SOFTWARE ENGINEERING LABORATORY SERIES SEL-88-001

SYSTEM TESTING OF A PRODUCTION ADA® PROJECT: THE GRODY STUDY

NASA
National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SYSTEM TESTING OF A PRODUCTION ADA® PROJECT: THE GRODY STUDY

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 and Analysis 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 include this document.

The contributors to this document include

Jeffrey Seigle (Computer Sciences Corporation)
Linda Esker (Computer Sciences Corporation)
Ying-Liang Shi (Computer Sciences Corporation)

Single copies of this document can be obtained by writing to

National Technological Information Service
5285 Port Royal Road
Springfield, Virginia 22161

NASA Scientific and Technical Installation Facility
P.O. Box 8757
BWI Airport, Maryland 21240

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

ABSTRACT

The use of the Ada®[1] language and design methodologies that utilize its features has a strong impact on all phases of the software development project lifecycle. At the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC), the Software Engineering Laboratory (SEL) conducted an experiment in parallel development of two flight dynamics systems in FORTRAN and Ada. The teams found some qualitative differences between the system test phases of the two projects. Although planning for system testing and conducting of tests were not generally affected by the use of Ada, the solving of problems found in system testing was generally facilitated by Ada constructs and design methodology. Most problems found in system testing were not due to difficulty with the language or methodology but to lack of experience with the application.

[1]Ada® is a registered trademark of the U. S. Government Ada Joint Program Office (AJPO).

TABLE OF CONTENTS

Section 1 - Introduction 1-1

Section 2 - Definition of the System Test Phase 2-1

Section 3 - Writing the System Test Plan 3-1

Section 4 - Conducting the System Test 4-1

4.1 Impact of Ada Features 4-1
4.2 Tools 4-2

Section 5 - Errors Discovered During System Testing . . 5-1

5.1 Source of Data 5-1
5.2 Classes of Errors Discovered During System Testing 5-1
5.3 Solving Errors Found During System Testing 5-3

5.3.1 Isolating Errors 5-3
5.3.2 Repairing Errors 5-4
5.3.3 Nonlanguage Differences 5-7

Section 6 - Lessons Learned 6-1

Section 7 - Acknowledgments 7-1

References

LIST OF ILLUSTRATIONS

Figure

2-1 Effort for Testing on the Ada Project. 2-2
2-2 Effort for Testing on the FORTRAN Project. . . 2-3
5-1 Classes of Errors Found During System Testing. 5-2
5-2 Time to Isolate Errors Found in System Testing. 5-5
5-3 Time to Repair Errors Found in System Testing. 5-6

SECTION 1 - INTRODUCTION

Ada®[1] is not just a new programming language but a part of a major advance in software engineering technology that includes new approaches for all phases of the software development lifecycle. This paper, one of a series of reports examining each project phase [Brophy 1987, Brophy 1988], evaluates the impact of the use of Ada when compared with FORTRAN in the system test phases of two projects.

The Software Engineering Laboratory (SEL) of the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) conducted an experiment involving the parallel development of a software system in both the Ada and FORTRAN programming languages.[2] NASA/GSFC and Computer Sciences Corporation (CSC) were cosponsors of the experiment, which was supported by personnel from the three SEL participating organizations: NASA/GSFC, CSC, and the University of Maryland. The chief goals of the study were to characterize the development lifecycle of a large project when Ada is used as the implementation language with a design methodology that can take advantage of its features and to determine the impact of the use of Ada on reusability, reliability, maintainability, productivity, and portability.

Two teams each developed a Gamma Ray Observatory (GRO) satellite dynamics simulator from the same specifications. One team used FORTRAN as the target language with a conventional

[1]Ada® is a registered trademark of the U. S. Government Ada Joint Program Office (AJPO).

[2]The acronyms were Gamma Ray Observatory (GRO) Dynamics Simulator in Ada (GRODY) for the Ada project and GRO Dynamics Simulator in FORTRAN (GROSS) for the FORTRAN project.

design methodology, which is the usual approach for this type of application. The other team used Ada, with an object-oriented design methodology developed at NASA/GSFC [Seidewitz, Stark 1986]. NASA uses the GRO dynamics simulator to test and to evaluate GRO flight software under conditions that simulate the expected in-flight environment as closely as possible [Agresti 1986]. By the end of the system testing phases, the teams had produced 39,767 lines of FORTRAN and 128,046 lines of Ada, where lines of code are the total number of physical lines including exe- cutable code and nonexecutable code, comments, and blank lines. Although these figures give a rough idea of the comparative sizes of the two efforts, they do not give a precise basis for comparison of the effort required for development in the two languages [Firesmith 1988].

Data were collected directly from team members and from a database maintained by SEL. Members of both teams who participated in system testing were interviewed and asked about their expectations, actual findings, problems, solutions, and opinions. Team members also completed forms throughout the projects describing their effort levels and changes to code, and that information was entered into the SEL database. Presented data are taken from the database, and other sources are referred to since much of the data has already been reported.

SECTION 2 - DEFINITION OF THE SYSTEM TEST PHASE

Ada unit testers performed some integration before system testing officially began. System testing and unit testing effort overlapped considerably. The team members reported their hours on Personnel Resource Forms (PRFs) and attributed hours to specific activities. Figures 2-1 and 2-2 show the weekly efforts for unit testing and system testing on the two projects.

In the FORTRAN project, a clear delineation exists between effort attributed to system testing and effort attributed to unit testing although they overlap slightly. In the Ada project, participants were performing system test work at the same time as unit test work, and the overlap is considerable. This overlap plus team members' comments suggest that the line between unit testing and system testing was blurred on the Ada project.

When the data from PRFs giving time attributed specifically to system testing is considered and this effort is calculated as a percentage of total project effort, the Ada project used 11.3 percent of its effort on system testing, and the FORTRAN project used 8.91 percent. In addition to defining activities by the hours attributed directly to them, each project phase had a formal start and end date. Regardless of attributed activity, the sum of all effort occurring during the system test phases was found and the effort during system test determined as a percentage of all effort on the entire project. Of the total effort on the two projects, the portion used during the system test phase was 22.8 percent on the Ada project and 17.9 percent on the FORTRAN project. The standard allotment for system testing is 20 percent in the flight dynamics area. The Ada project system testing phase was not grossly out of proportion, but a general conclusion cannot be drawn about the language since

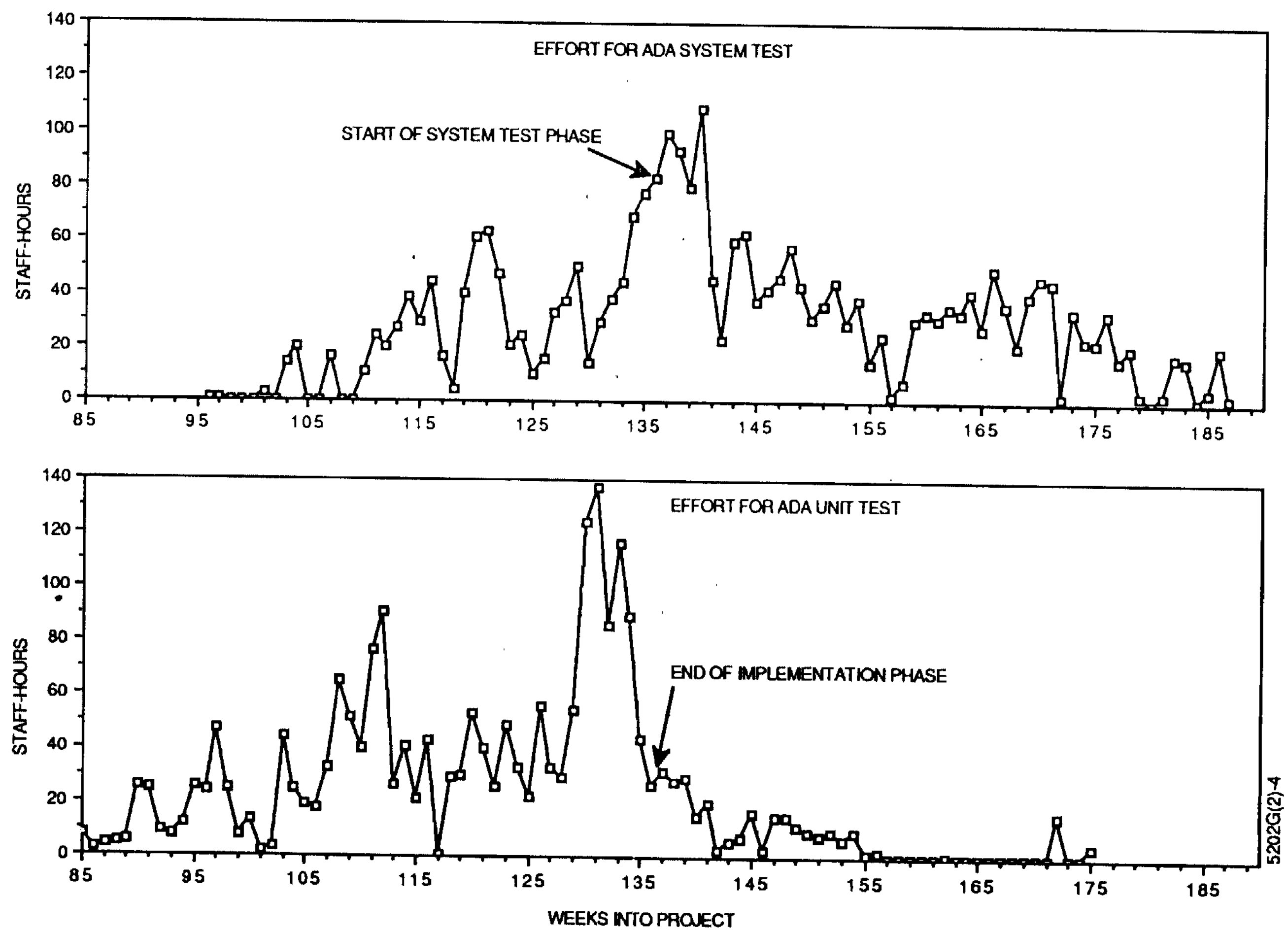

Figure 2-1. Effort for Testing on the Ada Project

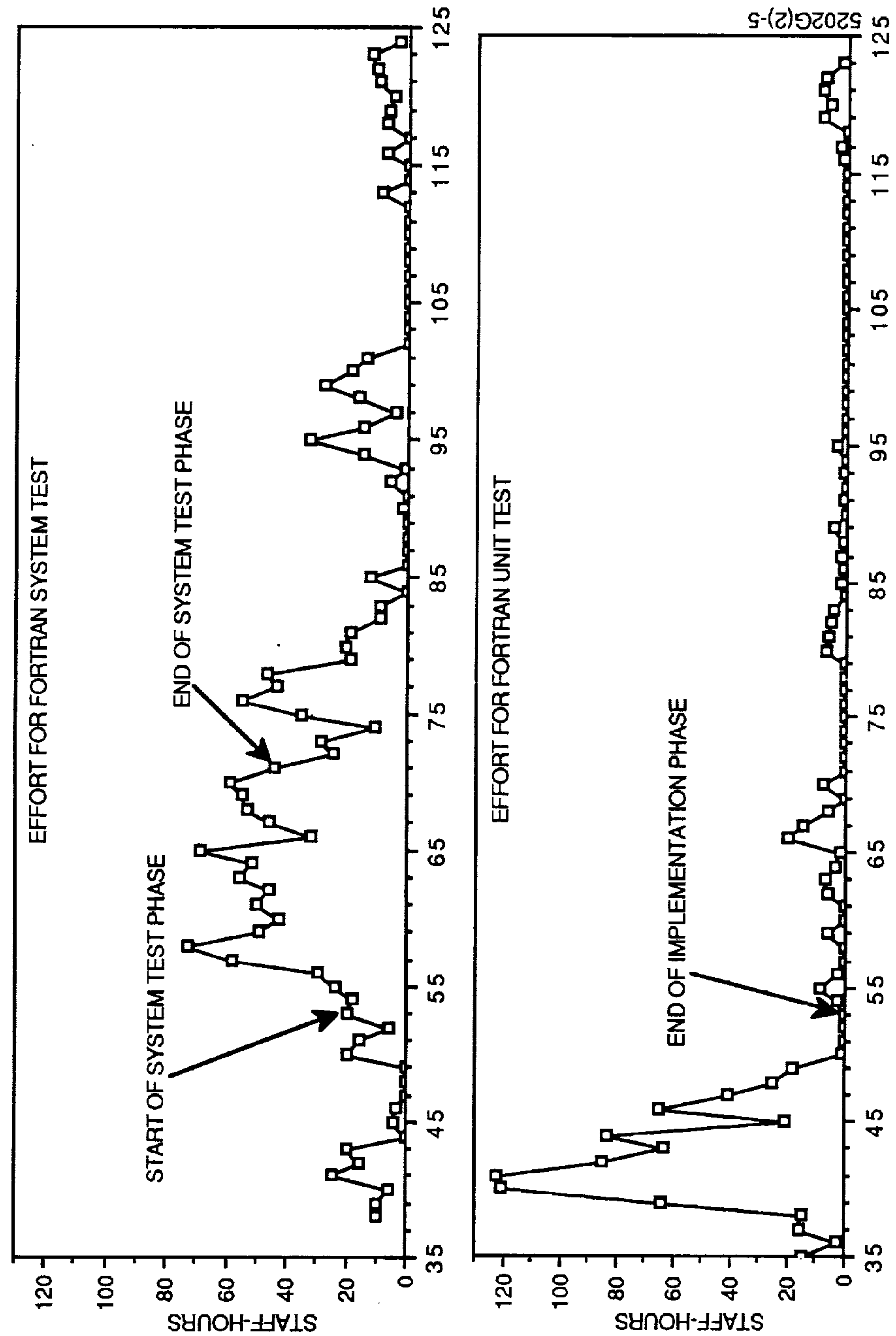

Figure 2-2. Effort for Testing on the FORTRAN Project

other variables, such as greater training time for the Ada project and overlap of activities other than system testing in the system testing phases, exist.

SECTION 3 - WRITING THE SYSTEM TEST PLAN

The author of the system test plan for the Ada experiment [Stark 1987] said that the plan was based on the FORTRAN plan being developed in parallel [Garrick]. He found no need for special consideration because of the use of Ada or the object-oriented design methodology; this is consistent with the idea that system test plans in this environment are generally written to test against functional specifications, which ideally do not depend on the implementation language. However, because the Ada team did not have the same schedule constraints as the FORTRAN team, they defined more tests--31 compared to 14 for the FORTRAN team.

SECTION 4 - CONDUCTING THE SYSTEM TEST

4.1 IMPACT OF ADA FEATURES

Conducting system tests was not generally different for the Ada project than for the FORTRAN project. The system test teams usually did not need to examine internals to run tests and to evaluate results. However, the Ada team did find a few Ada features that needed special attention.

One case in which an Ada feature was an issue was in inducing conditions that would cause Ada exceptions to be raised. Many times this inducement was relatively easy, such as deletion of a required file; other times it was not, i.e., for exceptions that flagged conditions that may not be introduced externally such as division by zero. Although some exceptions were difficult to test overall, the team felt that they aided in comprehensive error handling.

The Ada test team reported that it was difficult to coordinate concurrent tasks for testing although this coordination can be challenging regardless of the language. The Ada language offers tasking but FORTRAN does not, so the Ada team took advantage of the ease of tasking more than the FORTRAN team [Brophy 1988]. Although concurrency was easier to design and implement in Ada, the team reported that setting up tests and diagnosing problems were more difficult. They agreed, however, that these problems were not peculiar to Ada but would be found in any system using concurrency and that since tasking was easier to implement, Ada provided a net advantage when using concurrency.

The FORTRAN project used a form of tasking that was supported by the operating system; the method did not provide true concurrency but a series of tasks whose execution was controlled by logic within the application software. Only one task was active at any given time. The FORTRAN team did not report any unusual problems in testing a system with this architecture and attributed only one or two errors to difficulties stemming from their tasking approach.

Occasionally, the Ada rename feature caused confusion during debugging sessions. This was attributed to the debugger's failure to incorporate the rename feature rather than to a difficulty in the language. When the debugger did not recognize the name used to rename a variable, programmers would query the debugger for the value of a variable, and if it were a name used to rename another variable, they could not get the value. This problem was discussed with a member of the Digital Equipment Corporation (DEC) Ada team; she said she was unaware of the problem and would treat it as a bug. She believed the problem should be fixed and that it might even be resolved in the next release of the debugger.

Although the Ada exception handling, tasking, and rename features required special attention and caused some problems, none was a major roadblock, and the team felt the power added by these features outweighed the difficulties.

4.2 TOOLS

Ada development is still relatively new, so despite many excellent offerings of Ada tools, their availability is neither as great as nor as widely known as the tools for the more mature FORTRAN environment. The Ada team developed software on a DEC VAX/VMS system,[1] and DEC offers tools that are compatible with Ada for use on the VAX [Schultz 1988]. The DEC symbolic debugger and Code Management System (CMS) were the tools used in system testing. When asked what other tools would have been useful, one team member

[1]DEC, VAX, and VMS are registered trademarks of Digital Equipment Corporation.

suggested that the DEC Performance and Coverage Analyzer (PCA) would also have been helpful; other team members responded that they did not suggest that other tools were necessary because they had no information about other available tools. Although no clear need was identified for additional tools, more information regarding the availability of other Ada-oriented testing tools would have been helpful.

The FORTRAN team also developed their system on a VAX and used only a debugger. They felt that tool was sufficient for their testing.

SECTION 5 - ERRORS DISCOVERED DURING SYSTEM TESTING

5.1 SOURCE OF DATA

All team members recorded information for each software change on a Change Report Form (CRF). The CRF describes the type of change. Data were examined for changes with a type of error correction. When the type of change is an error correction, the form also describes the class of error, the source of the error, the time to isolate the error, and the time to implement the change. This data was entered into the SEL database.

5.2 CLASSES OF ERRORS DISCOVERED DURING SYSTEM TESTING

Brophy noted that Ada developers in the experiment found unit testing to be more difficult for Ada [Brophy 1988]. Since the team found isolation of Ada units to be difficult, unit testing usually involved combinations of units rather than single units. The team members reported that the types of errors discovered through this method of unit testing were often mismatched data interfaces and con- flicting assumptions between internal components, which are errors more typical of those discovered in later testing phases of conventional FORTRAN projects. Although this in- tegration increased unit testing effort, the team believed that it made system testing easier. The team members also found that the semantic checking performed by the Ada com- piler uncovered mismatched calling sequences at compile time that would not have been found in FORTRAN until run time.

The errors described on the CRFs were divided into the following classes: computational, data value (usage of variables), data initialization, external interface, internal interface, and logic. Figure 5-1 shows the distribution of errors for each project by class of error.

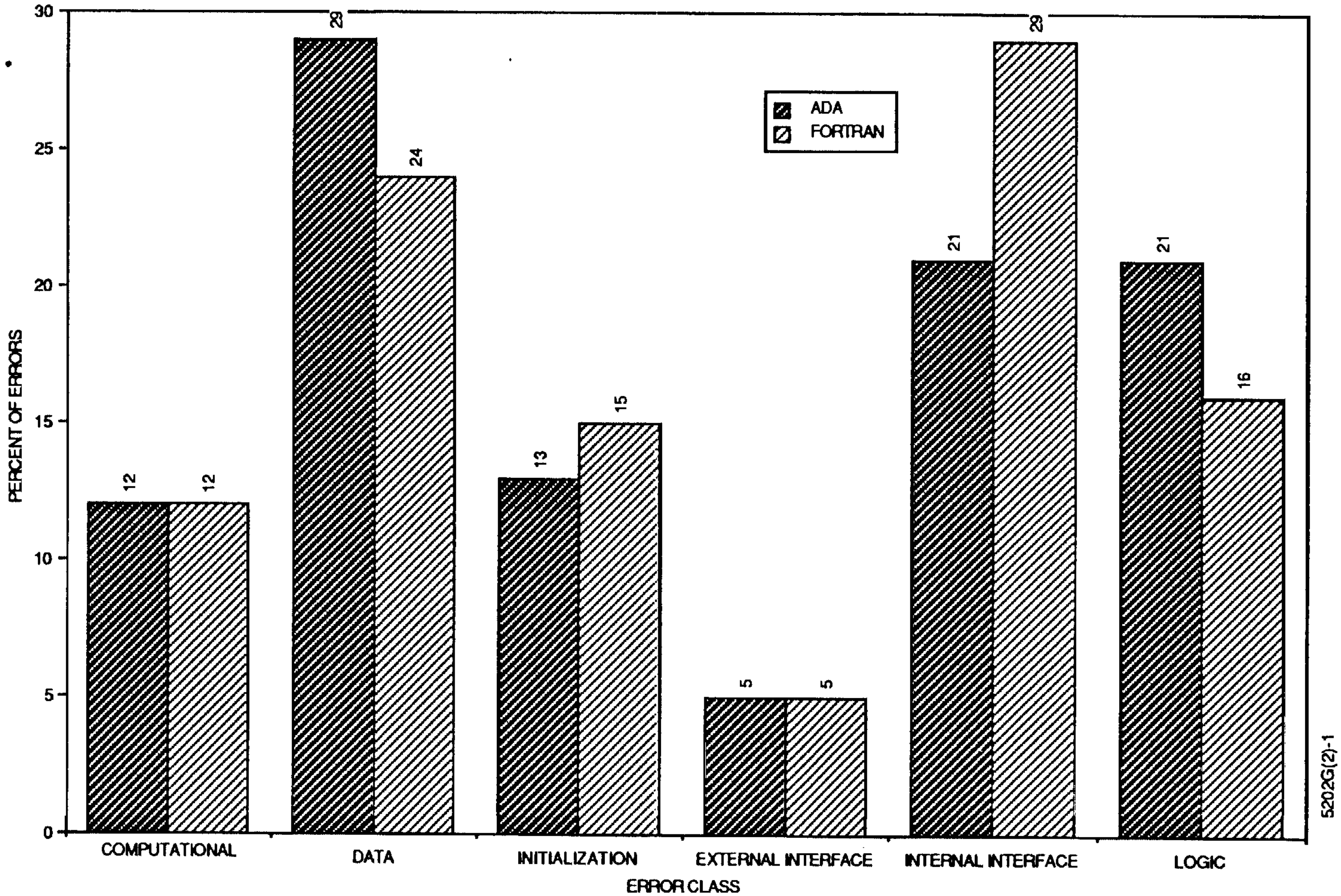

Figure 5-1. Classes of Errors Found During System Testing

Of the total errors found during system testing, internal interface errors accounted for 21 percent in the Ada project and 29 percent in the FORTRAN project. However, this apparent difference is not statistically significant.

Because the Ada system was not intended to become operational, managers placed a lower priority on it when assigning effort to it, and it was difficult to get support that the team thought they needed from analysts who had strong backgrounds in the specific application. The team attributed most errors to misinterpretation of the specifications, such as errors in mathematical computation, rather than design errors or coding errors.

The design for the FORTRAN project was largely based on stable designs of similar systems already developed, and approximately 36 percent of the code was reused from other systems. No precedent existed for an Ada system of the type being developed; therefore, the design was new, and only 2 percent of the code was reused from previous systems [McGarry, Agresti 1988]. This difference in reuse is another variable that may have affected the error profile of the FORTRAN project.

5.3 SOLVING ERRORS FOUND DURING SYSTEM TESTING

5.3.1 ISOLATING ERRORS

The Ada system test team reported that in some ways the Ada code was easier to debug than similar FORTRAN systems because the design methodology controls access to related data as opposed to the FORTRAN implementation that exploited large COMMON blocks with little control over data access. For the same reason the scope of effect of software errors was more limited in the Ada implementation. The team reported that they generally found errors easily in the Ada implementation because of the program structures that are enforced by the language. However, the times to isolate causes of errors indicate that the Ada team actually spent more time solving errors than the FORTRAN team. The CRFs described time to isolate an error defined as the time it took for the responsible developer to isolate an error and does not include the time to determine who is the responsible developer. As shown in Figure 5-2, both teams solved most of their errors in less than 1 hour; however, the FORTRAN team solved 82 percent of errors in less than 1 hour, and the Ada team solved only 58 percent in less than 1 hour. When the first two categories are combined, they show that the proportion of errors solved in less than 1 day were similar for both projects: 94 percent for the FORTRAN project and 96 percent for the Ada project.

The Ada compiler does semantic checking that spots some errors that would not be found until testing in a FORTRAN system, so the proportion of easier to solve errors may have been reduced in the Ada system test phase.

The development team found the readability of Ada as compared to FORTRAN, in part due to more rigid coding standards, to be a clear advantage in debugging, except where long variable names appeared in complex mathematical expressions. In some instances, this problem was easily solved by the judicious selection of variable names and by renaming variables with long names when they were used in such expressions.

5.3.2 REPAIRING ERRORS

As shown in Figure 5-3, once the problems were isolated the FORTRAN team needed slightly less time to make the changes. Although the Ada compiler is more comprehensive and detects some errors earlier, it often requires recompilation of unchanged units that are dependent on changed units. Compilation errors can occur even in unchanged units being recompiled. This recompilation was sometimes a significant effort, particularly because of the configuration of the Ada

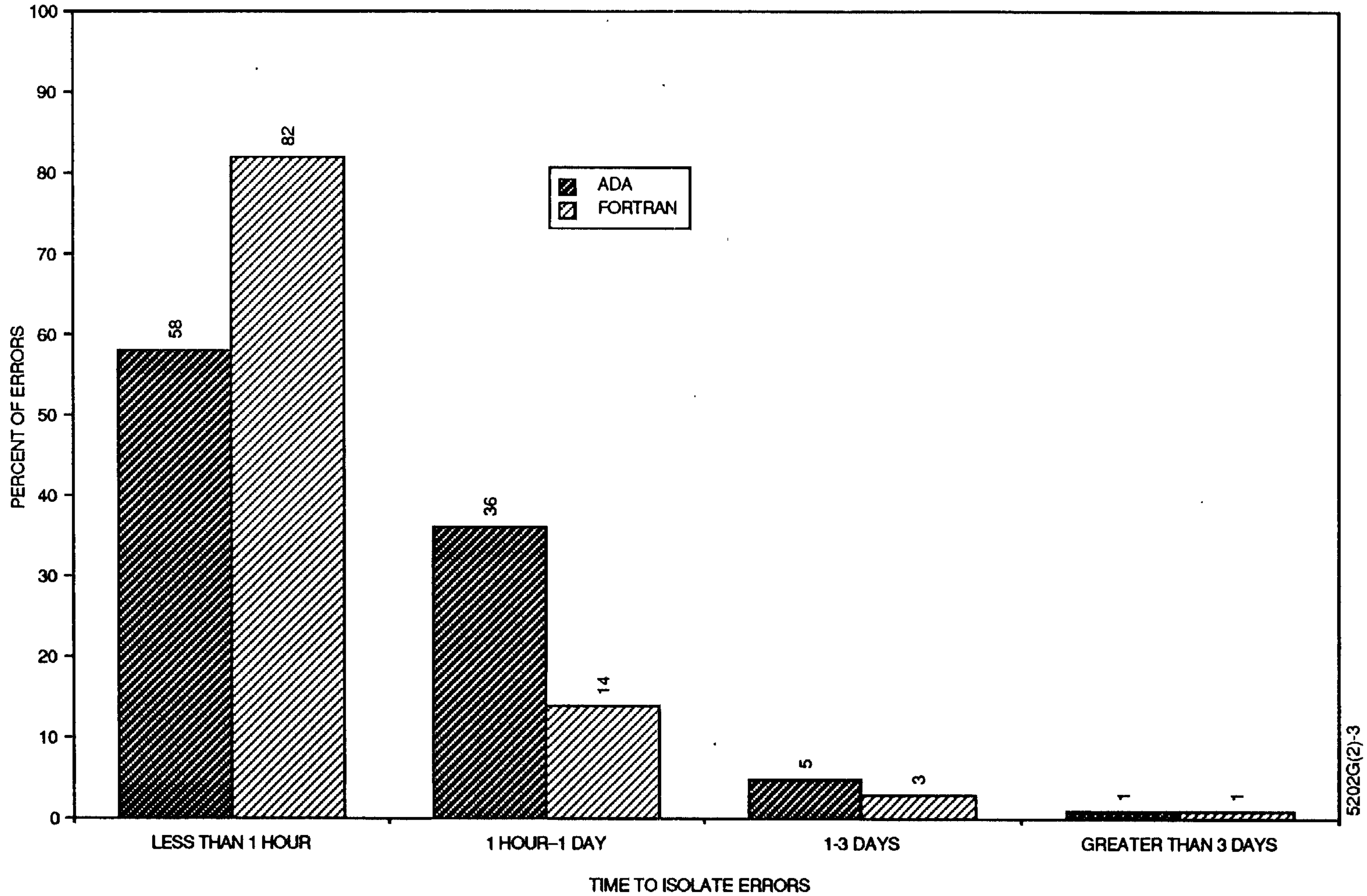

Figure 5-2. Time to Isolate Errors Found in System Testing

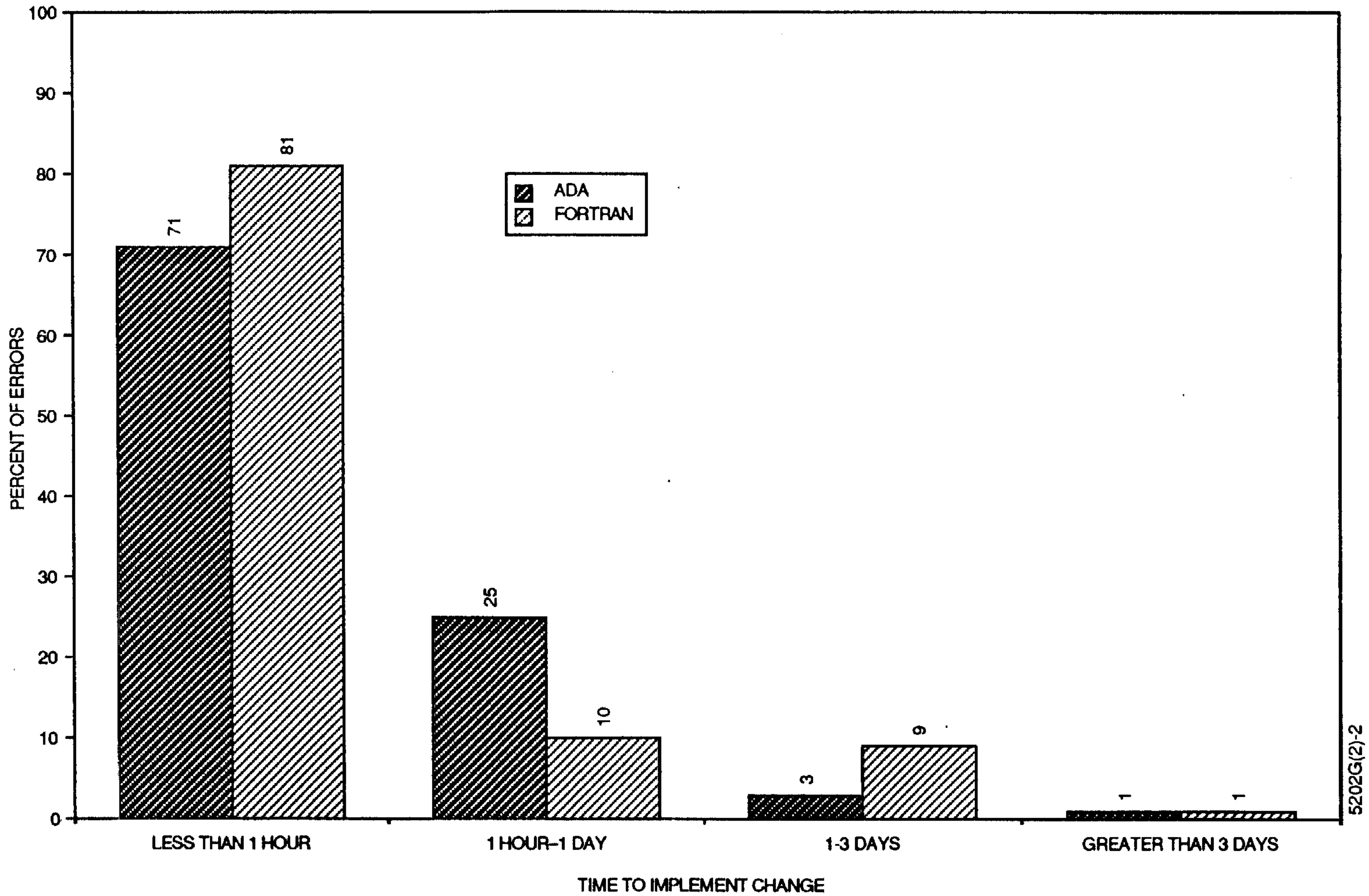

Figure 5-3. Time to Repair Errors Found in System Testing

system. The Ada implementation decision of nesting versus library units had a ripple effect in debugging at the system test level; a great deal of recompilation was necessary before some coding changes could be tested. This complaint also surfaced in the implementation phase [Brophy 1988].

5.3.3 NONLANGUAGE DIFFERENCES

The FORTRAN team members had greater experience in both the language in which they were working and in the particular application [McGarry, Agresti 1988]. The Ada team consistently reported that the single biggest obstacle to effective system testing was the lack of availability of people who were intimately familiar with the technical aspects of the application. Although the Ada team members were experienced software developers, having on the average more years of software development experience than the FORTRAN team members [McGarry, Agresti 1988], their lack of experience with the specific application made it more difficult for them to detect and solve errors than for the FORTRAN team. These differences in personnel background may account for the ability of the FORTRAN team to isolate and correct errors with equal or less effort than the Ada team, despite the language advantages described by the Ada team.

SECTION 6 - LESSONS LEARNED

All personnel involved in both projects believed that software development in Ada with an appropriate design methodology is a different experience than the conventional FORTRAN development of similar systems.

Team members have subjectively attributed the many differences between the results of the two projects to various differences in languages and design methodologies, but too many variables exist to be able to clearly assign all effects in the system test phases to their causes. Some general statements can be made about what was learned.

Preparing for system testing and executing the tests was not affected by the programming language. The system test plan for Ada was essentially the same as the plan for FORTRAN. When running the tests, testers were not concerned with the language.

A good repertoire of tools is important. The extra effort needed to resolve confusion and software problems due to the error in the debugger shows the impact of even minor problems with tool software. An organization can most effectively use its human resources if it has a good tool set and actively promotes the use of the tools.

Ada may reduce some types of errors. Team members consistently reported that the Ada compiler detected many of their interface errors even before testing began. Objective data neither confirms nor contradicts this assertion, but it is a reasonable one since the Ada compiler checks for correct interfaces, and the FORTRAN compiler does not.

Ada may be easier to debug. Team members reported that Ada's better readability and the organization of the team's design allowed them to find errors more easily than the

FORTRAN team. Objective data neither confirms nor contradicts this assertion, partly because of uncontrolled experimental variables.

Recompilation of Ada units can have a significant cost. Recompilation issues should be considered short of compromising the integrity of the design. It is important to control the design to avoid unnecessary dependencies that will require extensive recompilation in testing phases.

Definition of test phases for Ada systems is not well defined. Testing Ada software at the system level is not as clearly defined as was presumed at the outset of the project. Although the system test plan itself was nearly the same as for the FORTRAN project, and it was clear which tests were to be designated system tests, it is very difficult to draw a hard line between unit testing and system testing. Testing Ada software must be approached differently than testing systems where functions can be easily isolated for testing.

The differences that could clearly be attributed to the use of Ada were generally positive ones, and Ada features with negative aspects were either redeemed by their advantages or easily mitigated. As the Ada environment matures and as developers get more experience, we expect improvements to occur in the building and testing of systems built with Ada and object-oriented design when compared to methods that are still considered conventional.

SECTION 7 - ACKNOWLEDGMENTS

The authors thank Frank McGarry of NASA/GSFC and the Ada and FORTRAN teams for their effort and cooperation.

REFERENCES

Agresti, W., et al., "Designing with Ada for Satellite Simulation: A Case Study," Proceedings of the First Annual Symposium on Ada Applications for the NASA Space Station, Houston, Texas, June 1986

Brophy, C., et al., "Lessons Learned in Use of Ada-Oriented Design Methods," Proceedings of the Joint Ada Conference, Arlington, Virginia, March 1987

Brophy, C., et al., "Lessons Learned in the Implementation of a Large Ada Project," Proceedings of the Washington Ada Technical Conference, March 1988

Firesmith, D., "Mixing Apples and Oranges, or, What Is an Ada Line of Code Anyway," Ada Letters, September/October 1988, vol. VIII, no. 5, pp. 110-12

Garrick, J., GROSS System Test Plan (unpublished)

McGarry, 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, Kaila-Kona, Hawaii, January 1988

Schultz, B. J., "Industry Use of a Multi-Language Software Development Environment," Proceedings of the Sixth National Conference on Ada Technology, March 1988

Seidewitz, E. and M. Stark, General Object-Oriented Software Development, National Aeronautics and Space Administration, SEL-86-002, August 1986

Stark, M., Gamma Ray Observatory (GRO) Dynamics Simulator in Ada (GRODY) System Test Plan, Computer Sciences Corporation, December 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-81-102, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide Revision 1, P. Lo and D. Wyckoff, July 1983

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

SEL-81-106, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, May 1985

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-81-203, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, P. Lo, June 1984

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-82-003, Software Engineering Laboratory (SEL) Data Base Reporting Software User's Guide and System Description, P. Lo, August 1983

SEL-82-004, Collected Software Engineering Papers: Volume 1, 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-82-606, Annotated Bibliography of Software Engineering Laboratory Literature, S. Steinberg, November 1988

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

SEL-84-002, Configuration Management and Control: Policies and Procedures, Q. L. Jordan and E. Edwards, December 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-87-005, Flight Dynamics Analysis System (FDAS) Build 3 User's Guide, S. Chang et al., October 1987

SEL-87-006, Flight Dynamics Analysis System (FDAS) Build 3 System Description, S. Chang, October 1987

SEL-87-007, Application Software Under the Flight Dynamics Analysis System (FDAS) Build 3, S. Chang et al., October 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 and Y. Shi, November 1988

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

SEL-RELATED LITERATURE

Agresti, W. W., Definition of Specification Measures for the Software Engineering Laboratory, Computer Sciences Corporation, CSC/TM-84/6085, June 1984

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[3]Basili, 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

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

[5]Basili, V. and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions on Software Engineering (in press)

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

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

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

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

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

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

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

[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

[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

[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

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

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

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

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

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

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

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

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

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

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

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

[5]McGarry, F. 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

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

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

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

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

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

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

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

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

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

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

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

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

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

[5]Valett, 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

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

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

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

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

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

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

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.