SOFTWARE ENGINEERING LABORATORY SERIES SEL-85-002

ADA TRAINING EVALUATION AND RECOMMENDATIONS FROM THE GAMMA RAY OBSERVATORY ADA DEVELOPMENT TEAM

(NASA-TM-88593) Ada TRAINING EVALUATION AND RECOMMENDATIONS FROM THE GAMMA RAY OBSERVATORY Ada DEVELOPMENT TEAM (NASA) 42 p HC A03/MF A01 CSCL 09B

N86-19982

Unclas G3/61 05493

OCTOBER 1985

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES SEL-85-002

ADA TRAINING EVALUATION AND RECOMMENDATIONS FROM THE GAMMA RAY OBSERVATORY ADA DEVELOPMENT TEAM

OCTOBER 1985

NASA

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 organization members:

NASA/GSFC (Systems Development and Analysis Branch)
The University of Maryland (Computer Sciences Department)
Computer Sciences Corporation (Flight Systems 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.

Contributors to this document include

Bob Murphy	(Goddard Space Flight Center)
Mike Stark	(Goddard Space Flight Center)

Single copies of this document can be obtained by writing to

Frank E. McGarry
Code 552
NASA/GSFC
Greenbelt, Maryland 20771

ABSTRACT

The Ada training experiences of the Gamma Ray Observatory Ada development team are related, and recommendations are made concerning future Ada training for software developers. Training methods are evaluated; deficiencies in the training program are noted; and a recommended approach, including course outline, time allocation, and reference materials, is offered.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This report documents the experiences of the Gamma Ray Observatory (GRO) Ada development team regarding Ada training for software developers. The conclusions can be summarized as follows:

- Alsys, Inc., has prepared excellent training material both on videotape and for computer-aided instruction.
- Sufficient initial training can be given in 2 man-months. Expert support would still be needed as projects get underway. Less than 2 man-months may be inadequate.
- Intensive courses are less useful than spreading the same effort over a longer period of time. It is recommended that 24 hours be devoted solely to lecture, allocated over a 2-month period. This will allow sufficient time for studying and practicing with a compiler.
- Lectures should be arranged into five main parts:
  - Introduction--Provides, in a single lecture, background on why and how Ada was developed
  - Language Basics--Teaches enough of the Ada language features to enable the trainee to write nontrivial programs; provides the necessary background for discussing the software development process
  - Software Structure and Compilation--Examines the use of program libraries and separate compilation units; shows how the DEC Ada Compilation System can be used for configuration control

  - <u>Software Engineering in Ada</u>--Introduces design methodologies and shows how Ada is used to support software engineering concepts
  - <u>Advanced Ada Features</u>--Teaches the remaining language features, with special attention to generics, exceptions, tasking, and input/output

- Ada training requires hands-on practice in conjunction with any lecture course.

- A practice problem is a useful reinforcement immediately following a course because it allows developers to integrate the concepts that they have been taught. This practice problem should be done as a team project with each team member's effort being 1 to 2 man-months.

SECTION 1 - INTRODUCTION

Ada[1] is a software tool with the potential to reduce software life-cycle costs substantially. To exploit this potential to the fullest, it is extremely important that programmers receive the best possible training in use of the language. This document presents the views and recommendations of the Gamma Ray Observatory (GRO) Ada development team on the various training methods available during their training program. It also presents a final recommendation on an optimal training program to ensure the proper use of the Ada programming language.

The GRO Ada development team consists of four National Aeronautics and Space Administration (NASA) employees and three employees of Computer Sciences Corporation (CSC). Two of the NASA members are in the Data Systems Technology Division (Code 520), and two are in the Flight Dynamics Division (Code 550). The University of Maryland is also involved with the project in an advisory capacity.

The training program consisted of four main training methods. None of the team members had previous experience with Ada, so first on the agenda was Grady Booch's book <u>Software Engineering With Ada</u> (Reference 1). Videotaped tutorials made by Alsys, Inc. (Reference 2) were then viewed over a 40-hour period in a classroom environment in which there was opportunity for discussion. The videotapes were followed by a 24-hour seminar on a design methodology called the Process Abstraction Method (Reference 3) presented by George Cherry of Language Automation Associates. The last step was hands-on coding using Dec's Ada compiler.

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

The coding was an implementation of an Electronic Message System (EMS) that was required to maintain a data base of valid users and a data base of groups of users, allow a valid user to log on and read mail, allow mail to be sent to a single user or to each user in a group, and allow an authorized user to modify the data bases. The program was about 5700 source lines of code, 1400 of which were executable, and required a total of 1336 hours of effort to complete.

In addition to this training, several team members made use of other training methods such as Computer Thought Corporation's tutorial series (Reference 4) for the IBM PC and a course on Ada syntax taught by George Cherry (Reference 5). This document is based primarily on a survey of the seven team members; however, several other NASA employees took part in the survey when it applied to training methods with which they were familiar.

SECTION 2 - TRAINING METHOD EVALUATION

This section reviews each of the training methods that were available to the GRO Ada development team. A description of each training method is presented, followed by a summary of its strengths and weaknesses, and concluding with a recommendation for its use.

2.1 GRADY BOOCH TEXTBOOK

Description

Software Engineering With Ada, by Grady Booch is a comprehensive, 500-page textbook covering Ada syntax and an introduction to object-oriented design. Grady Booch, an Ada expert, has published many articles on Ada and software design methodologies. The book's main topics are listed in Appendix A.

Strengths

This textbook received favorable reviews from all seven team members. The book covers most aspects of Ada syntax well and was considered by all but one person to be a good introduction to object-oriented design with Ada. All but one person also found it to be a good reference after training was complete. This one person found it too wordy to be used as a reference. Packaging, abstraction, and types were the topics most survey members found were covered well.

Weaknesses

Two people felt that a shortcoming was an emphasis on design examples at the expense of more concrete examples using individual features. Generics, input/output (I/O), and concurrent processing were topics mentioned as not being well covered.

Recommendation

The group was split on the question of the level of complexity of this book. Half thought it introductory, whereas half though it intermediate. The book could really be used either way. This book was the team's first serious exposure to Ada and was very useful in that capacity. The structure of the book makes it easy to follow, the syntax of the language is covered well, and the book also provides an introduction to object-oriented design. At the same time, its depth of coverage, especially of syntax issues, makes it very useful to someone already familiar with Ada who is looking for a review or a better understanding of the language. As mentioned earlier, the book could also be used successfully as a reference.

2.2 ALSYS VIDEOTAPES

Description

The 27 Alsys, Inc., videotapes (and disk) tutorials cover all facets of Ada syntax. The lessons are taught by Jean Ichbiah, Robert Firth, and John Barnes, who were on the original Ada design team. The tapes average about 39 minutes each, ranging from 25 to 52 minutes. Appendix B lists the topics covered. A three-volume transcript of the tapes is available from Alsys as either preparatory or review material. Alsys also produces a course of computer-aided instruction on Ada (Reference 6) that is available for the DEC VAX or as a series of disks for the IBM PC. This course can be used as a companion to the videotapes or as a separate training method.

Strengths

These tapes, which received excellent reviews from the eight persons surveyed, are a very good method for learning Ada syntax. Ichbiah, Firth, and Barnes possess a thorough knowledge of Ada, and this is evident in the tapes. Their examples are clear and appropriate to the topic at hand, and their witty remarks liven up what could otherwise be dry subjects. Almost all of those surveyed felt that all of the topics were well covered. Singled out as especially good lessons were the tapes on the various data types and on tasking.

Weaknesses

The main area of concern about the tapes was the coverage of the organization of large projects into packages and tasks. Several people felt that more time could have been spent on library units and the overall structure of how a large Ada project is coded. The tapes use small examples to illustrate individual features, which is good for the sake of simplicity; however, one large example should have been developed as well.

A second area of concern was the lack of interaction inherent in the medium in which the lessons are presented. Although the speakers try to anticipate any questions, especially in the last two tapes, other questions are bound to arise. Some of these questions can be answered by rewinding the tapes and reviewing the area in question; however, answering the remaining questions requires the presence of a person who is experienced in Ada.

A lack of interaction can also cause a viewer to lose interest in what he or she is watching. The Alsys course for the PC or the VAX is a possible solution to this. Only a demonstration disk of this series of 27 lessons was available to the development team, but it did appear to be of the same high quality as the videotapes, with the added feature of interactive exercises. At various locations in this series, the user is prompted to type in answers varying in length from one line of code to a whole section of code.

Incorrect answers or parts of answers are then highlighted in red, and the user is prompted to correct these errors. If, after several tries, the user is still unable to answer correctly, the computer will furnish the correct answer and an explanation. This course would seem to be very useful as a companion to the videotapes. The one drawback of the series is the price: $6000 for the PC version and $9800 for the VAX 11/780 version. The VAX version also requires the use of a VT240 or VT241.

Recommendations

The Alsys videotapes should definitely be used in an Ada training program as a method of learning Ada syntax. The GRO Ada development team viewed them in a classroom setting over a 1-week period, with a person experienced in Ada present to answer questions. Although the opportunity for discussing problems with the rest of the class and with an Ada expert was very helpful, the time period allotted to view the tapes was much too short, and a lack of hands-on exercises was detrimental. It would be better if a day of tape viewing was alternated with a day of hands-on exercises emphasizing the tapes seen the previous day. It may also be necessary to extend the amount of time for viewing the tapes. The team averaged about five or six tapes each day, which was too much to be absorbed in that period.

The Alsys course for the PC or the VAX seems very good, and the interactive aspect is beneficial. The course may be useful as a source of exercises on the days between videotape viewings.

2.3 PROCESS ABSTRACTION METHOD SEMINAR

Description

The Language Automation Associates Advanced Ada Course is a 3-day seminar in a classroom environment. The course is taught by George Cherry and covers his Process Abstraction Method (PAM) of design and three examples using this method. It assumes a good knowledge of Ada syntax.

Strengths

PAM is an excellent method of design for programs using a large number of concurrent processes. It presents a simple yet comprehensive way of breaking down large problems into a manageable design that uses tasking correctly and efficiently.

Weaknesses

The team members had several complaints about the way in which PAM was presented. One complaint was that the course was very narrow in scope, covering only one design methodology; however, that is all the course is intended to cover. On the other hand, considering its scope, the course is spread out over too long a period of time; it could easily have been taught in 1 day. If more time were needed, it should have been spent doing an interactive example using PAM, developed by the class under the instructor's supervision. Instead, the instructor spent the last 2 days reviewing three examples that had already been developed. This was very tedious, and not very beneficial.

Several people also felt that the environment of the classroom was not conducive to interaction between students and the teacher. This severely reduced the benefits of learning in a classroom setting.

Recommendation

Despite its shortcomings, this course should be attended by Ada trainees. It is very helpful in learning the complex and crucial issues of tasking in Ada. However, the length of the course, which should include an interactive example, should be a maximum of 2 days. (The format of the course has recently been changed to 2 days of lecture on PAM after 3 days of lecturing on Ada syntax.)

2.4 COMPUTER THOUGHT CORPORATION TUTORIAL

Description

Computer Thought Corporation's disk series of Graphics Enhanced Ada Tutorials (GREAT) consists of computer-aided instruction for the IBM PC, using color graphics. The five-disk, menu-driven course provides the background of Ada and a general outline of Ada syntax. It would take approximately 40 to 50 hours of use to complete these tutorials. A list of the topics covered in the series is presented in Appendix C.

Strengths

Most topics on these disks were capably covered, and a few lessons were presented noticeably well. A broad outline of the history of Ada and its key features, the majority of the first disk, was put forth in a very clear and interesting manner. The same could also be said for the coverage of the flow of control in such things as "if" statements, "case" statements, and loops. The dynamic nature of the graphics provides an excellent method for presenting this subject. However, the "if" statement and the loop are language constructs that are not particular to Ada, and most people who have had any training in computer science would already have a good understanding of these concepts. The benefits from these lessons would therefore be limited to those with minimal training in computer languages.

Weaknesses

The development team had several complaints about this training method. The biggest was that it was not interactive. It would have been much more beneficial if the user had to type in answers occasionally instead of merely stepping through examples. An interactive approach to this training method would keep the user more involved and, therefore, more attentive to the subject being taught. The tutorial basically involves reading a book via the PC, and actually just reading the book without the computer would be preferable. Not only is a book less expensive, but it is also easier to access and requires less effort to refer back to earlier lessons.

The use of only one large, complex example throughout much of the tutorial was another complaint. Different sections of a message switch problem are used to exemplify most of the Ada concepts. Although this problem uses all of the unique aspects of Ada (such as generics and tasking) and provides an example of developing a large-scale program, it would have been better to use many different, smaller examples, for two reasons: First, a variety of examples is more interesting. Second, given the nature of the course's medium, it is quite possible that a user will view one disk and not see the following disk for several days. The user becomes annoyed at having to go back and review the explanation of the only available example every time he/she starts. Using smaller, self-contained examples, as well as the message switch example, would remedy this problem.

The last major complaint about this training method is that, because of the medium, some topics are so fragmented that they become confusing. The best example of this is the coverage of objects. Because only a limited number of words can fit on the screen at one time, ideas that should be in one place end up on several different screens. Although it is possible to move back and forth between screens, this can become confusing and cumbersome.

Recommendation

The development team does not recommend that the Computer Thought Corporation Ada tutorial be used as a main method of training Ada programmers. It could, however, be useful as a general overview for a person with no experience in Ada.

2.5 ADA SYNTAX SEMINAR

Description

The Language Automation Associates Introduction to Ada Course is a 4-day seminar in a classroom environment. The course is taught by George Cherry and covers all aspects of Ada syntax.

Strengths

The classroom setting of this method is its strongest feature. The ability to ask questions of an experienced Ada programmer while learning Ada is essential. Dr. Cherry competently teaches a good introduction to Ada syntax and is very enthusiastic about the subject. He uses many good examples illustrating almost all of the language's attributes.

Weaknesses

Although the examples are probably one of the course's biggest plusses, they are also one of its biggest drawbacks. Dr. Cherry spends too much time on examples. If he were to cut out some of the unnecessary examples, the course would be reduced to 3 days at most. (It should be noted that, at GSFC, this course has in fact been reduced to

3 days. It is now being taught as part of a 5-day course that includes an introduction to Dr. Cherry's Process Abstraction Method.)

A second problem with the course was its coverage of generics. The lesson was found to be too shallow for such a new and important topic.

Recommendation

Although this course is good, the team feels that a training program based on the Alsys videotapes would be better, providing they were taught in a classroom environment and augmented by an Ada expert to answer questions. The videotapes are just as good in content as the seminar and have two other good features: first, they are cheaper; once they are bought, there are no further expenses, whereas Dr. Cherry's course must be paid for every time it is taught. Second, providing an Ada expert is available on site to supervise the class, it is much easier to schedule a class featuring the videotapes, which are available at any time.

SECTION 3 - CONCLUSIONS FROM GRO PROJECT TRAINING

The GRO Ada development team found the most helpful aspects of the training in understanding Ada were (1) the discussions in the training classes and in team meetings and (2) the EMS practice problem. The chief drawback to the EMS was the size of the practice problem. What is needed is a smaller problem, but one that is not so trivial that concepts such as packages and data abstraction are no longer useful. The team designed the EMS with a user interface, a name interface to control the user and group data bases, and a mailbox interface to handle message traffic. The value of the EMS as a training exercise would have been improved if the group concept was eliminated. It would have simplified both the name interface and the user interface, while preserving the need to use packages and data abstraction.

The team found the following aspects of Ada to be either difficult or not sufficiently covered in the training:

- Input/Output (I/O)--Every training resource reviewed in Section 2 is sketchy on the working of I/O. This is mainly because Ada was designed to be as machine independent as possible, and efficient I/O is machine dependent. I/O needs a better explanation than provided by any of the classes or textbooks. Even the Ada Language Reference Manual (Reference 7) does not cover everything that can happen when different I/O packages are used together in the same compilation unit. The best approach to learning I/O is to practice as many operations in as many different combinations as possible.

- Tasking--Tasking is covered in detail in all of the training resources, but there are details that are left open by the language standard that have to be tested by actually running tasking applications. The select statement is an example. If there are multiple conditions, some compilers will always handle the first one reached in the code, whereas others will attempt to handle the selection process fairly. The point is that these things need to be tried on a computer. The tasking operations are, however, generally presented more clearly than the I/O, especially by George Cherry.

- Generics--Generics are pretty straightforward in theory but caused many problems in implementing the EMS. These problems may disappear when the DEC production compiler is used rather then the test version. Generic packages have been instantiated to handle I/O, but generic packages using all the different forms of generic parameters have not actually been written.

- Data Types--Data typing presented no real problems during the implementation of the EMS, wit the exception of using the type "Name," which conflicted with the I/O procedure "NAME." This, however, is an area that is not familiar to people without a computer science background. Typing supports software engineering concepts such as information hiding (private types) and provides a protection mechanism against unwanted operations. Types such as access types, variant and discriminated records, and the unconstrained array allow flexibility in implementation. In short, this area needs special attention, not because it is overly difficult, but because there are a lot of important details and benefits.

- Library Units and Library Structures--The concepts of separate compilation and program libraries were covered in the videotapes. The tapes did not, however, present the actual implementation of a library structure and how it would be used to support a large project. The DEC library structure is very sound; is straightforward and well designed to support the configuration of large projects. Team members therefore plan to prepare lectures and notes on the DEC ACS system to supplement the videotapes.

SECTION 4 - RECOMMENDED APPROACH TO ADA TRAINING

It is the opinion of the GRO Ada development team that the Alsys videotapes were excellent, with the major drawbacks being the short timespan spent viewing them and the lack of practice using a compiler. It is therefore recommended that a course be taught in a classroom environment and that it consist of viewing the videotapes, discussing the tapes and program problems, and listening to lectures covering methodologies and DEC's Ada environment. The course should finish with a larger problem that would allow the student to use Ada's features on an application program rather than on contrived examples. Group teaching is preferable to self study.

4.1 COURSE OUTLINE

The development team feels that software engineering issues should be discussed as soon as possible in the context of Ada training. Only about half of the material should be taught before software engineering issues are addressed. The recommended training course can be summarized as follows:

Part 1 - Introduction

- a. Purpose and history of Ada
- b Lexical elements

Part 2 - Language Basics

- a. Overview
- b. Declarations and typing
    1. Scalar types
    2. Declarations
    3. Array and record types
    4. Derived types, subtypes, and operations

c. Coding elements

   1. Names
   2. Expressions
   3. Statements

d. Basics of program units

   1. Subprograms
   2. Packages
   3. Tasks
   4. The package TEXT_IO

<u>Part 3 - Software Structure and Compilation</u>

a. Overloading, scope, and visibility

b. Use of Ada libraries

   1. Bottom-up construction from program library
   2. Top-down construction using separate compilation units

c. Use of DEC program libraries

<u>Part 4 - Software Engineering in Ada</u>

a. Introduction to design methodologies

   1. Compilable design
   2. Design methodologies

      (a) Traditional methods
      (b) Process abstraction
      (c) Object-oriented design

b. Data abstraction and information hiding

   1. Private and limited types
   2. Sample problems using software engineering concepts and Ada

Part 5 - Advanced Ada Features

a. Exceptions

b. Generic program units

c. Types and expressions

   1. Review of types
   2. Real-type modeling
   3. Access types and allocators
   4. Expressions

d. Tasking

   1. Rendezvous mechanism
   2. Implementation of task bodies
   3. Task types
   4. Example using PAM

e. Input/Output

   1. DIRECT_IO
   2. SEQUENTIAL_IO

f. Representation- and implementation-dependent features including LOW_LEVEL_IO

4.2 TIME ALLOCATION

The amount of time needed to cover this material in lecture is approximately 24 hours. This estimate is based on the 18-hour length of the Alsys videotapes plus 6 hours for the material on methodologies and the DEC compiler.

The development team recommends that the material be spread over a 2-month period. Two 1.5-hour lectures per week will satisfy this recommendation and will allow time for trainees to study and to practice using the compiler between lectures.

The five parts of the training course can then be divided into lectures as follows:

Part 1 - Introduction	1 lecture
Part 2 - Language Basics	4 lectures
Part 3 - Software Structure and Compilation	3 lectures
Part 4 - Software Engineering in Ada	4 lectures
Part 5 - Advanced Ada Features	4 lectures

The last stage of training should be a practice project such as the EMS. The project should be developed by a small group of three or four people and should take from 4 to 8 weeks to complete. This allows the trainees to integrate concepts that have been presented as separate topics.

4.3 TRAINING REFERENCES

The development team recommends that the following resources be used in Ada training:

- Software Engineering in Ada by Grady Booch (Reference 1)
- Alsys Inc., videotapes (Reference 2)
- Ada Language Reference Manual (LRM) (Reference 7)

These references should be used as follows throughout the training course:

Part 1 - Introduction

Booch Ch. 1-4
Tape 1
LRM Ch. 1-2.9

Part 2 - Language Basics

Booch Ch. 6, 15
Tapes 2-8, 10-12
LRM Ch. 3.1-3.7, 4.1-4.7, 5, 6.1-6.5, 7.1-7.3, 14.3

Part 3 - Software Structure and Compilation

Tape 14

LRM Ch. 6.6-6.8, 10

Lecture notes

DEC documentation

Part 4 - Software Engineering in Ada

Booch Ch. 5, 20

Tape 13

LRM Ch. 7.4-7.6

Lecture notes

Part 5 - Advanced Ada Features

Booch Ch. 11, 14, 16, 17, 19

Tapes 9, 10, 15-24

LRM 3.2-3.4, 3.8, 4.5.7-4.10, 9, 11-14

Booch's textbook is a good reference as well as a good teaching text. The chapters cited in the above outline are especially useful, but the entire book is helpful as a reference.

Lecture notes need to be prepared on two major subject areas. The first is the use of the DEC Ada Environment (ACS); the second is methodologies. Members of the GRO Ada development team and members of other Ada project teams at GSFC have enough experience with the ACS to prepare lecture material. In-house expertise is also available for preparing lectures on methodologies, but to give a wide enough view here would be beyond the scope of this document. George Cherry's process abstraction method is copyrighted, so it is recommended that a one-half to full day course be arranged with Dr. Cherry to give an overview of PAM. University of Maryland personnel could prepare lessons on software engineering in general, with members of the GRO Ada development team presenting what they have actually used on the project. Outside sources should be used only when they are necessary to fill in the gaps.

The end of initial training is not the end of learning about Ada. When Ada is being introduced into an environment, it is recommended that expert support be available to those who have completed the initial training and are working on Ada projects. This support can be provided by members of the GRO Ada development team or by Adasoft, Inc., experts under contract to Code 520.

APPENDIX A - CONTENTS OF SOFTWARE ENGINEERING WITH ADA BY GRADY BOOCH (REFERENCE 1)

Chapter 8 - Programming Into Ada

- Ada Programming Support Environment
- Software Life Cycle
- Trends and Conclusions

Chapter 9 - Appendices

- Syntax Charts
- Style Guide
- Predefined Language Environment
- Attributes and Pragmas

APPENDIX B - ALSYS INC., VIDEOTAPE TOPICS

Tape 1 - Introduction

Tape 2 - A Simple Program

Tape 3 - Types A: Introduction to Types

Tape 4 - Types B: Types and Subtypes

Tape 5 - Composite Types A: Arrays and Records

Tape 6 - Composite Types B: Discriminants

Tape 7 - Classical Programming A: Names, Expressions, and Statements

Tape 8 - Classical Programming B: Subprograms

Tape 9 - Access and Derived Types

Tape 10 - Numeric Types

Tape 11 - Program structure A: Visibility Rules

Tape 12 - Program Structure B: Packages

Tape 13 - Program Structure C: Private Types, Clauses

Tape 14 - Program Structure D: Separate Compilation

Tape 15 - Tasking A: Parallelism and Rendezvous

Tape 16 - Tasking B: The Select Statement

Tape 17 - Tasking C: Scheduling

Tape 18 - System-Dependent Programming A: Control of Representation

Tape 19 - System-Dependent Programming B: Access to Low-Level Features

Tape 20 - Exceptions A: Exception Handling

Tape 21 - Exceptions B: Programming With Exceptions

Tape 22 - Generic Units A: Introduction

Tape 23 - Generic Units B: Programming With Generic Units

Tape 24 - Input/Output

Tape 25 - Conclusions

Tape 26 - Technical Questions

Tape 27 - General Questions

APPENDIX C - COMPUTER THOUGHT CORPORATION DISK SERIES TOPICS

Disk 1 - Overview of Ada

- Example Overview
- Ada History
- Key Features

Disk 2 - Building and Using Ada Packages

- Package Structure
- Objects
- Types
- Subprogram Declarations

Disk 3 - Classical Programming in Ada

- Statements
- Control
- Exceptions
- Generics
- I/O Processing

Disk 4 - Concurrent Processing Using Ada Tasks

- Concurrency
- Tasking
- Termination
- Interrupts
- Pitfalls

Disk 5 - Summary of Advanced Ada Features

- Representation
- Low-Level I/O
- Other Languages
- Summary

REFERENCES

1. G. Booch, Software Engineering With Ada, Menlo Park, California: Benjamin/Cummings Publishing Co., Inc., 1983

2. J. Ichbiah, J. Barnes, and R. Firth, "Ichbiah, Barnes, and Firth on Ada," Alsys, Inc., Waltham, Massachusetts, 1983

3. G. W. Cherry, "Advanced Software Engineering With Ada--Process Abstraction Method for Embedded Large Applications," Language Automation Associates, Reston, Virginia, 1985

4. Computer Thought Corporation, "Graphics Enhanced Ada Training," Plano, Texas, 1985

5. G. W. Cherry, "Introduction to Ada for Software Engineers," Language Automation Associates, Reston, Virginia 1984

6. Alsys, Inc., "Lessons on Ada," Waltham, Massachusetts, 1985

7. MIL-STD 1815A-1983, Ada Language Reference Manual (LRM)

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-77-001, The Software Engineering Laboratory, V. R. Basili, M. V. Zelkowitz, F. E. McGarry, et al., May 1977

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

SEL-77-003, Structured FORTRAN Preprocessor (SFORT), B. Chu and D. S. Wilson, September 1977

SEL-77-004, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-001, FORTRAN Static Source Code Analyzer (SAP) Design and Module Descriptions, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

SEL-78-003, Evaluation of Draper NAVPAK Software Design, K. Tasaki and F. E. McGarry, June 1978

SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson and B. Chu, September 1978

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-78-202, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 2), W. J. Decker and W. A. Taylor, April 1985

SEL-79-001, SIMPL-D Data Base Reference Manual, M. V. Zelkowitz, July 1979

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-80-001, Functional Requirements/Specifications for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, A. L. Green, and C. E. Goorevich, February 1980

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-80-104, Configuration Analysis Tool (CAT) System Description and User's Guide (Revision 1), W. Decker and W. Taylor, December 1982

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, September 1982

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-82-306, Annotated Bibliography of Software Engineering Laboratory Literature, D. N. Card, Q. L. Jordan, and F. E. McGarry, November 1985

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-83-104, Software Engineering Laboratory (SEL) Data Base Retrieval System (DARES) User's Guide, T. A. Babst, W. J. Decker, P. Lo, and W. Miller, August 1984

SEL-83-105, Software Engineering Laboratory (SEL) Data Base Retrieval System (DARES) System Description, P. Lo, W. J. Decker, and W. Miller, August 1984

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-RELATED LITERATURE

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

^{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

^{3}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: Computer Societies Press, 1981

Basili, V. R., "SEL Relationships for Programming Measurement and Estimation," University of Maryland, Technical Memorandum, October 1979

[3]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: Computer Societies Press, 1980 (also designated SEL-80-008)

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

[3]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

[3]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

[1]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

[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

[3]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

[1]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

[1]Basili, V. R., and J. Ramsey, "Analyzing the Test Process Using Structural Coverage," Proceedings of the Eighth International Conference on Software Engineering, August 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, October 1979

[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

[1]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, August 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

[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

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

[3]Basili, V. R., 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

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

[3]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: Computer Societies Press, 1978

[1]Card, D. N., "A Software Technology Evaluation Program," Annais do XVIII Congresso Nacional de Informatica, Sao Paulo, Brasil, October 1985

[1]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," Proceedings of the Eighth International Conference on Software Engineering, August 1985

[3]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: Computer Societies Press, 1981

[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: Computer Societies Press, 1983

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

[1]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

[1]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

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

[1]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

[3]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: Computer Societies 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

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

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

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

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