domain.

2. The impact of an unstable requirements and specifications environment was lessened by a concentrated effort in the team review processes, which encouraged detailed PDL and accurate inline code documentation, and made later stages of the Cleanroom methodology less prone to the effect of requirements and specifications changes.

3. Coordination with the main AGSS, which was developed without Cleanroom, did present some minor problems. Since some software was common to both groups, the Cleanroom team was directly affected when modifications were made or errors were discovered. Furthermore, because the Cleanroom team accessed the common libraries earlier in the development phases, they were often the first group to uncover an error, and thereby felt the greatest impact. Additionally, communication between the groups was not as frequent or thorough as necessary during the initial phases of the projects. Formal mechanisms to aid the communication process were added and proved helpful. The Cleanroom team also needed to make some adjustments in the format of their design materials in order to participate in the combined formal design reviews.

4. The application of methodology and team structure seemed to have minimal impact on the project personnel's level of professional satisfaction. While team members did note that there was an adjustment in the importance of activities, they also indicated that the overall success of the project was still of primary interest. Overall, the psychological impact of the methodology was not a factor in the project's degree of success.

Preliminary analysis of the first two goals of the experiment, understanding the Cleanroom process and product, has been performed. Additional analysis with respect to these goals remains. The results of this analysis will be used to refine the SEL Cleanroom model for use on future projects. This packaging will attempt to incorporate the successes in this project, while learning from failures or mistakes made in the planning or execution of the methodology's application. Finally, the refinement must be based upon the concepts that can be applied effectively to the SEL Cleanroom model. For example, previous studies indicate that reading by stepwise abstraction is more consistent with the methodology and may prove, with proper skill development, more effective. A redesign of the entire system may permit a broader application of state machines. The result of this process will be an appropriate SEL Cleanroom model that is more effective in the environment.

SECTION 7—CONCLUSION

Analysis of the Cleanroom case study shows that the methodology can be applied in this environment. There are indications of an increase in developer productivity and product quality. Compared to typical SEL activities, Cleanroom produced a lower failure rate (3.3 errors/KSLOC versus 6 errors/KSLOC), and there are indications of a more complete and consistent set of inline code documentation. In terms of effort, there is a different distribution of phase effort activity (more time in Design and meetings, and less time in Coding) and a different growth profile in terms of lines of code developed (code appears much later).

The Cleanroom case study generated important lessons for application on future projects. It is clear that the development effort is a true team effort, and all team members must be able to function as such. Additionally, there is visible benefit in the adoption of strict design and coding standards for all development team members to follow. These standards should be defined before a project begins, reducing debate among team members on personal preferences. It appears that the actual standards are not as significant as the consistent application of the standards by all participants. It is also recognized that the test team and development team, while performing distinct functions, need frequent communication. Both groups need to be aware of specification changes, and the developers require input from the testers regarding desired diagnostic data for test item validation. Configuration control and the transfer of modules between teams also requires a well-coordinated effort.

Subjectively, project personnel felt that the activities associated with the methodology are quite similar to activities typically done in the FDD environment. The difference in using Cleanroom is evident in the goals and discipline level associated with the activities. All members of both teams indicated that they would be willing to use the methodology on future projects.

There are also a few specific issues that will affect the refinement of the methodology in the FDD environment. Since developers do not have access to the mainframe, they must rely heavily on their personal knowledge of the development environment. Situations may arise where more time is expended tracking down a solution through documentation or a colleague's recommendation than would have been spent in a trial-and-error session on the mainframe. It is also unclear whether the design phase should be completed for all builds before any coding begins (to isolate design inconsistencies), or if the design and code for each build should be completed before design on the next build commences (to produce testable software earlier).

More analysis of the Cleanroom project is planned, as well as applications on future systems. Upon completion of acceptance testing, the appropriate final

adjustments to the data on effort, productivity, and quality will be made. Comparisons between the Cleanroom process and products and the process and products typically used in the FDD will continue. All the data gathered will be further analyzed to reach a better understanding of the causes of the results and the contexts in which the data should be interpreted. An appropriately tailored Cleanroom model for the FDD environment will evolve using the results of this experience as a basis.

GLOSSARY

AGSS	Attitude Ground Support System
CFADS	Coarse/Fine Attitude Determination Subsystem
COBOL/SF	COBOL Structuring Facility
CPU	central processing unit
FDD	Flight Dynamics Division
GSFC	Goddard Space Flight Center
IBM-SID	IBM Systems Integration Division
IC	integrated circuit
KSLOC	thousand source lines of code
MTTF	mean time to failure
MVS	multiple virtual storage
NASA	National Aeronautics and Space Administration
PC	personal computer
PDL	program design language
SEL	Software Engineering Laboratory
SFR	Software Failure Report
SLOC	source lines of code
UARS	Upper Atmosphere Research Satellite

REFERENCES

1. Goddard Space Flight Center, Software Engineering Laboratory, SEL-86-001, *Programmer's Handbook for Flight Dynamics Software Development*, March 1986

2. --, SEL-81-205, *Recommended Approach to Software Development*, April 1983

3. P. A. Currit, M. Dyer, and H. D. Mills, "Certifying the Reliability of Software," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1, January 1986, pp. 3-11

4. IBM, Systems Integration Division, TR. 86.0002, *An Approach to Statistical Testing for Cleanroom Software Development*, M. Dyer, August 12, 1983

5. --, TR. 86.0004, *A Design Method for Cleanroom Software Development*, M. Dyer, August 15, 1983

6. --, TR. 86.0003, *Software Validation in the Cleanroom Development Method*, M. Dyer, August 19, 1983

7. M. Dyer, "A Formal Approach to Software Error Removal," *The Journal of Systems and Software 7*, 1987, pp. 109-114

8. R. C. Linger and H. D. Mills, "A Case Study in Cleanroom Software Engineering: The IBM COBOL/SF Structuring Facility," *Proceedings of COMPSAC '89*, Chicago, Illinois, October 5-7, 1988, pp. 10-17

9. R. W. Selby, V. R. Basili, and F. T. Baker, "Cleanroom Software Development: An Empirical Evaluation," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 9, September 1987, pp. 1027–1037

10. R. Linger, H. D. Mills, and B. I. Witt, *Structured Programming: Theory and Practice*. Reading, Massachusetts: Addison Wesley, 1979

11. M. E. Fagan, "Design and Code Inspections to Reduce Errors in Programs," *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 182-211

12. IBM, Systems Integration Division, TR. 86.0008, *Cleanroom Testcase Generator*, J. F. Gerber, June 18, 1986

13. Goddard Space Flight Center, Software Engineering Laboratory, SEL-81-104, *The Software Engineering Laboratory*, February 1982

14. V. R. Basili and R. W. Selby, Jr., "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 12, December 1987, pp. 1278-1296

15. University of Maryland, UMIACS-TR-89-84, *Lessons Learned in the Transition to Ada from FORTRAN at NASA/Goddard*, C. Brophy, August 1989

16. Goddard Space Flight Center, Software Engineering Laboratory, SEL-87-008, *Data Collection Procedure for the Rehosted SEL Database*, October 1987

17. D. M. Weiss and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 2, February 1985, pp. 157-168

18. Goddard Space Flight Center, Software Engineering Laboratory, SEL-86-006, "Determining Software Productivity Leverage Factors in the SEL," *Proceedings of the Eleventh Annual Software Engineering Workshop*, F. McGarry, S. Voltz, and J. Valett, December 1986

19. IBM, Systems Integration Division, TR. 86.0009, *Cleanroom Reliability Analysis Software*, J. F. Gerber, August 3, 1987

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-77-004, *A Demonstration of AXES for NAVPAK*, M. Hamilton and S. Zeldin, September 1977

SEL-77-005, *GSFC NAVPAK Design Specifications Languages Study*, P. A. Scheffer and C. E. Velez, October 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 and W. A. Taylor, July 1986

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

SEL-79-003, *Common Software Module Repository (CSMR) System Description and User's Guide*, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 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-80-003, *Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study*, T. Welden, M. McClellan, and P. Liebertz, 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-81-008, *Cost and Reliability Estimation Models (CAREM) User's Guide*, J. F. Cook and E. Edwards, February 1981

SEL-81-009, *Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation*, W. J. Decker and F. E. McGarry, March 1981

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 From 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-81-107, *Software Engineering Laboratory (SEL) Compendium of Tools*, W. J. Decker, W. A. Taylor, and E. J. Smith, 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-81-205, *Recommended Approach to Software Development*, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

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 I*, July 1982

SEL-82-007, *Proceedings From 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, F. E. McGarry, and M. G. Rohleder, October 1983

SEL-82-806, *Annotated Bibliography of Software Engineering Laboratory Literature*, M. Buhler and J. Valett, November 1989

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-83-006, *Monitoring Software Development Through Dynamic Variables*, C. W. Doerflinger, November 1983

SEL-83-007, *Proceedings From 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-84-001, *Manager's Handbook for Software Development*, W. W. Agresti, F. E. McGarry, D. N. Card, et al., April 1984

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 From the Ninth Annual Software Engineering Workshop*, November 1984

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., May 1985

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

SEL-85-006, *Proceedings From 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 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, October 1986

SEL-86-006, *Proceedings From 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-87-008, *Data Collection Procedures for the Rehosted SEL Database*, G. Heller, October 1987

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

SEL-87-010, *Proceedings From 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-89-003, *Software Management Environment (SME) Concepts and Architecture*, W. Decker and J. Valett, August 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-89-101, *Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 1)*, M. So, G. Heller, S. Steinberg, K. Pumphrey, and D. Spiegel, February 1990

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

4Agresti, 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

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

1Bailey, 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

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

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

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

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)

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

1Basili, 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

1Basili, 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

3Basili, 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

4Basili, 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

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

1Basili, 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

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

3Basili, 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 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

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

5Basili, V., and H. D. Rombach, "T A M E: Tailoring an Ada Measurement Environment," *Proceedings of the Joint Ada Conference*, March 1987

5Basili, V., and H. D. Rombach, "T A M E: Integrating Measurement Into Software Environments," University of Maryland, Technical Report TR-1764, June 1987

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

7Basili, 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

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

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

Basili, V. R., and R. W. Selby, Jr., *Comparing the Effectiveness of Software Testing Strategies*, University of Maryland, Technical Report TR-1501, May 1985

3Basili, 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

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

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

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

3Basili, 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

[5]Brophy, C., W. Agresti, and V. 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., and W. Agresti, "Resolving the Software Science Anomaly," *The Journal of Systems and Software*, 1987

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

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

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

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

3Card, 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

1Chen, 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

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

2Doerflinger, 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

5Doubleday, D., "ASAP: An Ada Static Source Code Analyzer Program," University of Maryland, Technical Report TR-1895, August 1987 (NOTE: 100 pages long)

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

Hamilton, M., and S. Zeldin, *A Demonstration of AXES for NAVPAK*, Higher Order Software, Inc., TR-9, September 1977 (also designated SEL-77-005)

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

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

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

6Mark, 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

5McGarry, F., and 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

7McGarry, 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

3McGarry, 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

National Aeronautics and Space Administration (NASA), *NASA Software Research Technology Workshop (Proceedings)*, March 1980

3Page, 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

5Ramsey, C., and V. R. Basili, *An Evaluation of Expert Systems for Software Engineering Management*, University of Maryland, Technical Report TR-1708, September 1986

3Ramsey, 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

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

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

6Rombach, 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

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

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

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

4Seidewitz, 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

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

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

7Sunazuka. 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

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

Turner, C., G. Caron, and G. Brement, *NASA/SEL Data Compendium, Data and Analysis Center for Software*, Special Publication, April 1981

5Valett, J., and F. 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

3Weiss, 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

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

1Zelkowitz, 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

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

6Zelkowitz, 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

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

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

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.