TR-1403 May 1984

A Quantitative Analysis of a Software Development in Ada+*

Victor R. Basili,
Nora Monina Panlilio-Yap,
Connie Loggia Ramsey,
Chang Shih,
and Elizabeth E. Katz

Department of Computer Science
University of Maryland
College Park MD 20742

+ Ada is a trademark of the Department of Defense.

* Research for this study was supported in part by the Office of Naval Research and the Ada Joint Program Office under grant N00014-82-K-0225 to the University of Maryland.

ABSTRACT

A considerable amount of money and resources has been spent on the development of the new programming language Ada. The University of Maryland and General Electric have studied the development of a software project written in Ada. This paper presents the analysis of the effort, change, and error data. The total effort spent on training and methodology was 20% of the total effort spent on the project; this was more than the effort spent on any other phase. The greatest error rates appeared to be associated with the most Ada-specific features: tasking, generics and compilation units. Experience with high-level languages seemed to be associated with a better ability to grasp Ada concepts. Finally, the results strongly indicate the need for support tools for an Ada programming environment.

Acknowledgement

The authors would like to thank John W. Bailey, James T. Ramsey, and Dr. Marvin V. Zelkowitz for their invaluable assistance in this project.

1. INTRODUCTION

The Department of Defense has spent a considerable amount of money and resources on the development of the new programming language Ada. To develop a better understanding of the nature of this language, it is necessary to pull it out of the research arena and use it in an industrial environment where one must deal with issues such as training, budgets and support facilities.

To gain insight into the use of this new language, the University of Maryland and General Electric have jointly undertaken a study of the development of a software project written in Ada. A subset of a previously written satellite ground control system was to be reimplemented by four GE programmers over a period of approximately one year using Ada. The process involved two separate groups: (1) a team consisting of four programmers whose task was to build the software and to view this as a "normal" industrial task to as great an extent as possible, and (2) a team consisting of individuals from GE and the University of Maryland who would observe the programmers with a minimal amount of interference.

A set of goals and questions was established at the beginning of the project. These included generic goals for any software development project, goals relating to Ada as a design and implementation language, and goals relating to metrics for the Ada Programming Support Environment (APSE). Data collected from the project were analyzed. This paper describes the observations which provide answers to a subset of the goals and questions. While some of the answers are relevant only to our particular environment, others apply to any group wishing to use Ada for the first time, and still others apply to any Ada environment. More specifically, this paper addresses those goals that are related to effort, changes, errors and programmer characteristics.

2. BACKGROUND

The project under study involved the redesign and implementation in Ada of a portion of a satellite ground control system originally written in FORTRAN. This subset included an interactive operator interface, graphic output routines, and concurrent telemetry monitoring.

Four programmers with diverse backgrounds were selected for this project in order to determine whether a programmer's experience and education will influence his understanding and use of Ada. Table 2.1 shows the education and experience of each programmer and the programming languages each is familiar with. The lead programmer was fluent in FORTRAN and assembler languages. He had many years of industrial experience working in the application area. The senior programmer had a master's degree in computer science and had worked with many languages, such as COBOL, PL/1, Lisp, ALGOL, and SNOBOL in addition to FORTRAN and

assembler. He had a reasonable amount of experience with the application area, though not nearly as extensive as the lead programmer. The junior programmer had just obtained a bachelor's degree in computer science and had programmed in Pascal as well as all the languages that the senior programmer knew. He had no industrial experience whatsoever and was not at all familiar with the application area. The librarian's computer science background consisted of a single course in FORTRAN programming.

None of the programmers knew Ada before the project began. They received one month of intensive training in Ada. They viewed fifteen hours of videotaped lectures given by Ichbiah, Firth, and Barnes (the major developers of the language) over a period of four days. This was followed by six days of further training given by George W. Cherry of Language Automation Associates which was spread over a period of four weeks. During this time, the programmers also practiced writing Ada programs, read the Ada reference manual and reviewed their class notes. The NYU Ada/Ed interpreter was used for programming assignments which included a 500-line team project. Finally, the programmers were given a half-day class on software engineering techniques by Victor R. Basili of the University of Maryland. Among the topics discussed were chief-programmer teams, design and code walkthroughs, and structured programming.

The programmers never saw the comparable Fortran source programs. It was estimated that the equivalent Fortran subset was about 10,000 lines of Fortran code - a size amenable for the programmers to build in a year.

Table 2.1 - Backgrounds of Programmers

Programmer	Years of Professional Experience	Education	Languages
Lead	9	B.S. (Comp. Sci.)	FORTRAN, Assembler
Senior	7	M.S. (Comp. Sci.)	FORTRAN, Assembler, COBOL, PL/1, Lisp, ALGOL, SNOBOL
Junior	0	B.S. (Comp. Sci.)	FORTRAN, Assembler, COBOL, PL/1, Lisp, ALGOL, SNOBOL, Pascal
Librarian	0	High School Degree	FORTRAN

The project began in February 1982 and ended in July 1983. Requirements analysis of this project was done concurrently with training during the first month. Following this, system design using an Ada-like Programming Design Language (PDL). coding, and testing were done.

3. GOALS

This study attempts to answer goals and questions from four different areas of the Ada project. We characterized the effort, the changes, and the errors. We also tried to associate each programmer's background with his use of Ada and his performance on the project.

In order to collect the data necessary for the study, the programmers were asked to complete various forms. The forms relevant to this study are:

a) component status report forms
b) change request forms
c) error description forms
d) individual document change report forms

A component status report form contains information on the weekly breakdown of effort spent by a programmer on each phase of the project. Each time a need for change was determined, a change request form was filled out. If the change was an error correction, an error description form was also completed. An individual document change report form was filled out for every component involved in a change. Samples of these forms may be found in Appendix 7.

In addition to the data collected on the forms, a copy of the last design and code versions for each module was kept. The source code measures used in this paper were taken from the latest version for all modules.

3.1 Effort

In order to analyze the effort in the project, we determined how the effort was distributed over the phases (i.e. requirements, design, coding, testing, training, etc.) of the project. In addition, we determined how the effort for the project was distributed over time.

3.2 Changes

Our second goal was to analyze the changes in the project. We calculated the number of each type of change, where the types of changes are:

a) error corrections
b) changes in problem domain
c) planned enhancements
d) avoidances of apparent problems with the compiler
e) avoidances of other problems in the developing environment
f) adaptations to a change in the developing environment
g) improvements of clarity, maintainability or documentation
h) optimization of time, space or accuracy
i) insertion or deletion of debug code
j) other than above

Furthermore, the number of changes according to document type was tabulated based on the highest level document modified in any component for each change. The document types are requirements, PDL, and code modules.

Another aspect of the analysis involved determining how the changes were distributed over the software development cycle. We calculated the number of components involved in each change and the number of interface changes made. Finally, we determined how long it took to establish the need for change and how long it took to design and implement the change. From these results, we hoped to answer other questions such as how effective Ada is in producing software that is easily changed and whether the need for change was easily determined.

3.3 Errors

Our third major goal was to characterize the errors that resulted. We wanted to determine which of the following types of errors occurred:

a) requirements incorrect
b) requirements misinterpreted
c) design incorrect
d) design misinterpreted
e) code incorrect
f) external environment misunderstood (not language or compiler)
g) clerical error

There were specific questions that we tried to answer regarding the errors. What activities were used to detect and isolate the errors? Were they easy to find and to correct? How much effort was required to correct them? Where was the information needed to correct errors found? At which stage in the project did they enter the system? For each type of error, we determined the highest level document that needed to be changed. We also obtained the number of interface errors and the distribution of errors over time.

The errors were classified as follows:

a) Language
b) Problem
c) Clerical

Language errors were those which involved the syntax or semantics of a feature or those which involved the concept behind a feature. The problem category encompassed logic errors and errors related to the environment. Clerical errors included those due to carelessness, e.g. typographical errors.

We determined which features of Ada are commonly involved in errors. We also summarized the programmers' responses to questions regarding their understanding of the features, such as: Does the documentation explain the features clearly? Can the errors be attributed to lack of understanding of Ada? lack of experience with a feature? confusion with another language?

We tried to determine if the use of Ada PDL causes a preoccupation with syntax during the design stage. Furthermore, we attempted to find out if there were errors uncovered during the design stage that ordinarily would have been found during coding because of the use of Ada PDL.

3.4 Programmers

Our fourth major goal was to determine whether there was any relationship between the background of the programmers involved and their use of Ada. We obtained breakdowns of effort, changes, and errors by programmer. Are certain types of errors associated with particular programmers? Are some features of the language used incorrectly or inappropriately by the programmers? Finally, we tried to determine if programmers with no or little previous high-level language experience have more or fewer problems with Ada than programmers with substantial high-level language experience.

4. OBSERVATIONS

4.1 FACTORS

Several factors affected the outcome of this study, and it is important to understand them so the results of the analysis can be interpreted properly. First of all, no full production quality processor was available at the time the experiment was being conducted, and this had a major impact on the project. No compiler was available initially, and several rumored products were "promised" imminently. Finally, it became apparent that no such compiler would be made

available, so the useful, but very slow, NYU Ada/Ed interpreter was used. However, even that became unusable towards the end of the project as the size of the developing system grew. This had a demoralizing effect on the programmers who did not finish coding or testing the project. When the ROLM compiler became available, some further testing was done. The results in this paper are based on data collected through coding and some unit testing. The effort data used in this study show that little time was spent on testing. (See Table 4.1a.) In addition, the vast majority of the Ada-related errors were syntax errors. This might not have been the case if the code had actually been executed and testing had been completed. It is probable that many more logic errors would have been uncovered had syntax error-free compilation of all the modules been achieved.

Secondly, the lack of an automated PDL processor prevented the programmers from investigating deeper logic issues instead of simple syntax errors. Many errors which first appeared in the design stage would have been caught then had a PDL processor been used. However, they were not caught until compilation during the coding stage. In several of these cases, the code was changed but the programmers failed to correct the design document.

Thirdly, even though the requirements were taken from a previous FORTRAN project, only a subset of the project was to be recoded in Ada. Thus, the requirements document had to be cut down accordingly. The resulting set of requirements was improved and made into a consistent subsystem. This accounts for the effort spent on requirements and the substantial number of requirements changes. (See Tables 4.1a and 4.2a.)

Appendix 1 shows the effort spent on coding throughout the project. There was a 'false start' in the 12th week of the project. Since the programmers had a problem with methodology, in particular, writing abstract PDL, they began coding the project before they were prepared to do so. They were told to stop, and they did not resume coding until the 25th week of the project. This accounts for the large gap of no coding activity for nine weeks in the middle of the figure.

The project was temporarily discontinued after the 49th week, and further testing was resumed in the 62nd week by the junior programmer. This explains the periods of no activity in the graphs of effort, changes and errors over time contained in the appendices.

A final factor which influenced the results was that Ada is a completely new language with features not present in any other programming language. Thus, 20% of the total effort was spent on training and methodology, which is more than the effort spent on any other phase of the project. (See Table 4.1a.) This is a much higher percentage than would typically be spent on most projects. Furthermore, even this amount seems insufficient because the programmers indicated that they did not feel comfortable with Ada until after they left the project.

4.2 DISCUSSION

4.2.1 Effort

Table 4.1a shows the amount of effort spent on each phase of the project. As stated previously, training and methodology accounted for 20% of the effort, a very high percentage compared to most other projects. The distribution of effort over time follows normal software development patterns and is shown in Appendix 1. Productivity was measured and is presented in Table 4.1b. It was determined that 18.52 lines of code (including comments but excluding blank lines) and 9.83 lines of text (lines containing part of an Ada statement) were developed per hour spent in code development. Productivity for the effort spent on the entire project was also calculated, but the values are upper bounds (and may not be meaningful) since the project was not completed.

Table 4.1a - Effort for Each Phase of the Project

Project Phase	Amount Of Time (in hours)	Percentage
Requirements Analysis	530.5	12.73%
Requirements Writing	113.6	2.73%
Design Creation	514.4	12.34%
Design Reading	37.7	0.91%
Formal Design Review	162.4	3.89%
Coding	305.6	7.33%
Code Reading	13.3	0.32%
Formal Coding Review	62.3	1.50%
Unit Testing	332.7	7.98%
Integration Testing	0	0.00%
Review Testing	0	0.00%
Training and Methodology	849.1	20.38%
Other Activity	1245.7	29.89%
Total Requirements	644.1	15.46%
Total Design	714.5	17.14%
Total Code Development	381.2	9.15%
Total Testing	332.7	7.98%
Total Training and Methodology	849.1	20.38%
Total Other Activity	1245.7	29.89%
Entire Project	4167.3	100.00%

Table 4.1b - Productivity

Code Developed	Productivity (Lines of Code per Hour)	
	Total Code Development	Entire Project
Text	9.83	1.09
All Non-blank Lines	18.52	2.04

Note - Text refers to any line containing part of an Ada statement.

4.2.2 Changes

The analysis of the 337 change request forms and the 439 individual document change forms resulted in the following observations. The number of changes according to document type (the highest level of document modified in any component for this change) is displayed in Table 4.2a. Code changes accounted for 61% of the changes. As stated previously though, many of these changes were errors which should have been caught at the design stage. Thirty-two percent of the changes were in design documents, and only seven percent were in requirements documents. The number of overall changes and the number of non-error changes per thousand lines of code are presented in Tables 4.2b and 4.2c respectively. These numbers are lower bounds and may not be meaningful because the project was not completed.

The breakdown by type of change is shown in Table 4.3. The majority (57%) of the changes were error corrections which will be described in detail later. Of the non-error changes, 52% were improvements of clarity, maintainability and documentation. Most of these were PDL and requirements changes as shown in Appendix 2; this indicates that there was concern for good design.

The time to determine the need for change was one hour or less in almost all cases as shown in Table 4.4. In addition, 46% required only one tenth of an hour. This indicates that the need for these changes was easily determined. There were a few changes which took much longer than the average of one half hour per change. It took one or more days to determine the need for each of two planned enhancements. This was reasonable for the type of change. A different code change which took

Table 4.2a - Changes by Document Type

Document Type	Number of Changes	Percentage
Requirements	24	7.12%
PDL	107	31.75%
Code Module	206	61.13%

Table 4.2b - Number of Changes per Thousand Lines of Code

	Changes in Code Modules Only	All Changes
Text	44	71
Non-blank lines	23	38

Table 4.2c - Number of Non-Error Changes per Thousand Lines of Code

	Changes in Code Modules Only	All Changes
Text	12	31
Non-blank lines	7	16

Table 4.3 - Breakdown of Changes by Type

Type of Change	Number of Changes	Percentage
Error Corrections	192	56.96%
Changes in Problem Domain	1	0.29%
Planned Enhancements	9	2.67%
Avoidances of Apparent Problems with the Compiler	18	5.37%
Avoidances of Other Problems in the Developing Environment	2	0.59%
Adaptations to a Change in the Developing Environment	7	2.08%
Improvements of Clarity, Maintainability or Documentation	76	22.55%
Optimization of Time, Space or Accuracy	2	0.59%
Insertion or Deletion of Debug Code	9	2.67%
Other Than Above	21	6.23%

two days involved avoiding a problem with the compiler. Another change which involved a global definitions package took two days. It entailed the detailed modification and interfacing of several components.

Surprisingly, the amount of time needed to design and implement changes was very small. (See Table 4.5.) Again, the vast majority of

Table 4.4 - Time to Determine Need for Change

Effort	Number of changes	Percentage
0.1 hour	155	46.00%
0.2 hour	81	24.04%
0.3 hour	23	6.82%
0.4 hour	2	0.59%
0.5 hour	36	10.68%
0.6 hour	5	1.48%
0.8 hour	9	2.67%
0.9 hour	3	0.89%
1 hour	11	3.26%
2 hours	2	0.59%
3 hours	1	0.30%
4 hours	4	1.19%
6 hours	1	0.30%
1 day	1	0.30%
2 days	3	0.89%

Total Effort = 166.3 hours
Mean = 0.49 hours per change
Standard Deviation = 1.64 hours per change
Median = 0.20 hours per change

changes took one hour or less to handle. Of the code changes, all except five took two hours or less. Two changes, which took three hours and one day respectively, involved avoiding problems with the compiler. One change which took one and a half days resulted from an adaptation to a change in the development environment. One code change which took four hours was an error correction and will be discussed in the errors section. One code change took four days; this involved the same global definitions package as described above. The few other changes which took much longer than usual were mostly planned enhancements and improvements of clarity, maintainability and documentation of requirements documents. The change which took one week was a planned enhancement in a requirements section.

Table 4.5 - Time to Design and Implement Changes

Effort	Number of Changes	Percentage
0.1 hour	164	48.66%
0.2 hour	78	23.14%
0.3 hour	19	5.63%
0.4 hour	14	4.14%
0.5 hour	15	4.44%
0.6 hour	7	2.08%
0.7 hour	2	0.59%
0.8 hour	0	0.00%
0.9 hour	2	0.59%
1 hour	10	2.97%
1.1 hours	2	0.59%
1.2 hours	1	0.30%
1.3 hours	0	0.00%
1.4 hours	1	0.30%
1.5 hours	4	1.19%
2 hours	4	1.19%
3 hours	1	0.30%
3.5 hours	1	0.30%
4 hours	1	0.30%
5 hours	1	0.30%
5.2 hours	1	0.30%
6 hours	1	0.30%
0.8 day	1	0.30%
1 day	2	0.59%
1.5 days	1	0.30%
2 days	1	0.30%
3 days	1	0.30%
4 days	1	0.30%
1 week	1	0.30%

Total Effort = 260.1 hours
Mean = 0.77 hours per change
Standard Deviation = 3.34 hours per change
Median = 0.20 hours per change

The total effort spent on determining the need for and implementing changes was 426.4 hours, which is 10% of the total effort for the entire project. The average cost was 1.27 hours per change. However, it should be noted that most of the changes took much less time than this. The changes began in the nineteenth week of the project and the distribution of the changes throughout the software development cycle is shown in Appendix 3.

The number of components involved in each change is shown in Table 4.6. Only one component was modified in 77% of the changes, but up to five components were involved in some changes. We also determined the number of interface changes. (See Tables 4.7a and 4.7b.) In this paper, an interface change is defined as one which entails a change in more than one component at the same level of document. There was a total of 70 interface changes (21% of all changes). Only 2.9% of these were in the requirements, and the rest were equally divided between design and code. As many as five components were involved in these interface changes.

Table 4.6 - Number of Components Involved in Each Change by Type of Document

Number of Components Involved	Number of Changes by Type of Document			
	Req.	PDL	Code	All Levels of Document *
1	23	82	177	260
2	2	19	24	47
3	0	7	7	17
4	0	6	3	10
5	0	2	0	3

* The first three columns do not add up to the last column. This is because a change may involve several components at different levels of document. (e.g. one change may involve three components - two PDL components and one code component.)

Table 4.7a - Interface Changes

	Number of Components Involved					
	2	3	4	5	Total	%
Req.	2	0	0	0	2	2.86%
PDL	19	7	6	2	34	48.57%
Code	24	7	3	0	34	48.57%
Total	45	14	9	2	70	100.00%
%	64.28%	20.00%	12.86%	2.86%	100.00%	

Table 4.7b - Non-Error Interface Changes

	Number of Components Involved					
	2	3	4	5	Total	%
Req.	2	0	0	0	2	3.92%
PDL	14	3	6	2	25	49.02%
Code	14	7	3	0	24	47.06%
Total	30	10	9	2	51	100.00%
%	58.82%	19.61%	17.65%	3.92%	100.00%	100.00%

4.2.3 Errors

A total of 192 error description forms were examined. Each of these forms corresponds to a change request which falls under the error correction category. Table 4.8 shows a breakdown of the errors by type. From this, we can see that the vast majority (79%) of the errors were due to incorrect code. Most of the remaining errors were attributable to incorrect design.

The activities used in an attempt to detect errors were mostly compilation, design reading, design walkthroughs, code reading, or some combination of these. Approximately half of the errors were successfully detected through compiler messages, and a slightly smaller number were successfully detected through readings and walkthroughs. These same activities were used to isolate the source of the error. Code reading was more successful at isolating the source of the error than at detecting it, and the opposite is true of compiler messages. In the case of design reading and walkthroughs, detection of the error and isolation of its source usually took place simultaneously, but in many cases the programmer only checked the detection columns on the form. A complete listing of the activities used to detect and isolate errors is provided in Appendix 4.

Table 4.8 - Breakdown of Errors by Type

Type of Error	Number of Errors	Percentage
requirements incorrect	2	1.04%
requirements misinterpreted	4	2.08%
design incorrect	30	15.63%
design misinterpreted	0	0.00%
code incorrect	151	78.65%
external environment misunderstood (not language or compiler)	0	0.00%
clerical error	5	2.60%

Over 80% of the errors took at most twelve minutes (0.2 hour) to isolate. Approximately as many errors required as little time to correct. Only seven errors took an hour or more either to isolate or to correct. One error took an hour to isolate but only required 0.1 hour to correct. It was a design incorrect error which involved renaming a file. Another error classified as code incorrect took two hours to isolate but only 0.3 hour to correct. An undefined part of a string was passed as an argument to a function. Two errors involving incorrect design each required only 0.1 hour to isolate but over an hour to correct. One of these was a tasking error involving a synchronization problem between two components and it required 5.2 hours. Another, which required 1.5 hours, was a logic error involving input/output. The remaining three errors took an hour or more to isolate and an hour or more to correct. Two of them involved incorrect code. One required the insertion of error checks and exception handlers in a routine to conform to the specifications; this took one hour to isolate and one hour to correct. The other took four hours to isolate and four hours to correct; it was an input/output syntax error. The last error which took one hour to isolate and one hour to correct was a requirements incorrect error. A superfluous requirements section was found, and this was eventually deleted. Table 4.9 shows the time required to isolate errors; Table 4.10 gives a breakdown by error type of effort needed to correct errors.

Table 4.11 shows when errors entered the system. Seventy-two percent occurred during the Ada coding stage. Twenty-four percent also entered the system during design. As stated previously, several of the errors reported as coding errors actually originated in the design stage.

A summary of the different types of errors and the highest level document that had to be changed for each one is presented in Table 4.12a. Eight errors where the code was incorrect involved changes in the PDL. This may seem anomalous. However, these errors resulted in mere code changes to the PDL document and not changes to the design itself.

Table 4.9 - Time to Isolate Errors

Effort (hours)	Number of Errors	Percentage
0.1	115	59.90
0.2	44	22.92
0.3	10	5.21
0.5	9	4.69
0.6	2	1.04
0.8	5	2.60
0.9	2	1.04
1.0	3	1.56
2.0	1	0.52
4.0	1	0.52

Mean = 0.23 hours per error
Standard Deviation = 0.36 hours per error
Median = 0.10 hours per error

Table 4.10 - Time to Correct Errors

Effort (hours)	Number of Errors	Percentage
0.1	116	60.43
0.2	47	24.48
0.3	12	6.25
0.4	5	2.60
0.5	5	2.60
0.6	2	1.04
1.0	2	1.04
1.5	1	0.52
4.0	1	0.52
5.2	1	0.52

Mean = 0.22 hours per error
Standard Deviation = 0.48 hours per error
Median = 0.10 hours per error

Table 4.11 - Stage in Which Error Entered the System

Stage of the Project	Number of Errors	Percentage
requirements	2	1.04
design	46	23.96
Ada coding	139	72.40
testing	1	0.52
implementing another change	4	2.08
other or can't tell	0	0.00

(For example, a semicolon error in the PDL document would have been counted as incorrect code, not incorrect design.) Table 4.12b shows the number of errors per thousand lines of code.

The number of interface errors (those errors which entailed modifications in more than one component at the same level of document) was

Table 4.12a - Type of Error vs. Highest Level Document Changed

Type of Error	Number of Errors		
	Req.	PDL	code module
requirements incorrect	2	0	0
requirements misinterpreted	0	4	0
design incorrect	0	30	0
design misinterpreted	0	0	0
code incorrect	0	8	143
external environment misunderstood	0	0	0
clerical error	0	0	5
TOTAL	2	42	148
PERCENTAGE	1.04	21.88	77.08

Table 4.12b - Number of Errors per Thousand Lines of Code

	Errors in Code Modules Only	All Errors
Text	31	41
Non-blank lines	17	22

calculated. Only 10% of all the errors were interface errors. Approximately half were in the design, and half were in the code. The maximum number of components involved for any single error was three. (See Table 4.13.)

Appendix 5 contains the distribution of errors over the software development cycle. The errors displayed a normal development pattern.

As explained in Section 3.3, the errors were classified as follows:

a) Language
b) Problem
c) Clerical

Language errors were those which involved the syntax or semantics of a feature or those which involved the concept behind a feature. The problem category encompassed logic errors and errors related to the environment. Clerical errors included those due to carelessness, e.g. typographical errors.

Of the 192 error description forms examined, 146 (76%) claimed that the use of Ada contributed to the error. As shown in Table 4.14, the vast majority of the errors were language errors, and furthermore, 69% of these were merely syntax errors, which explains why so many of the errors took so little time to correct. There were 24 syntax errors per thousand lines of text (any line containing part of an Ada statement) and 13 syntax errors per thousand non-blank lines. The language/problem/clerical classification is further broken down by error type in Table 4.15. As might be expected, most of the errors involving requirements were problem errors, and most of the errors involving incorrect design or code were language-related errors.

Table 4.13 - Interface Errors

Document	Number of Errors	
	2 components involved	3 components involved
Requirements	0	0
PDL	5	4
Code	10	0

Table 4.14 - Number of Language, Problem and Clerical Errors

Language		166
Concept	7	
Syntax	114	
Semantics	45	
Problem		21
Clerical		5

Table 4.15 - Type of Error vs. Language, Problem or Clerical Classification

Type of Error	Number of Errors		
	Language	Problem	Clerical
requirements incorrect	0	2	0
requirements misinterpreted	1	3	0
design incorrect	25	5	0
design misinterpreted	0	0	0
code incorrect	140	11	0
external environment misunderstood	0	0	0
clerical error	0	0	5
TOTAL	166	21	5

Several Ada language features were involved in errors. Most common among these were low-level syntax (e.g. semicolon, parenthesis, assignment) and loops. There were also a considerable number of errors involving tasks, separate compilation, generics, procedures/functions, parameters, and declarations. As previously stated, most of the errors were syntax errors. There were only seven concept errors, and these involved tasking, exceptions, access types, file input/output and packages. Of the forty-five semantic errors, there were six each involving parameters and generics and five each involving compilation units and declarations. (See Table 4.16a.)

The errors involving the various Ada language features were normalized with respect to the number of times each feature was used. (See Table 4.16b.) For example, there were eight errors involving tasks and 21 occurrences of tasking in the project; this gives a 38% error rate for tasking which was the highest percentage for any feature. Access types and generics had error rates of 27% and 24% respectively. Compilation units and PRAGMA each had a 13% error rate. It is worthwhile to note that all of these features are specific to Ada.

The error description forms included questions to assess the programmers' comprehension of Ada features. For errors in the PDL or code, the programmer said that the documentation explained the features clearly in most cases and that he understood the features but did not apply them correctly. In some cases, the programmer did not understand

Table 4.16a -Errors Categorized by Ada Language Feature

Ada Language Feature	Number of Errors			
	Concept	Semantics	Syntax	TOTAL
semicolon	0	0	17	17
parenthesis	0	0	12	12
colon	0	0	3	3
:=	0	0	6	6
quotes	0	0	4	4
comment	0	0	4	4
identifier	0	1	4	5
loop	0	0	11	11
CASE	0	0	1	1
IF	0	0	6	6
BEGIN/END	0	0	4	4
RETURN	0	0	1	1
scoping	0	0	2	2
typing	0	1	5	6
aggregate	0	1	0	1
strings	0	1	0	1
arrays	0	3	2	5
records	0	2	2	4
access type	1	2	0	3
declarations	0	5	8	13
parameters	0	6	4	10
procedures/ functions	0	4	7	11
tasking	2	2	4	8
exceptions	2	0	1	3
generics	0	6	2	8
packages	1	0	1	2
compilation units	0	5	2	7
attributes	0	1	0	1
PRAGMA	0	2	0	2
file input/output	1	0	1	2
overloading	0	3	0	3

Table 4.16b - Errors Normalized with Respect to Feature Usage

Ada Language Feature	Percentage of Errors			
	Concept	Semantics	Syntax	TOTAL
semicolon	0	0	0.48	0.48
parenthesis	0	0	3.81	3.81
colon	0	0	0.28	0.28
:=	0	0	0.82	0.82
comment	0	0	0.08	0.08
identifier	0	0.27	1.07	1.34
loop	0	0	6.55	6.55
CASE	0	0	5.88	5.88
IF	0	0	2.73	2.73
BEGIN/END	0	0	2.08	2.08
RETURN	0	0	0.83	0.83
aggregate	0	0.06	0	0.06
strings	0	0.38	0	0.38
arrays	0	0.21	0.14	0.35
records	0	0.25	0.25	0.50
access type	9.09	18.18	0	27.27
declarations	0	0.37	0.59	0.96
parameters	0	0.96	0.63	1.59
procedures/ functions	0	0.70	1.23	1.93
tasking	9.52	9.52	19.05	38.09
exceptions	1.01	0	0.51	1.52
generics	0	18.18	6.06	24.24
packages	6.25	0	6.25	12.50
compilation units	0	10.00	4.00	14.00
attributes	0	0.70	0	0.70
PRAGMA	0	12.50	0	12.50

the features fully, and in a few others he understood the features separately but not their interactions. The programmer usually remembered how the features should be applied or obtained information from another programmer to correct the error. For more details, see Appendix 6.

4.2.4 Programmers

Table 4.17a shows a breakdown of the effort for each programmer. The productivity of each programmer is shown in Table 4.17b. The types of changes versus the programmer who authored the document in which the change was made are presented in Table 4.18a, while the types of changes requested by each programmer are presented in Table 4.18b. In all except 67 cases, the programmer who requested the change was the same person who had authored the document in which the change was made. In

Tables 4.19a and 4.19b, we see how many of each type of error the various programmers committed and found. Table 4.19c shows the number of errors per thousand lines of code for each programmer.

The lead programmer who had the most industrial experience and the most experience in the specific application area spent most of his time working on the requirements of the project, and a considerable amount of time on design. He made and found the vast majority of the design errors.

The senior programmer worked mostly on design. He made and found all four of the requirements misinterpreted errors. Note that the highest level document changed for these errors was the PDL. He did not do much coding, but was by far the most productive of the four. He averaged 39.85 lines of text per hour during code development (six times more productive than the other programmers) and 2.81 lines of text per hour for the entire project.

The junior programmer spent almost an equal amount of time on design and coding. He seemed to have the easiest time grasping ideas in Ada, but he had the highest error rate. He did most of the unit testing and therefore found most of the coding errors. He and the senior programmer requested 80% of the changes. They also made the most coding errors, but they had written the most code and their code was tested

Table 4.17a - Effort for Each Programmer

Project Phase	Amount of Time (in hours) spent by each programmer			
	Lead	Senior	Junior	Librarian
Requirements Analysis	284.5	91.0	110.0	0
Requirements Writing	86.3	8.9	17.9	0
Design Creation	139.1	114.1	222.7	38.5
Design Reading	16.0	9.7	3.5	4.5
Formal Design Review	62.7	42.2	50.1	7.4
Coding	88.0	34.5	164.6	18.5
Code Reading	0	6.0	7.3	0
Formal Coding Review	22.6	14.2	19.2	6.3
Unit Testing	0	70.5	241.5	20.7
Training and Methodology	226.5	216.5	270.2	135.9
Other Activity	321.0	167.9	292.6	458.7
Total Requirements	370.8	99.9	127.9	0
Total Design	217.8	166.0	276.3	50.4
Total Code Development	110.6	54.7	191.1	24.8
Total Testing	0	70.5	241.5	20.7
Total Training and Methodology	226.5	216.5	270.2	135.9
Total Other Activity	321.0	167.9	292.6	458.7
Entire Project	1246.7	775.5	1399.6	690.5

Table 4.17b - Productivity of Each Programmer

Code Developed		Productivity (Lines of Code per Hour) Amount of Code Developed	Entire Project
Text	Lead	6.56	0.58
	Senior	39.85	2.81
	Junior	6.08	0.83
	Librarian	6.33	0.23
All Non-blank Lines	Lead	11.50	1.02
	Senior	71.44	5.04
	Junior	18.47	2.52
	Librarian	7.78	0.28

Note - Text refers to any line containing part of an Ada statement.

Table 4.18a - Types of Changes vs. Document Author

Type of Change	Lead	Senior	Junior	Librarian
Total Changes	48	143	139	7
and % of all changes	14.24	42.43	41.25	2.08
Requirements	9	4	10	1
PDL	29	45	33	0
Code Module	10	94	96	6
Error Corrections	28	75	85	4
Changes in Problem Domain	1	0	0	0
Planned Enhancements	3	3	3	0
Avoidances of Apparent Problems with the Compiler	0	7	11	0
Avoidances of Other Problems in the Developing Env.	0	1	1	0
Adaptations to a Change in the Developing Env.	0	6	0	1
Improvements of Clarity, Maint. or Documentation	9	37	28	2
Optimization of Time, Space or Accuracy	0	1	1	0
Insertion or Deletion of Debug Code	0	7	2	0
Other Than Above	7	6	8	0

Table 4.18b - Types of Changes Requested by Each Programmer

Type of Change	Lead	Senior	Junior	Librarian
Total Changes	56	90	178	13
and % of all changes	16.62	26.70	52.81	3.86
Requirements	9	4	10	1
PDL	29	43	34	1
Code Module	18	43	134	11
Error Corrections	31	49	104	8
Changes in Problem Domain	1	0	0	0
Planned Enhancements	3	1	3	2
Avoidances of Apparent Problems with the Compiler	0	4	14	0
Avoidances of Other Problems in the Developing Env.	0	1	1	0
Adaptations to a Change in the Developing Env.	0	2	4	1
Improvements of Clarity, Maint. or Documentation	9	32	33	2
Optimization of Time, Space or Accuracy	0	1	1	0
Insertion or Deletion of Debug Code	0	0	9	0
Other Than Above	12	0	9	0

Table 4.19a - Types of Errors Made by Each Programmer

Type of Error	Number of Errors			
	Lead	Senior	Junior	Librarian
requirements incorrect	1	0	1	0
requirements misinterpreted	0	4	0	0
design incorrect	17	9	4	0
design misinterpreted	0	0	0	0
code incorrect	10	60	77	4
external environment misunderstood	0	0	0	0
clerical error	0	2	3	0

Table 4.19b - Types of Errors Found by Each Programmer

Type of Error	Number of Errors			
	Lead	Senior	Junior	Librarian
requirements incorrect	1	0	1	0
requirements misinterpreted	0	4	0	0
design incorrect	17	9	4	0
design misinterpreted	0	0	0	0
code incorrect	13	33	97	8
external environment misunderstood	0	0	0	0
clerical error	0	3	2	0

Table 4.19c - Error Rate (Errors per thousand lines of code)

Type of Code	Number of Errors per 1000 LOC			
	Lead	Senior	Junior	Librarian
Text	38.6	34.4	51.1	25.5
Non-blank lines	22.0	19.2	24.1	20.7

more fully than the other programmers' code. In addition, they implemented most of the improvements of clarity, maintainability and documentation.

The librarian who had the least programming experience spent the least time on coding; he wrote only one module, 1.74% of the total amount of non-blank lines of code. He was mainly responsible for librarian duties. He requested only 4% of the changes, and these were mostly error corrections. He made four coding errors but caught a total of eight coding errors.

A breakdown by author of errors classified as language, problem and clerical is presented in Table 4.20a. The senior programmer was responsible for 67% of the problem errors, probably because he worked mostly on design. He and the junior programmer made 83% of the language errors. Furthermore, the junior programmer made over half of all the syntax errors and had the highest rate for these errors as shown in Table 4.20b. However, it should be noted that the junior programmer probably tested his own code more thoroughly than the other programmers' code using the ROLM compiler.

Table 4.21a shows the errors categorized by Ada language feature for each programmer and Table 4.21b shows the errors involving each feature normalized by usage for each programmer. Although tasking, generics and compilation units were not used much, they presented the most problems to all the programmers who used them. Again, it should be emphasized that these features are unique to Ada. The lead programmer had the highest error rate for these three features but he did not use them as much as the junior and senior programmers did. Exceptions were

Table 4.20a - Error Classification by Programmer

Error Class	Number of Errors				
	Lead	Senior	Junior	Librarian	TOTAL
language	25	59	78	4	166
concept	2	5	0	0	7
syntax	17	34	60	3	114
semantics	6	20	18	1	45
problem	3	14	4	0	21
clerical	0	2	3	0	5

Table 4.20b - Syntax Errors per Thousand Lines of Code for Each Programmer

Type of Code	Number of Syntax Errors per 1000 LOC			
	Lead	Senior	Junior	Librarian
Text	23.4	15.6	36.1	19.1
Non-blank lines	13.4	8.7	17.0	15.5

used a total of 117 times and only the lead programmer had difficulty using this feature. Two of the three exception errors he made were concept errors. Eighty-eight percent of the PRAGMA usage was by the lead programmer and both of the PRAGMA errors were made by him. Six out of the eleven uses of access types were by the senior programmer; all three of the errors were made by him. Being a recent computer science graduate, the junior programmer was most familiar with Pascal. It is interesting to note that he was not responsible for any of the concept errors.

Table 4.22 shows how the programmers responded to questions on the error description forms regarding understanding Ada features. As expected, the senior programmer and the junior programmer who were the most well-versed in high-level languages understood features in Ada and tried to apply them, sometimes unsuccessfully as evidenced by the errors. However, they recognized them readily and tried to correct them. The lead programmer had the most difficulty understanding Ada features.

Table 4.21a -Errors Categorized by Ada Language Feature

Ada Language Feature	Number of Errors				
	Lead	Senior	Junior	Librarian	Total
semicolon	0	3	13	1	17
parenthesis	3	5	4	0	12
colon	2	1	0	0	3
:=	0	2	4	0	6
quotes	1	2	1	0	4
comment	0	2	2	0	4
identifier	1	1	3	0	5
loop	4	2	5	0	11
CASE	1	0	0	0	1
IF	0	2	3	1	6
BEGIN/END	0	1	3	0	4
RETURN	0	0	1	0	1
scoping	1	0	1	0	2
typing	0	3	3	0	6
aggregate	0	1	0	0	1
strings	0	1	0	0	1
arrays	0	3	2	0	5
records	0	1	3	0	4
access type	0	3	0	0	3
declarations	0	4	8	1	13
parameters	2	4	3	1	10
procedures/ functions	1	3	7	0	11
tasking	1	5	2	0	8
exceptions	3	0	0	0	3
generics	1	2	5	0	8
packages	0	1	1	0	2
compilation units	2	3	2	0	7
attributes	0	1	0	0	1
PRAGMA	2	0	0	0	2
file input/output	0	2	0	0	2
overloading	0	1	2	0	3

Table 4.21b - Errors Normalized with Respect to Feature Usage for Each Programmer

Ada Language Feature	Percentage of Errors			
	Lead	Senior	Junior	Librarian
semicolon	0	0.20	0.97	0.03
parenthesis	9.3	4.10	2.65	0
colon	2.74	0.13	0	0
:=	0	0.81	1.29	0
comment	0	0.10	0.09	0
identifier	0.76	0.67	4.23	0
loop	8.00	3.77	9.26	0
CASE	3.33	0	0	*
IF	0	3.13	3.13	6.25
BEGIN/END	0	1.64	3.26	0
RETURN	0	0	1.64	0
aggregate	0	0.14	0	0
strings	0	0.68	0	*
arrays	0	0.48	0.41	0
records	0	0.36	0.88	*
access type	0	50.00	0	*
declarations	0	0.54	1.70	3.22
parameters	6.25	1.01	1.60	8.33
procedures/ functions	4.55	0.89	3.50	0
tasking	100.00	38.46	28.57	*
exceptions	11.11	0	0	0
generics	50.00	11.76	35.71	*
packages	*	20.00	9.10	*
compilation units	50.00	15.00	8.00	0
attributes	0	1.85	0	0
PRAGMA	14.29	*	0	*

Note - An "*" means that this programmer never used this feature.

Table 4.22 - Understanding Features By Programmer

UNDERSTANDING FEATURES BY PROGRAMMER	Lead	Senior	Junior	Librarian
no reply	3	3	2	0
understood features separately, but not their interaction	1	1	6	1
understood features but did not apply them correctly	15	44	83	5
did not understand features fully	12	1	12	2
confused feature with a feature in another language	0	0	1	0

5. SUMMARY AND CONCLUSIONS

This report analyzes the data from the development of a system in Ada by four programmers at GE. The data analyzed include effort, change and error data as well as basic metrics on the size of the project and features of the language used. The project was not completed, and little time was spent on testing. (Only unit testing was done and even that was not completed.) Several modules were never coded.

The majority of the changes were error corrections. Most of the remaining changes were improvements of clarity, maintainability or documentation. The highest level document changed was the code module in most cases. The vast majority of the coding errors were code incorrect. In addition, design incorrect accounted for 16%. There are probably many more errors still in the system since it was not fully tested.

A large number of language errors were made. Many involved Ada-specific features. The programmers used most of the language features of Ada, but not necessarily as they were intended by the language designers. The error rates of the Ada-specific features were generally very high. There were only a few concept errors, and these involved tasking, exceptions, access types, packages and file input/output. Tasking, generics and compilation units were not used much, but they presented the most problems to all the programmers who used them. These features are unique to Ada.

The need for change was determined in less than an hour for almost all of the changes. In addition, the time to design and implement the change was one hour or less for almost all of the changes. Most errors took less than fifteen minutes to isolate and as little time to correct. (Many of the errors were syntax errors.)

Because of the learning curve, it was not possible to judge the impact of Ada on schedules, costs or milestones. Because Ada is a completely new language with features not present in other programming languages, about 20% of the total effort was spent on training and methodology, which is more than the effort on any other phase of the project. This is a much higher percentage than would typically be spent on most projects. Furthermore, even this amount seems insufficient because the programmers indicated that they did not feel comfortable with Ada until after they left the project.

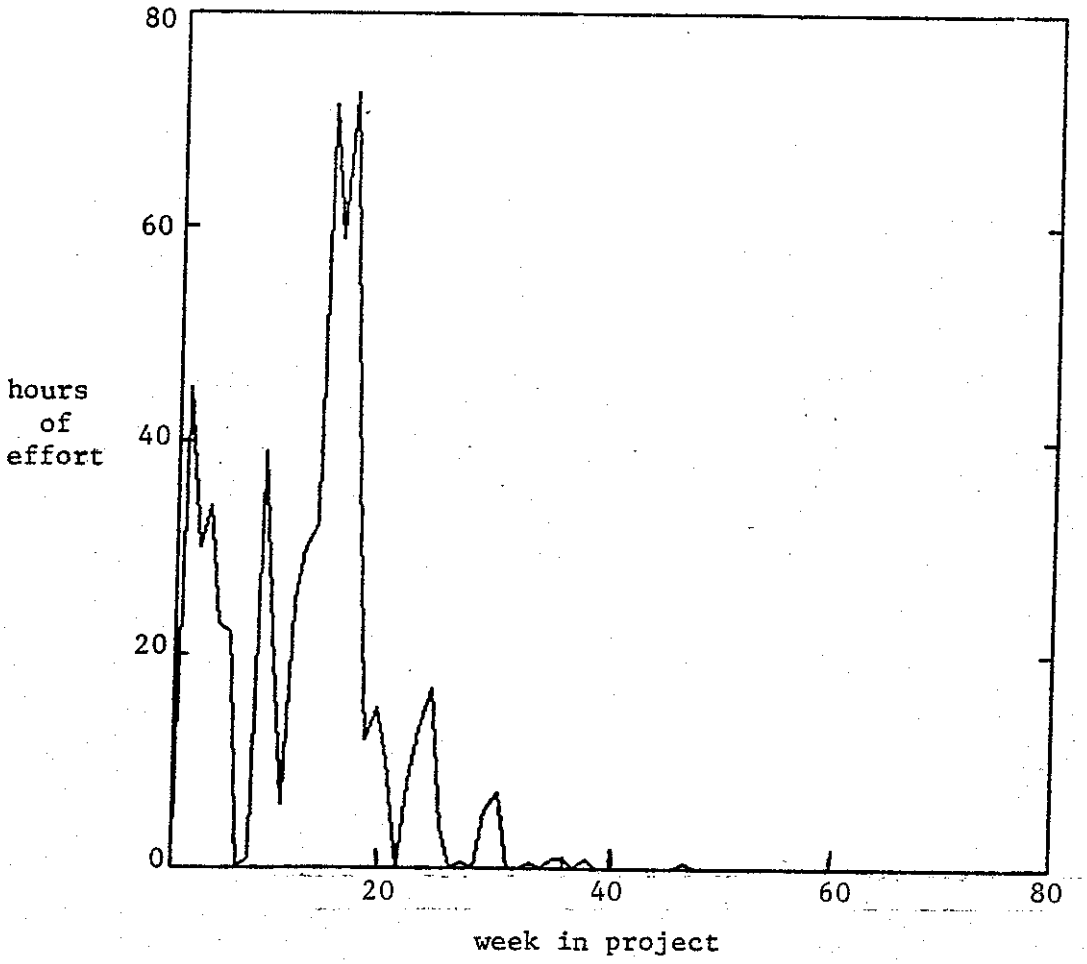
Lack of support tools discouraged the programmers. This shows that automated tools are paramount for success in a software project. This is especially true given a language with the complexity of Ada. Many syntax errors were uncovered, and programmers could have spent this time looking for logic errors. Tools needed include a structured editor, data dictionaries, call structure and compilation dependency tools, and cross references. Errors could be caught earlier in development with a PDL processor. (The earlier an error is caught, the less expensive it is to fix in general.)

The greatest error rate appeared to be associated with the most Ada specific features: tasking, generics and compilation units. The lead

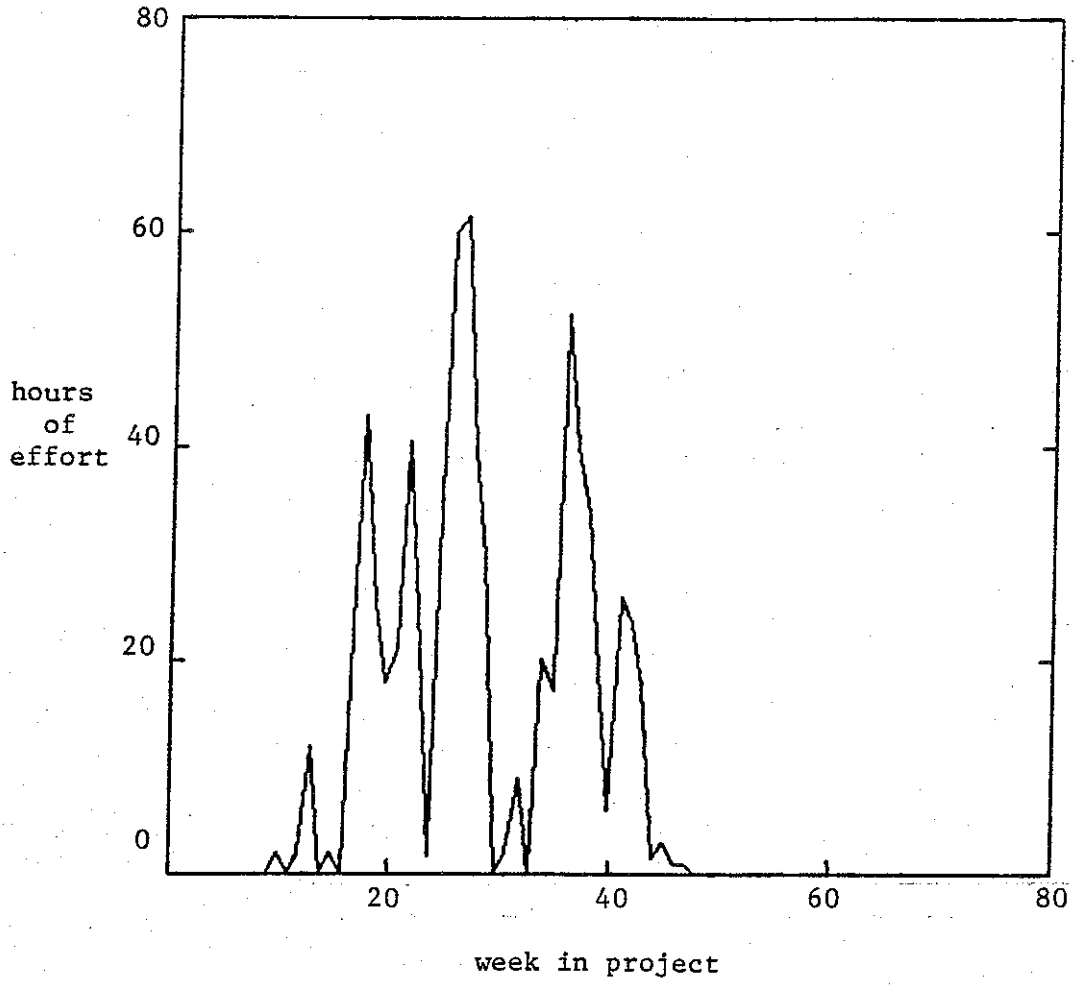
programmer had the highest error rate in these categories. The junior programmer did not make any of the concept errors and he seemed to have the easiest time grasping ideas in Ada. The junior programmer had recently graduated, while the lead programmer had worked in industry for many years.

There is further analysis that can and will be done. The effectiveness of the features will be examined. The use of packages has already been studied [Gannon, et al. 83].

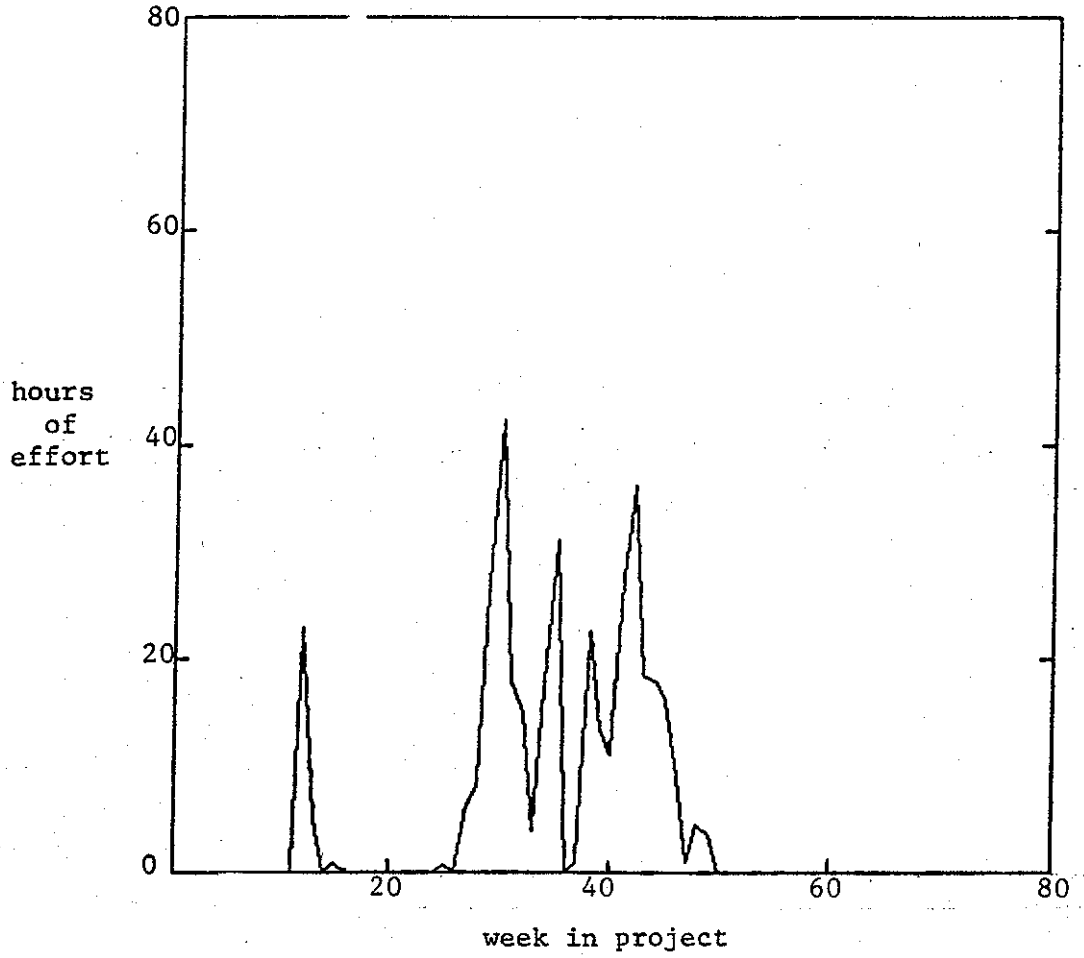
Appendix 1.1 Distribution of Requirements Effort Over Time



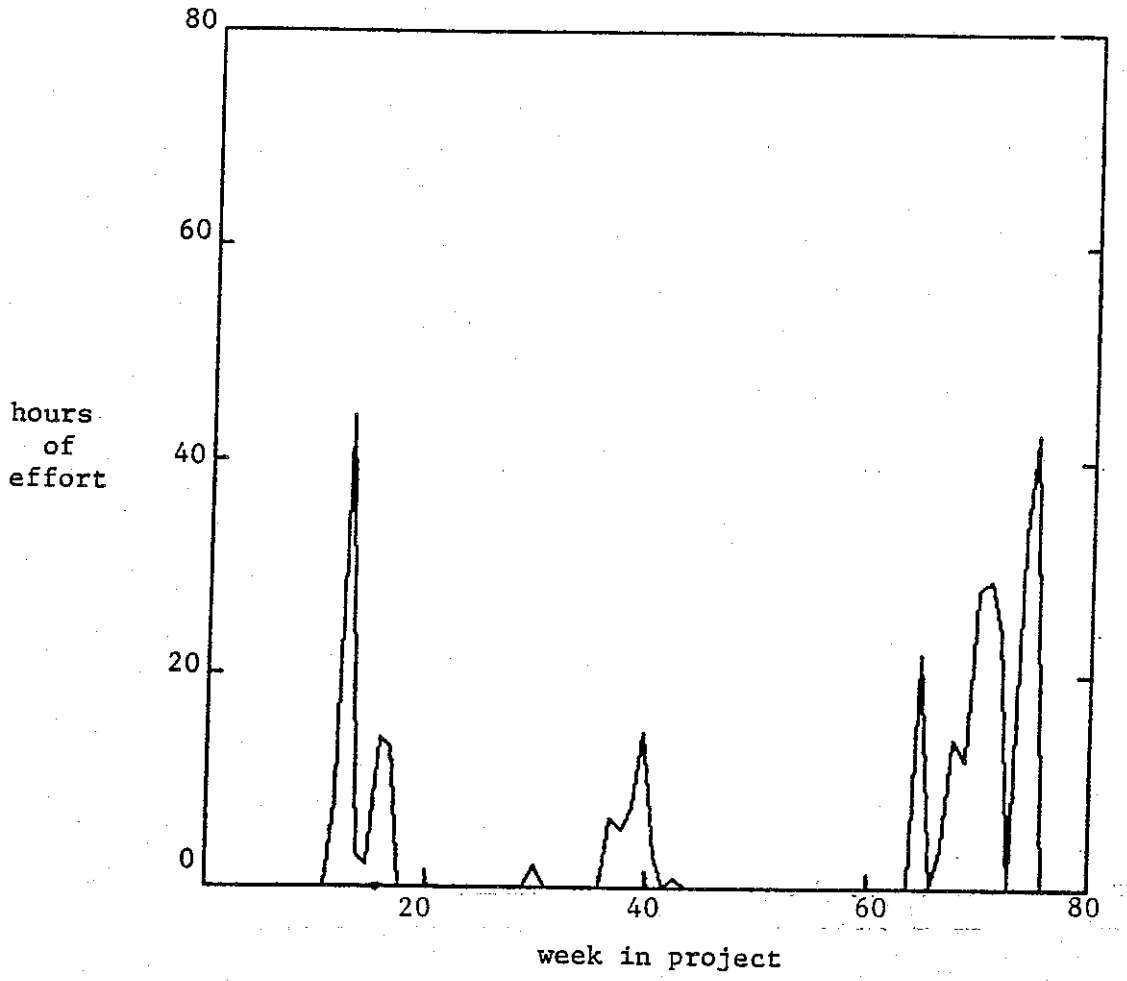
Appendix 1.2 Distribution of Design Effort Over Time



Appendix 1.3 Distribution of Coding Effort Over Time



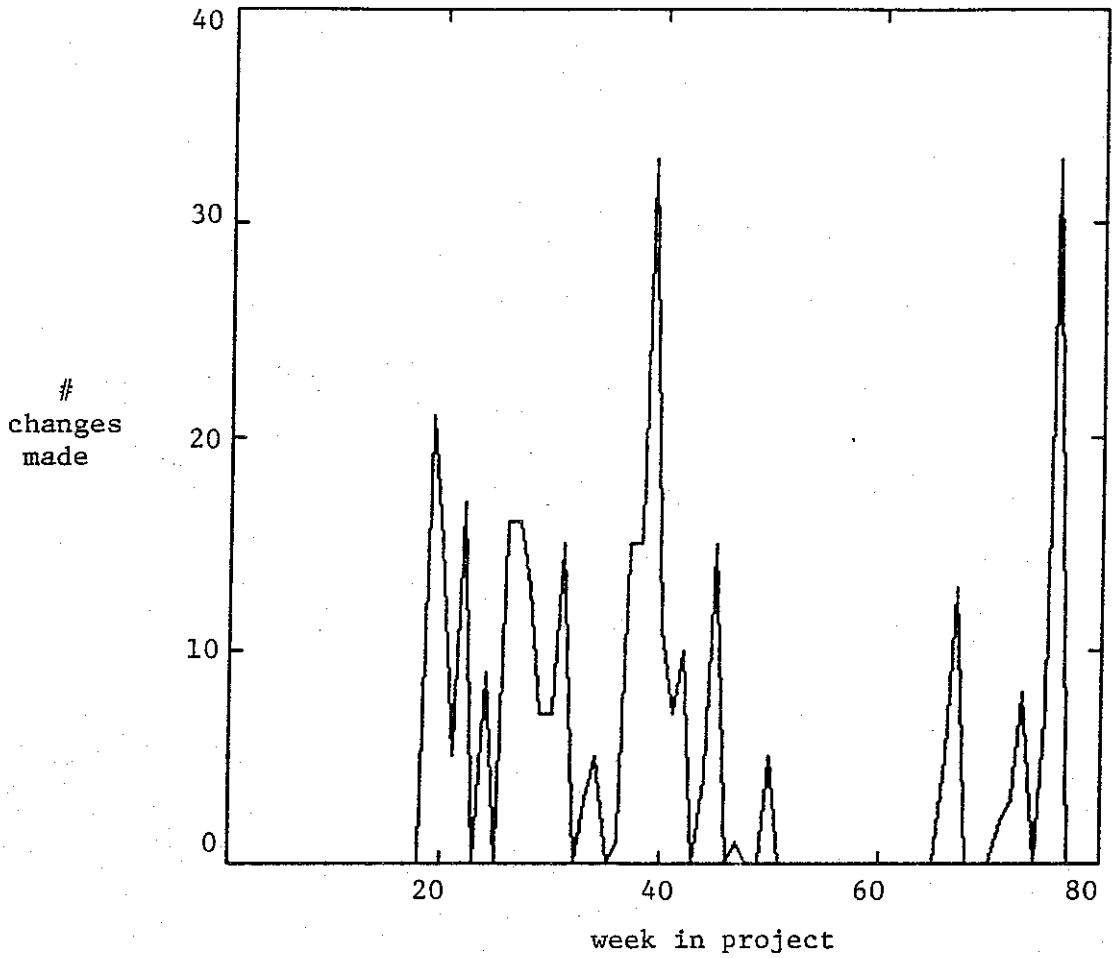
Appendix 1.4 Distribution of Testing Effort Over Time



Appendix 2 - Improvements of Clarity, Maintainability or Documentation by Type of Document

Type of Document	Number of Changes	Percentage
Requirements	14	18.42%
PDL	54	71.05%
Code Module	8	10.53%
Total	76	100.00%

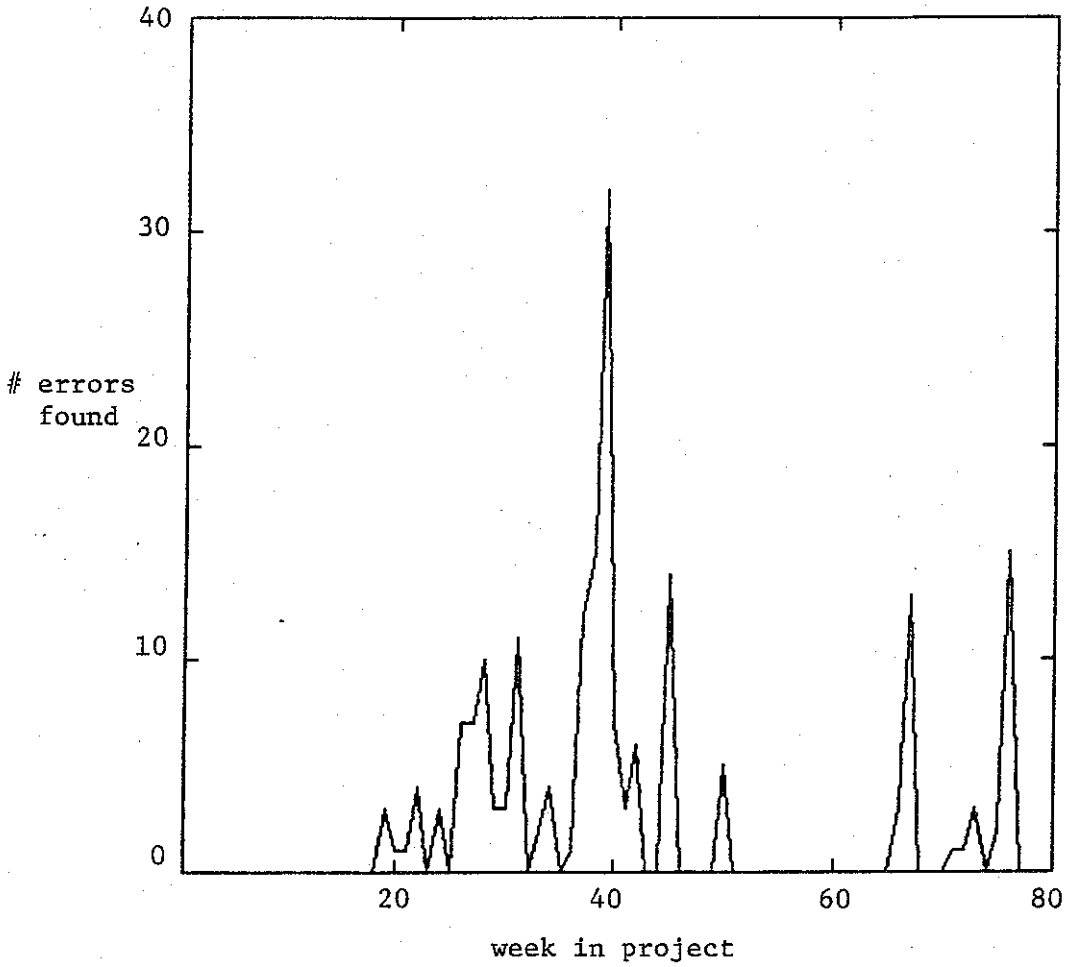
Appendix 3. Distribution of Changes Over Time



Appendix 4 - ACTIVITIES USED TO DETECT AND ISOLATE ERRORS

	DETECTING ERROR:		ISOLATING SOURCE:	
	ACTIVITIES USED FOR PROGRAM VALIDATION	ACTIVITIES SUCCESSFUL IN DETECTING ERROR SYMPTOMS	ACTIVITIES TRIED TO FIND CAUSE	ACTIVITIES SUCCESSFUL IN FINDING CAUSE
design reading	34	24	19	17
design walkthrough	35	31	18	15
code reading	58	31	57	57
code walkthrough	2	1	3	2
talk with other programmer	2	2	2	2
reading documentation	3	1	6	4
compiler messages	105	105	76	75
system error messages	4	4	3	3
project error messages	1	1	1	1
trace	5	5	6	6
dump	0	0	0	0
inspection of output	1	1	1	0
pre-acceptance test run	4	2	1	1
acceptance test	0	0	0	0
Ada runtime check	0	0	0	0
other	4	4	2	2

Appendix 5. Distribution of Errors Over Time



Appendix 6 - FOR AN ERROR IN THE PDL OR CODE

DOES THE DOCUMENTATION EXPLAIN THE FEATURE CLEARLY?

not applicable	16	8.33%
yes	157	81.77%
no	19	9.90%

WHICH OF THE FOLLOWING IS MOST TRUE?

no reply	8	4.17%
understood features separately, but not their interaction	9	4.69%
understood features but did not apply them correctly	147	76.56%
did not understand features fully	27	14.06%
confused feature with a feature in another language	1	0.52%

WHERE THE INFORMATION NEEDED TO CORRECT ERROR WAS FOUND

no place specified	4
class notes	6
Ada reference manual	7
another programmer	19
remembered	144
viewgraphs from tapes	0
test program	1
other	16

Appendix 7.1

CHANGE REQUEST = _____

Person requesting change _____ Approved by _____ Date ___/___/___

1. What is the reason for this change? _____

_____

2. When was the need for the change determined? ___/___/___

3. Describe the necessary change: _____

_____

_____

4. To completely implement the desired change, the highest level document that needs changing is:

(complete one:) requirements section: _____

PDL module: _____

code module: _____

5. When did the effort begin to understand and isolate this change? ___/___/___

6. How much effort has been spent so far in isolating and understanding what needs to be changed?

_____!_____!_____!_____!_____!_____!_____

1 hr. 4 hrs. 2 days 1 wk. 2 wks. 1 mo. 2 mos.

7. This change is a/an (check one):

_____ error correction (attach completed ERROR DESCRIPTION FORM)
_____ change in the problem domain
_____ planned enhancement
_____ avoidance of an apparent problem with the compiler (explain below)
_____ avoidance of some other problem in the development environment (explain below)
_____ adaptation to a change in the development environment (describe below)
_____ improvement of clarity, maintainability or documentation
_____ optimization of time, space or accuracy
_____ insertion or deletion of debug code
_____ other than above (describe below)

(over)

8. List all known documents which will require a change as a result of this change request to maintain system consistency:

Rqmts.	PDL	Code	section nos./module name	(to be filled in by librarian) date change submitted to librarian
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___
				___/___/___

(Note that <u>each</u> of these will require a <u>separate</u> Individual Document Change Report.)

9. What <u>additional</u> documents were examined or will be examined in determining the chang

Rqmts.	PDL	Code	section nos./module name

Appendix 7.2

ERROR DESCRIPTION FORM for CHANGE REQUEST #_____

1. Type of Error:

   _____ requirements incorrect
   _____ requirements misinterpreted
   _____ *design incorrect
   _____ *design misinterpreted
   _____ *code incorrect
   _____ external environment misunderstood (not language or compiler)
   _____ clerical error

   *Was the error in the use of data ___ or in function ___ ?

2. Did the use of Ada as a design and implementation language contribute to this error? ____ If so, was it only a syntax error? ____

3. Whether related to Ada or not, which language features were involved in the error?

   _____

   _____

4. For an error in the PDL or code:

   a. does the documentation explain the feature clearly?____Yes____No

   b. which of the following is most true?

      _____ understood features separately, but not their interaction
      _____ understood features but didn't apply them correctly
      _____ didn't understand features fully
      _____ confused feature with a feature in another language

   c. where was the information needed to correct the error found?

      _____ class notes
      _____ Ada reference manual
      _____ another programmer
      _____ remembered
      _____ viewgraphs from tapes
      _____ test program
      _____ other:

5.

	Detecting error:		Isolating source:	
	Activities Used for Program Validation	Activities Successful in Detecting Error Symptoms	Activities Tried to Find Cause	Activities Successful in Finding Cause
Design reading				
Design walkthrough				
Code reading				
Code walkthrough				
Talk w/other programmer				
Reading documentation				
Compiler messages				
System error messages				
Project error messages				
Trace				
Dump				
Inspection of output				
Pre-acceptance test run				
Acceptance test				
Ada runtime checking				
Other:				

6. What was the time used to isolate the source of the error?

1 hr.	4 hrs.	2 days	1 wk.	2 wks.	1 mo.	2 mos.

If never found, was a workaround used? _____ (Explain in 8, below.)

7. When did the error enter the system?

_____ requirements _____ design _____ Ada coding _____ testing

_____ implementing another change, Change Report No. _____

_____ other or can't tell (explain below)

8. Use this space to give any additional information that might help in understanding the cause of the change and its ramifications:

Appendix 7.3

INDIVIDUAL DOCUMENT CHANGE REPORT

1. This change has been approved through CHANGE REQUEST #_____

2. Document being changed (complete one):

   requirements spec. section _____

   PDL module: _____

   code module: _____

3. How much effort (person-time) was spent changing this document (not librarian's time)?

15min	1hr	4hrs	1day	2days	1wk	2wk	1mo

4. Person responsible for this change _____ Date ____/____/____

5. Instructions to Librarian: Priority (H, M, L) _____

   See Listing _____

   Other (Explain) _____

   Should this module be compiled? _____

Remainder of form to be completed by Librarian:

1. Is the Change Request indicated in question 1 of this form on file? _____
   (If not, do not make change. Return form incomplete.)

2. Has question 9 of the Change Request indicated in question 1 here been updated by author of this form? _____

3. If this is a change to a compiled module of design or code, list any other modules subsequently requiring recompilation: _____

Date Change completed ____/____/____ (or returned incomplete ____/____/____)

Appendix 7.4

COMPONENT STATUS REPORT

PROGRAMMER ______________________ DATE ____________

COMPONENT	REQUIREMENTS		DESIGN			CODE DEVELOPMENT			TEST			OTHER	
	ANALYSIS	WRITING	CREATE	READ	FORMAL REVIEW	CODE	READ	FORMAL REVIEW	UNIT	INTEG	REVIEW	ACTIVITY	HRS
												TRAVEL	
												FORMS	
												MEETINGS	
												ACC TEST	
												TRAINING	

Selected References

[Basili et al. 82]
Victor R. Basili, John D. Gannon, Elizabeth E. Katz, Marvin V. Zelkowitz, John W. Bailey, Elizabeth E. Kruesi, and Sylvia B. Sheppard, "Monitoring an Ada Software Development Project," Ada Letters II, 1 (July 1982), 1.58-1.61.

[Basili, Katz 83]
Victor R. Basili and Elizabeth E. Katz, "Metrics of Interest in an Ada Development," IEEE Workshop on Software Engineering Technology Transfer, Miami, FL, April 1983, pp. 22-29.

[Basili, Weiss 82]
Victor R. Basili and David M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," Computer Science, Univ. of Maryland, 1982, UOM-1235.

[Duncan, et al. 84]
A.G. Duncan, J.S. Hutchison, J.B. Bailey, T.M. Chapman, A. Fregly, E.E. Kruesi, T. McDonald, D. Merrill, S.B. Sheppard, "Communications System Design Using Ada," Proc. 7th Intl. Conf. on Software Engineering, Orlando, FL, March 1984, pp. 398-407.

[Gannon, et al. 83]
John D. Gannon, Elizabeth E. Katz, and Victor R. Basili, "Characterizing Ada Programs: Packages," The Measurement of Computer Software Performance, Los Alamos National Laboratory, August 1983.