RADC-TR-88-213
Final Technical Report
October 1988

AN EXPERIMENTAL INVESTIGATION INTO SOFTWARE RELIABILITY

Syracuse Universtiy

Amrit L. Goel

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This effort was funded totally by the Laboratory Director's fund.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-88-213 has been reviewed and is approved for publication.

APPROVED: Joseph P Cavano

JOSEPH P. CAVANO
Project Engineer

APPROVED: Raymond P. Urtz

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: John A. Ritz

JOHN A. RITZ
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific doucment require that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-213
6a. NAME OF PERFORMING ORGANIZATION Syracuse University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)
6c. ADDRESS (City, State, and ZIP Code) CASE Center Syracuse NY 13244-5300	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) COEE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-81-C-0193
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 61101F PROJECT NO.: LDFP TASK NO: 04 WORK UNIT ACCESSION NO: C5

11. TITLE (Include Security Classification)
AN EXPERIMENTAL INVESTIGATION INTO SOFTWARE RELIABILITY

12. PERSONAL AUTHOR(S)
Amrit L. Goel

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Nov 85 TO Jan 87	October 1988	126

16. SUPPLEMENTARY NOTATION
This effort was funded totally by the Laboratory Director's Fund.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	05		Software Reliability Software Measurement
12	08		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report presents the results of an experiment investigating the effect of Fortran and Ada languages on program reliability. The experimental design employed was a 2^2 full factorial design, i.e., a design in two variables, each at two levels. The problem used in the experiment was the Launch Interceptor Program (LIP), a simple but realistic anti-missile system. Reliability comparisons between Ada and Fortran programs were based on the total number of errors as well as on errors found during various testing phases. Some comparisons were also based on error density, the number of errors per 100 non-comment lines of code. It was found that on the average, the Ada programs had about 70 percent less errors than the Fortran ones. If errors during unit testing were excluded, the Ada program had about 78 percent less errors. Similar differences were found for data based on error causes and error types.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Joseph P. Cavano	22b. TELEPHONE (Include Area Code) (315) 330-4476 22c. OFFICE SYMBOL RADC (COEE)

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ACKNOWLEDGEMENT

The work reported here was initiated by Joseph Cavano of RADC who was also the technical monitor for this research project. Significant computing and other assistance was provided by F. Farhat and Tom Little of Syracuse University. Thanks are due to all these individuals for their support and assistance throughout the course of this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

EXECUTIVE OVERVIEW

This report documents an experiment investigating the effect of Fortran and Ada languages on program reliability. The experimental design employed here was a 2^3 full factorial design, i.e, a design in three variables, each at two levels. The variables and their levels were language (Fortran and Ada), programmer experience (intermediate and advanced) and application type (scientific and text processing). Due to experimental and resource constraints, this study concentrated on a 2^2 factorial design in two variables, language and programmer experience. Some of the experimental points in this design were replicated to determine precision of the effects of language and experience on program reliability. The scientific problem used here was the Launch Interceptor Program (LIP), a simple but realistic anti-missile system which has been used elsewhere in connection with software testing and fault tolerance research. The second problem used was the well-known text formatting program. The programmers were graduate students in computer engineering with varying degrees of experience in programming languages.

The programmers developed their own designs from given specifications and did indepandent compilations and unit testing. Data on these activities were collected for comparison purposes. The function testing of each LIP version was done after removing errors detected during unit testing and compilation. Fifty-four test cases for this purpose were developed manually using a hybrid of structure-dependent and structure-independant testing techniques.

Since it was not feasible to determine the oracle for the LIP program, a variation of majority voting technique was used to determine the correct output. Specifically, elements

of the conditions met vector, were compared from each version and the majority was taken to be the correct value. This technique proved to be very useful and efficient in testing and debugging. Also, this method was successful in providing a very high structural and functional coverage of the programs.

Errors revealed by each of the above fifty-four test cases were removed one-at-a-time and the remaining test cases were run on the incrementally corrected versions. The plots of cumulative error symptoms versus test case number seem to follow homogeneous Poisson processes in each case and provide some useful insights into the error symptom occurrence behavior.

Operational testing of the LIP programs was done after removing errors found during function testing using three sets of test cases. The first set of 120 tests was developed manually using the same hybrid techniques as for function testing. The other two sets of 100 and 1000 were random tests.

Data on errors were collected during development and unit testing, function testing and operational testing. This included number, causes and types of error and times taken to isolate and fix them. Some data on productivity, design and code metrics, as well as on structural coverage during testing were also recorded. The key measure of reliability used in this study is the number of errors found during various development phases.

Reliability comparisons between Ada and Fortran programs were based on each of the following categories: the total number of errors, number of errors found during development and functional testing, number of errors found during functional and operational testing, and errors found during operational testing alone. Some comparisons were also based on

error density, defined as the number of errors per 100 non-comment lines of code. Further analyses were based on error causes as well as on error types. The main results of these analyses are given below.

The number of errors in the Ada programs were about 70% less than those in Fortran when comparisons were based on errors found during all phases. If errors during development and unit testing were excluded, i.e , if only functional and operational testing data were considered, the Ada programs had about 78% less errors. Similar differences were found for data based on error causes and error types.

Using error density during development and functional testing as a measure, the average difference between Ada and Fortran programs was 5.70 errors per 100 non-comment lines with a standard deviation of 1.35 units. The effect of programmers' experience was to reduce error density by 2.1 units with a standard deviation of 1.35 units. If data during functional and operational testing alone is considered, then the Ada programs had 3.2 less errors per 100 non-comment lines. The effect of experience during these phases was not statistically significant.

Regarding the development effort, times spent in specification reading, design and coding was almost the same for all programs. However, unit testing efforts differed due the differences in the extent of such testing. Analyses of the data on error isolation and correction indicated that the Ada programs were easier to debug than the Fortran programs.

Summarized above are the main results of this study. Even though they are based primarily on implementations of one problem, they do indicate that there is a statistically

significant evidence in support of higher reliability of Ada over Fortran programs. The extent of this difference, however, is likely to vary from one application to another as well as across different development environments.

1. INTRODUCTION

1.1 Introduction

An important goal of any software development activity is to ensure that the resulting programs have a very high probability of being correct. This probability can be increased by reducing the probability of programming errors and by increasing the probability of detecting any errors that are made during program development.

It is well known that the characteristics of the programming language can have a significant impact on the number and types of the errors in a program. Modern languages provide a variety of features to help ensure correctness, i.e., to ensure high reliability. However, programs in some langauges tend to be more reliable than those in others because of some specific features. For example, Ada possesses many more features than Fortran to reduce program errors. Many errors, such as type mismatches, array indices out of bounds, and incorrect ranges of variables are detected during compilation in Ada and may not be found until testing or operation in Fortran programs. This will tend to yield a higher reliability of Ada over the Fortran programs.

While certain language features may reduce the occurrence of errors, there are more possibilities of the introduction of errors during problem implementation due to difficulties in learning and due to possible incorrect use of these features. Also, it is not known a-priori whether the combined use of specific features will affect program reliability and even if it did, the extent of such effect is not known.

1.2 Study Objectives

The objective of this study is to perform a comparative assessment of the reliability

of Ada and Fortran implementations via controlled experiments. The rationale for an experimental comparison of reliability is that many hypotheses have been postulated about the role of Ada features in improving program reliability. Even though it may be possible to prove each hypothesis by itself, their overall impact is not possible to prove, due to the interdependencies among these features and can best be observed in a controlled experimental setting.

Ada is of particular interest because it has been developed to become the language of the Department of Defense. Applications of Ada range from data-processing to embedded microprocessor-based control systems. Reliability and ease of maintenance have been two critical issues driving its development. Fortran continues to be the workhorse of the computer industry; certainly thousands of programs have been developed and many more continue to be developed in Fortran costing a tremendous amount of money. It is the purpose of this investigation to provide some insights into the relative reliability of implementations in Ada and Fortran languages that could be useful in choosing one or the other using reliability as a criterion.

1.3 Report Overview

Section 2 of this report provides a description of the experimental design, the problems chosen for the study, the experimental environment and details of testing. The data collected during various phases of development are given in Section 3. Analyses of the data and reliability comparisons of the Fortran and Ada implementations are discussed in Section 4. Finally, conclusions of this study and recommendations for further work are summerized in Section 5.

2. EXPERIMENTAL DETAILS

The purpose of this section is to provide details of the experimental approach employed in this study. The experiment design employed is described in Section 2.1 and a description of the problems used for implementation is given in Section 2.2. The background of the programmers who participated in this study is summarized in Section 2.3 and a brief description of the computer systems and software tools employed is given in Section 2.4. A detailed description of the experimental plan for the LIP programs is given in Section 2.5.

2.1 Experimental Design

The original experimental design was a full factorial design in 3 variables, each at two levels, as follows:

Language :	Fortran and	Ada
Programmer		
Experience :	Intermediate and	Advanced
Application:	Scientific and	Text Processing

A diagramatic representation of the various combinations of these variables along with their levels is shown in Figure 2.1. In this diagram each vertex of the cube represents an experimental condition, i.e., a combination of the values of the three variables. For example, vertex number 1 represents a program to be written in Fortran by an intermediate programmer for the scientific application. Similarly, vertex number 6 represents the experimental conditions for a program to be written in Ada by an intermediate programmer to implement the text processing problem.

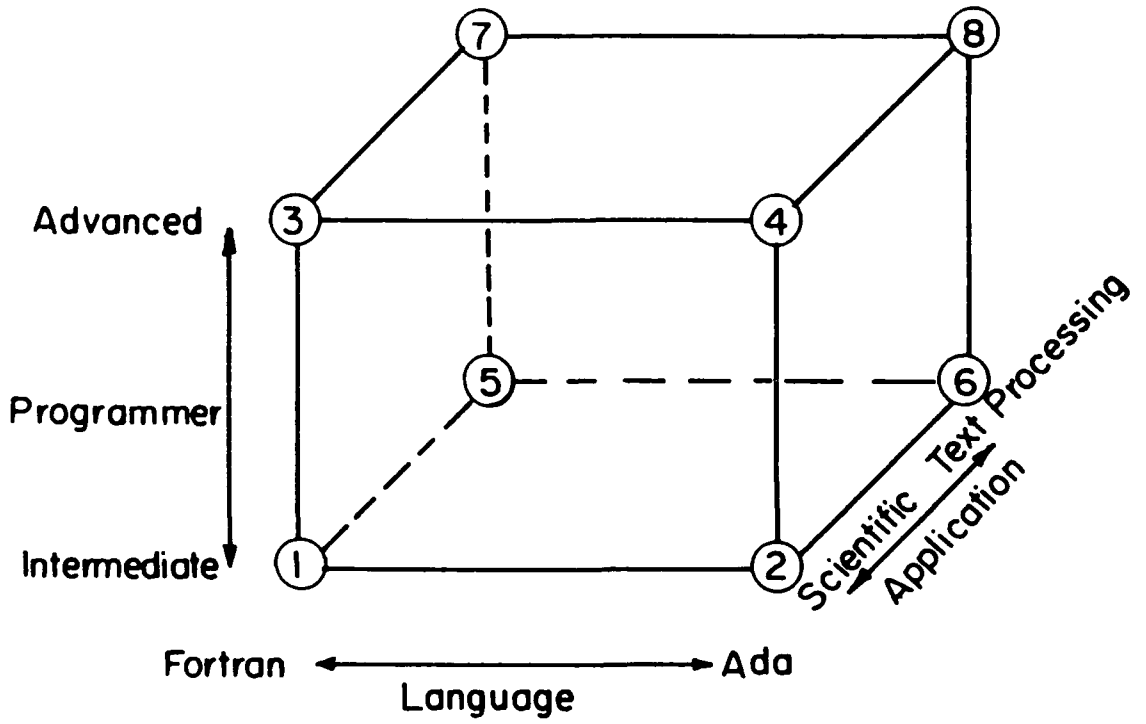

Figure 2.1. Diagrammatic Representation of Experimental Design

A complete set of experiments for this 2^3 factorial design consists of eight experimental conditions, i.e., eight different programs to be developed according to the combinations in Figure 2.1. The results of this experiment can provide estimates of the effects on reliability, i.e., of the change in reliability due to each of the three variables. It can also provide estimates of the interaction effects among these variables. However, precision of these effects cannot be assessed from this experimental set up unless certain interaction effects can be assumed to be small and then used to estimate experimental error.

In order to determine the precision of these effects, we can replicate all the experimental conditions leading to a total of sixteen programs, two for each of the eight experimental combinations in Figure 2.1. The data from these replicated programs can then be used to estimate error which can provide a measure of the precision of the effects on reliability.

After careful consideration of the above issues, vis-a-vis the availability of resources and the scope of this study, it was felt that developing sixteen programs was infeasible. The choice then was to reduce the number of variables and eliminate some replications. In order to obtain the most information with limited resources, it was decided to concentrate on the scientific application with some replications to determine the precision of estimates. Also, it was decided to develop two programs for the text processing application. Based on these considerations, two versions of scientific problem were developed for each of the experimental condition 1 and 2, and one each for experimental conditions 3, 4, 5 and 6 for a total of six versions. Two versions of the text processing program were also developed. Thus, a total of eight different implementations were used in this study, six representing a full factorial in two variables with some replications and two representing a fractional

factorial.

2.2 Description of the Problems Used

The two problems used here were chosen to be representative of the applications that seemed to be most relevant to the objective of this study. The scientific and the text processing applications were a launch interceptor and a text formatting problem, respectively. These are described below.

2.2.1 Launch Interceptor Problem

It is a simple but realistic anti-missile system which has been implemented before in connection with software reliability and fault-tolerance research [Knight & Leveson 86]. The program uses as input data that represent radar reflections and computes a set of conditions based on some inter- relationships among these points. There are the so-called Launch Interceptor Conditions (LIC's) which take true or false values, i.e., each one is either satisfied or not in accordance with a large set of input parameters. The boolean value of the satisfied LIC's are combined into a vector called the conditions met vector (CMV) that is further reduced to produce a final launch or no-launch output signal.

The elements of CMV(CMVE's) are operated upon by another input, the Logical Connector Matrix (LCM). This matrix consists of boolean operators AND, OR, and (NOTUSED), and serves to combine the satisfied LIC's as defined by the calling program. The results of this operation are the off-diagonal elements of a boolean matrix called the Preliminary Unlocking Matrix (PUM). The diagonal elements of the PUM are input values and are used to formulate the final unlocking vector (FUV). These individual elements determine whether the rows of the PUM are to be considered in the final decision to launch

interceptors, in the generation of the FUV.

Finally, the decision to launch is made based on the boolean values in the FUV. If and only if all of the elements of the FUV are TRUE will the signal LAUNCH be set to TRUE.

A detailed set of functional requirements for this program is given in Appendix A.

2.2.2 Text Formatting Problem

The specifications of the second problem, the text formatting problem, are based on those given in [Naur 69]. This problem has also been addressed earlier in [Goodenough and Gerhart 77] and [Meyer 85] in connection with specifications and testing research. More recently, it was used in [Selby 85] for an experimental investigation of the software development and testing approaches. The following description of the problem is taken from this reference.

Given an input of text characters up to 80 characters consisting of words separated by blanks or new-line characters, the program formats it into a line-by-line form such that 1) each output line has a maximum of 30 character, 2) a word in the input text is placed on a single output line, and 3) each output line is filled with as many words as possible.

The input text is a stream of characters, where the characters are categorized as either break or nonbreak characters. A break character is a blank, a new-line character, or an end-of-text character. New-line characters have no special significance; they are treated as blanks by the program. The new-line and end-of-text characters should not appear in the output.

A word is defined as a nonempty sequence of nonbreak characters. A break is a

sequence of one or more break characters and is reduced to a single blank character on start of a new line in the output.

When the program is invoked, the user types the input line, followed by a character and a carriage return. The program then echos the text input and formats it on the terminal.

If the input text contains a word that is too long to fit on a single output line, an error message is typed and the program terminates. If the end-of-text character is missing, an error message is issued and the program awaits the input of properly terminated line of text.

2.3 Programmers' Background

Six programmers were involved in this study, all graduate students in the Department of Electrical and Computer Engineering at Syracuse University with good academic standing. One programmer was a full time software professional in local industry. The experience of the programmers in several programming languages, including Fortran and Ada, is summarized in Table 2.1. These programmers will be referred to as FL1, FL2, FL3, AL1, AL2, and AL3 in this report. The FL programmers wrote the Fortran programs while the AL programmers wrote Ada programs for the Launch Interceptor Problem (LIP).

From the values in Table 2.1, we note that FL1 and FL2 had about 5 years of programming experience each with an average of 1.5 years in Fortran. Each of AL1 and AL2 had about 7 years of experience with an average of 0.8 years in Ada. These are the four programmers with intermediate level of programming experience.

Each of FL3 and AL3 had about 9 years of programming experience. FL3 had 4 years

TABLE 2.1

Programmers' Experience in Several Languages

	Years of Experience					
Language	FL1	FL2	FL3	AL1	AL2	AL3
Assembly Langauge	0.5	0	3	3	2	3
FORTRAN	1	2	4	0.5	0.5	3
PASCAL	0.8	0.5	1	0.5	2	0
ADA	0	0	0	0.8	0.8	1.5
C	1	0.5	0	0.5	0	1.5
BASIC	1	2	1	1	1	0
Others	1	0	0.5	1	1	0
TOTAL	5.3	5.0	9.5	7.3	7.3	9.0

of Fortran and AL3 had 1.5 years of Ada experience. These are the two programmers with advanced level of programming experience.

In addition to the programming experience listed in Table 2.1, each programmers had taken a graduate level course in software engineering. Thus, all of them were familiar with systematic software development approaches including a detailed knowledge of testing techniques.

2.4 Computer Systems and Software Tools

Various programs used in this study were developed on the computer systems at Syracuse University. Programs written by AL1, AL2, FL2 and FL3 were developed on VAX 11/785 under VMS version 4.4. The program written by FL1 was developed on a VAX 11/780 under BSD 4.2 UNIX and then moved to VAX/VMS for testing. Finally, the program by AL3 was developed on SUN workstation and later transferred to VAX/VMS for testing. The compilers for Fortran and Ada were supplied by DEC to run under VAX/VMS.

Several simple software tools were developed during the course of this investigation. The main tool was MATCHER, which was used to compute the CMV values computed by each of the programs as described later in this report. Also, two simple programs were developed to collect data on CMV combinations tested for determining the structural coverage of the Ada programs during testing.

2.5 Experimental Plan for LIP Development

The LIP development followed the traditional software phases consisting of specifications, top-level detailed design, coding, unit testing, function testing and operational

testing. Relevant data to assess programs reliability was collected in each phase. Some additional data, such as productivity and test coverage were also collected wherever feasible.

A diagrammatic representation of the experimental plan for LIP is shown in Figure 2.2. Note that the detailed designs were developed independently by each programmer. Also, unit testing and compilations were done differently by dfifferent programmers. However, function and operational testing was done using the same test data for each of the programs. Additional relevant details about the development are given in the following subsections.

2.5.1 LIP Designs

All programmers employed a hybrid of functional decomposition and structured design techniques [Bergland 81,Fairley 85] for developing the high-level or architectural designs. The basic approach taken in each case was to decompose the launch decision function into subfunctions which consisted of computing various intermediate quantities, such as the Conditions Met Vector (CMV) and the Preliminary Unlocking Matrix (PUM). These subfunctions were then decomposed into computational steps which involved determination of various conditions. These were further decomposed into computations of geometrical relationships. The basic flow of data underlying the designs is shown in Figure 2.3.

The lower level subfunctions were related to the higher level functions differently by each programmer. Thus, all the designs were different but the basic approach used was essentially the same. The structure chart for one design is given in Figure 2.4.

2.5.2 Design and Code Based Metrics

The number of packages, subprograms, procedures, subroutines, functions and tasks used in the programs are given in Table 2.2. These measures summarize features of the

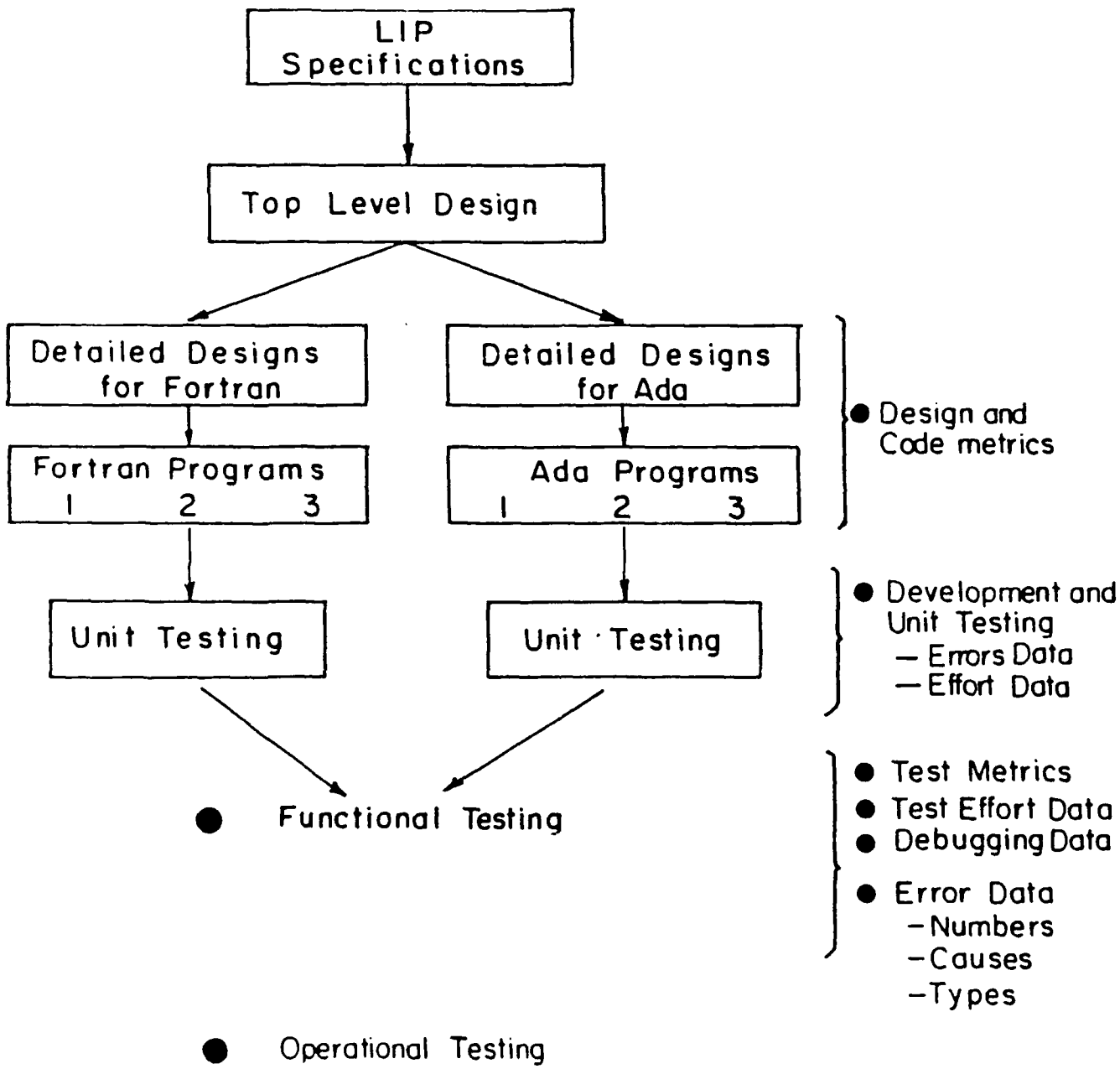

Figure 2.2. Experimental Plan for the Scientific Application

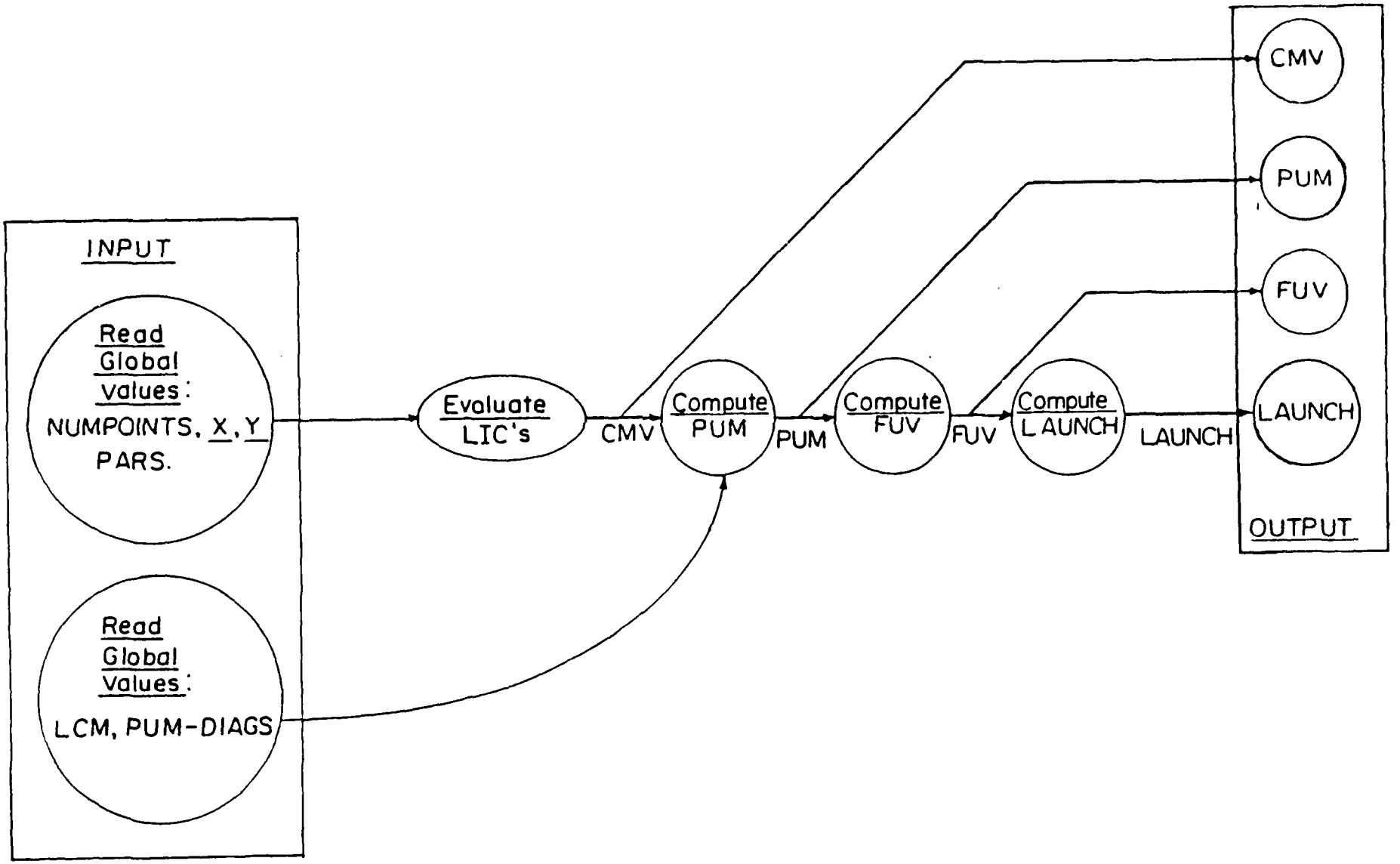

Figure 2.3. Schematic of Computational Steps in LIP Designs

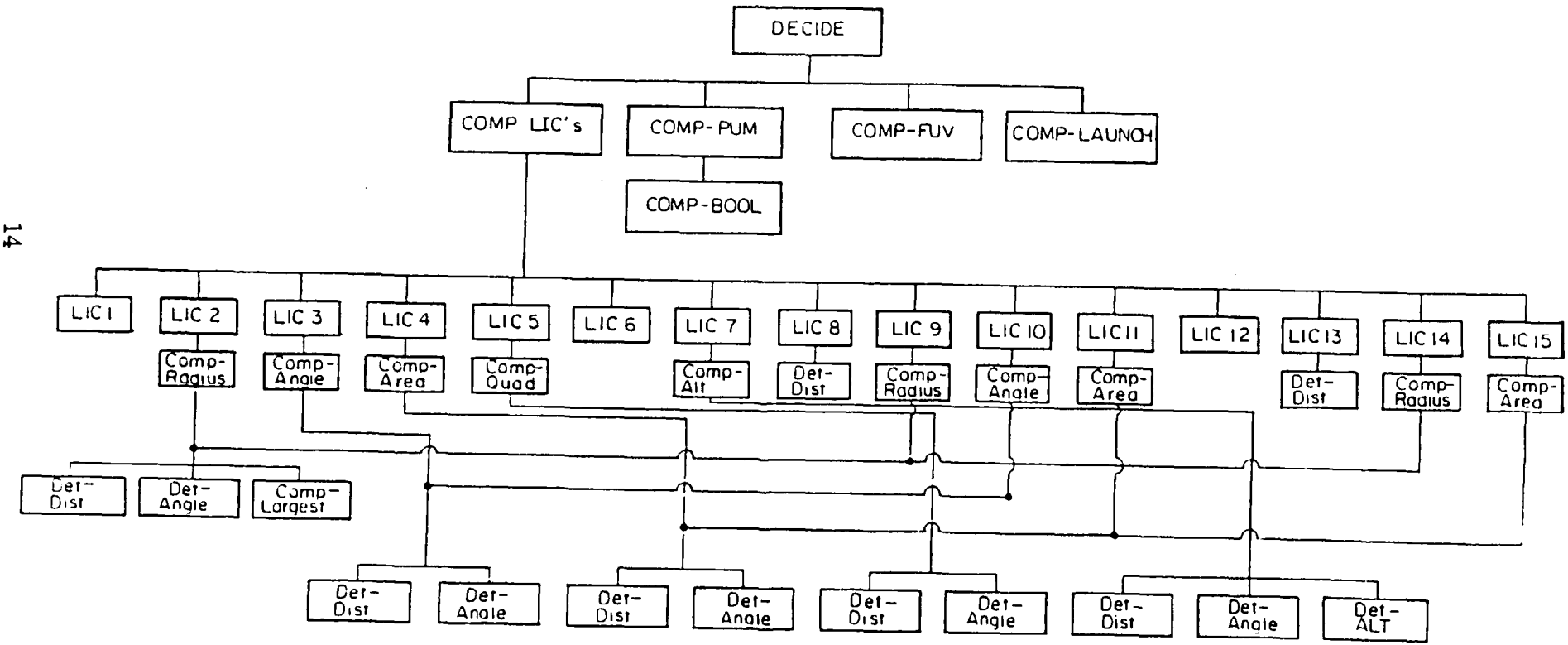

Figure 2.4. High Level Design for one LIP Program

detailed designs. Note that the metrics packages, procedures and tasks are applicable for the Ada programs only while subroutines are used only for the Fortran programs. Package is an Ada feature used for grouping of data structures and subprograms. A subprogram could be a procedure, function, or a task. A task in Ada is a procedure that can run concurrently with other tasks. Also, the number of subprograms includes functions, tasks and procedures; and a package includes the package body and specifications.

The code level size metrics given in Table 2.2 include source lines, lines with comments, non-comment source lines, and executable statements. Source lines is simply a count of all textual lines of code in the final version of a program. It includes compiler directives, documenting comments, data variable declarations, blank lines and executable statements. Comment and non-comment lines have the usual meanings and their sum equals the number of source lines. The number of executable statements is a count of the assignments and control statements used in each program.

Values of the control ststements IF, IF-THEN, ELSE, CASE, LOOP, WHILE LOOP, FOR LOOP, EXIT WHEN, DO LOOP and GO TO are counts of these programming constructs used in the programs. Finally, the values of exceptions and raise refer to the number of exceptions declared and the number of times they were used in the given program, respectively.

TABLE 2.2

Summary of Various Metrics for LIP Programs

Metric	FL1	FL2	FL3	AL1	AL2	AL3
Packages				6	4	3
Subprograms	21	19	28	30	23	29
Procedures				4	4	1
Subroutines	13	19	28			
Functions	8	0	0	11	4	28
Tasks				15	15	0
Source Lines	696	446	526	691	624	851
Comment Lines	146	24	87	59	126	251
Non-comment Lines	550	422	439	632	498	600
Executable Statements	212	246	174	214	184	137
IF	0	45	17			
IF THEN	68	11	31	50	43	52
IF ELSE	10	11	7	11	9	13
CASE				0	0	1
LOOP				1	0	0
WHILE Loop				18	21	0
FOR Loop				0	2	25
EXIT WHEN				0	0	0
DO LOOP	5	20	24			
GO TO	38	20	2	0	0	0
Exceptions				2	0	2
Raise				0	0	3

2.5.3 Unit Testing

Each programmer followed a somewhat different approach for unit testing and debugging of the LIP programs. Structure based testing was performed on many of the units of FL1 and FL2, function based testing was done on units of AL2 and AL3 and very little formal unit testing was done by programmers of FL3 and AL1. In addition to unit testing, some code reading was also performed on each program.

Several structure based testing criteria were used to generate unit level test cases for FL1 and FL2. As an example, consider Launch Interceptor Condition 1 (LIC1) for the Fortran program FL2. This LIC determines whether there exists at least one set of two consecutive points that are a distance greater than LENGTH1 apart. The directed flow graph of the unit to check this condition is shown in Figure 2.5. The following criteria were used to generate test cases for this unit: statement testing, branch testing, path testing, mutation testing, error-based testing, and domain testing.

2.5.4 Function Testing

The basic questions faced in testing the LIP programs, just as in testing any program, were the choice of test criterion, selection of test cases consistent with the chosen criterion and knowledge of the expected output values, i.e. of an oracle. Even though several criteria can be used for testing at the unit level testing, very few explicit choices are available for function level testing because most of the structure-based testing methods are useable primarily at the unit level and cannot be easily extended to higher level testing.

The level of difficulty in determining test cases and the value of the oracle is largely a function of the criterion used and the computational complexity of the program. In

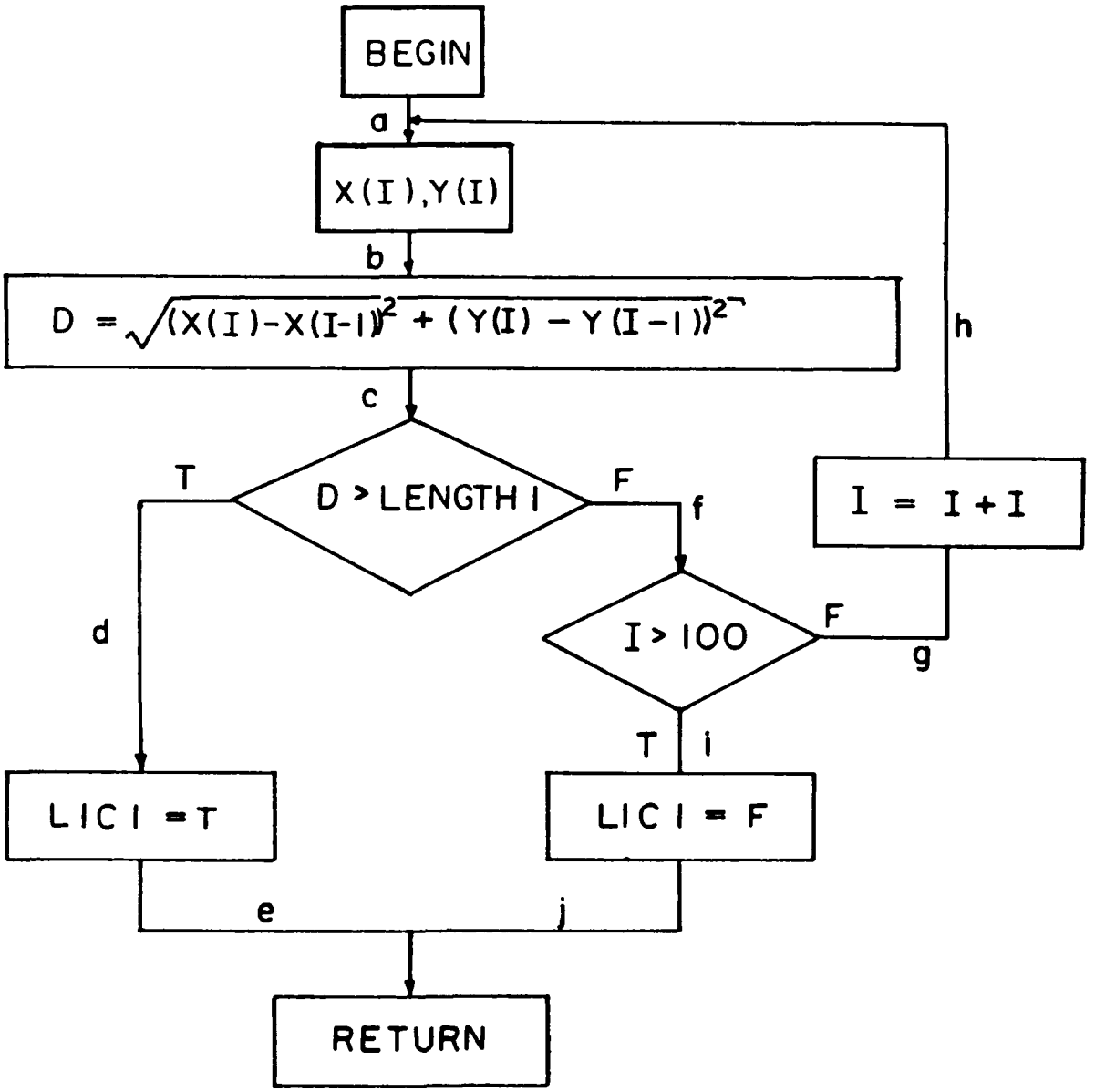

Figure 2.5. Directed Flow Graph of the Unit to check LIC1

general, given a test criterion, it may not be feasible to find test cases to satisfy that criterion. Further, even if the test cases can be found, it may not be possible to determine the oracle.

In case of the LIP programs, determining the oracle became an almost impossible task because of the large number of computations required to calculate the output values. For example, given 20 input pairs of (x,y) values and the other input parameters, it is not feasible to determine the correct values of LIC's, CMV etc. For similar reasons, it is not possible to determine the input values to ensure a given combination of output values for the PUM, FUV etc. It should be noted that such programs have been labelled as non-testable programs in [Weyuker 82]. In order to overcome these difficulties, we developed a testing methodology for LIP which took advantage of the nature of computational steps involved in this program as well as of the fact that multiple versions of the LIP were developed in this research. This methodology is described below.

We first note that the main computations here involve calculation of the launch interceptor conditions for a given set of input values. These conditions are then used to set up the Condition Met Vector. Therefore, for testing purposes we first concentrate on ensuring that LIC's are computed correctly. An implicit assumption here is that if the LIC's are correct, the other quantities such as FUV, Launch, etc., are likely to be correct due to the simpler nature of the calculations involved. Of course, these calculations also need to be tested for correctness but the testing need not be as thorough as for the LIC's.

Next, we note that the computations for the LIC's involve checking the specified geometric relationships among the input (x,y) pairs and that several of the LIC's are

based on similar calculations. Hence, similar LIC's can be grouped together. By doing so, testing can be concentrated on one group at a time. Since the number of LIC's in a group is small, we should be able to determine test cases for desired combinations of the LIC or CMV conditions in each group without much difficulty. It is this basic approach that we have employed in testing the various LIP programs. We first judiciously choose the true/false combinations of the LIC's to test as a group. This is equivalent to choosing the elements of CMV, i.e., choosing CMVE's. Next we determine the test cases by analyzing the detailed design of the program for chosen CMVE. If the computed CMVE's for these inputs match the expected values, we conclude that the computations of at least part of LIP are correct.

The above approach, however, does not tell us anything about the correctness of the other CMVE's until they too have been tested by the corresponding test cases. Theoretically, there is nothing wrong with implementing a testing approach based on the above methodology assuming that all LIC's or CMVE's will be tested by some test cases. We could, though, make it more efficient if the other conditions could also be checked for correctness since their computed values become available along with those of the chosen conditions.

To achieve this additional objective, we now make use of the multiple versions of the LIP programs available to us. While testing for a given set of CMVE's we compare the CMV's computed by each of the programs and accept the majority value of each element as the correct value. Thus, for each test case generated to check the correctness of the selected CMV elements, we can also determine whether the other elements are being

computed correctly.

To summarize, the above approach is a two step testing methodology. First we check the correctness of a specific, small set of CMVE's by comparing the computed and the correct values. Next, we check the other CMV elements against the majority values. We found this methodology to be not only practical but also very efficient in revealing errors.

2.5.5 CMVE Combinations and Test Cases

The set of CMVE's to test in a group is determined primarily from the LIP specifications. Factors that dictate this choice are the type of computations to be performed and interdependencies among the defining LIC's. Having decided upon the groups, the next step is to assign a true or false value to each element within a group. This assignment as well as the total number of combinations to test is a subjective decision. The choice is partially governed by the programmer's assessment of which CMV elements should be tested more than others. It should be noted that some true/false combinations of CMVE's are not feasible due to the definition of the CMV's and hence cannot be included in test groups. The test cases that would yield the chosen values of CMV elements can now be determined from an analysis of the detailed design. This is a slow process involving a trial and error approach.

For testing the LIP's developed in this experiment, we have developed two test sets. The first consists of 54 test cases based on program AL3, and the second has 120 test cases based on program FL1 as detailed below.

CMVE Combinations and Test Cases for Set Number 1

The grouping of CMVE's and the number of test cases for the first set is shown in

Table 2.3. As an example, a breakdown of the eleven true/false combinations chosen for group 1 is given in Table 2.4. A complete list of The test cases for this set is given in Appendix B.

It is useful to study the various combinations of CMV elements that are tested by the 54 test cases in set 1. A high degree of concentration of one or more combinations would indicate an unbalanced test set. Theoretically, the total number of distinct combination is 2^{15}. Some of these are not feasible due to interdependencies among the underlying LIC's and hence the actual upper bound will be smaller.

The actual combinations tested by these 54 test cases are listed in Table 2.5. We note that there are only eight redundant test cases; six of these are repeated twice and one is repeated three times. Thus, we see that this test set is quite efficient in terms of the coverage of CMV elements.

CMVE Combinations and Test Cases for Set Number 2

The second test set consists of 120 test cases which were developed based on program FL1. The grouping of CMVE's and the number of test cases in each group are given in Table 2.6. As an example, details of the twenty true/false combinations for group number 1 are shown in Table 2.7.

TABLE 2.3

CMVE Groups and number of Test Cases for Test Set 1

Group Number	CMVE's	No. of Cases
1	1,8,13	11
2	2,9,14	10
3	3,10	6
4	4,11,15	14
5	5	6
6	6	1
7	7	6
TOTAL		54

TABLE 2.4

Combinations of CMVE's 1, 8 and 13 Tested by Test cases 1 through 11 of Test Set 1

	Expected Values of CMVE's		
Test Case	(1)	(8)	(13)
1	T	T	F
2	F	T	F
3	F	F	F
4	T	T	F
5	T	T	T
6	F	F	F
7	T	T	F
8	T	F	F
9	T	T	T
10	T	T	F
11	F	F	F

TABLE 2.5

Combinations of CMV Elements Tested By

54 Test Cases of Test Set 1

>>>> CMV vector results for 54 tests <<<<

OUTNUM	CMV VECTOR															REPEATED
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	T	T	T	T	F	T	F	T	T	F	T	T	F	T	F	1
2	T	T	T	T	T	T	T	T	T	F	F	T	F	F	F	1
3	T	T	T	T	T	T	T	T	T	T	T	T	F	F	T	1
4	T	T	T	T	F	T	F	T	T	F	F	F	T	F	F	1
5	F	T	T	T	T	T	T	F	F	F	F	T	F	F	F	1
6	T	T	T	T	T	F	F	T	T	F	F	F	T	F	F	1
7	T	T	T	T	F	T	T	F	T	T	T	T	F	F	T	1
8	T	T	F	T	F	T	T	T	F	F	F	F	T	F	F	1
9	T	T	T	T	T	T	T	F	F	F	F	T	F	F	F	1
10	T	T	T	T	T	T	F	F	T	T	T	T	F	F	F	1
11	F	T	T	T	F	T	F	F	T	T	T	T	F	F	T	1
12	T	T	T	F	T	T	T	T	F	F	F	T	F	F	F	1
13	F	T	T	F	T	T	T	F	T	T	T	T	F	F	F	1
14	T	T	T	F	T	T	F	T	F	T	T	T	F	F	T	1
15	T	T	T	T	F	T	F	F	T	F	T	T	F	F	F	1
16	T	T	T	T	T	T	T	T	T	T	T	T	T	F	T	1
17	T	T	T	F	T	T	F	F	T	F	F	T	F	F	F	1
18	T	T	T	T	T	T	T	T	T	F	T	T	F	T	F	1
19	T	T	T	T	F	T	T	T	T	F	T	T	F	F	T	1
20	T	T	T	T	T	T	T	T	T	T	F	T	F	F	F	1
21	T	T	T	F	T	T	T	T	T	T	F	T	T	F	F	1
22	T	T	T	T	F	T	F	T	T	T	T	T	F	F	T	2
23	T	T	T	T	T	T	F	F	F	T	F	T	F	F	F	1
24	T	T	T	T	F	T	T	T	F	F	F	T	T	F	F	1
25	T	T	F	T	T	T	T	F	F	F	F	T	F	F	F	1
26	T	T	F	T	F	T	T	T	T	F	T	T	F	F	F	1
27	T	T	T	T	T	T	T	T	F	T	T	F	F	F	T	1
28	T	T	T	T	F	T	T	T	T	T	T	T	T	T	F	1
29	T	T	T	T	F	T	T	T	T	T	T	T	F	T	F	1
30	T	T	T	T	T	T	F	T	F	T	T	T	T	F	T	1
31	T	T	T	T	F	T	F	T	T	F	F	T	F	T	F	1
32	T	T	T	T	F	T	F	T	T	F	F	T	F	F	F	1
33	T	T	T	T	T	T	F	T	T	F	F	T	F	T	F	1
34	T	T	T	T	F	T	T	T	T	T	T	T	T	F	T	2
35	T	T	T	T	T	T	T	T	T	T	T	T	T	F	F	1
36	F	T	T	T	T	T	F	F	F	T	F	T	F	F	F	1
37	T	T	T	T	T	T	T	T	F	T	F	T	F	F	F	1
38	T	T	T	T	T	T	T	T	F	T	F	T	T	F	F	2
39	T	T	T	T	T	T	F	F	F	F	F	T	F	F	F	2
40	T	T	T	T	T	T	T	T	T	T	T	T	F	F	F	2
41	F	T	T	T	T	T	T	F	T	T	T	T	F	T	F	1
42	T	T	T	T	T	T	F	T	T	T	T	T	T	F	F	3
43	T	T	T	T	F	T	T	T	T	T	T	T	F	F	F	2
44	F	T	T	T	T	T	T	F	T	F	T	T	F	F	F	1
45	F	T	T	T	T	T	T	F	T	F	T	F	F	F	T	1
46	T	F	T	T	T	T	T	T	F	F	F	F	F	F	F	1

Number of redundant tests is : 8

TABLE 2.6

CMVE Groups and number of Test Cases for Test Set 2

Group Number	CMVE's	No.of Cases
1	2,9,14	20
2	1,8,13	26
3	3,10	19
4	4,6,11,15	24
5	6	1
6	6,7	16
7	5	14
TOTAL		120

TABLE 2.7

Combinations of CMVE's 2, 9 and 14 Tested by Test cases 1 through 20 of Test Set 2

	Expected Values of CMVE's		
Test Case	(2)	(9)	(14)
1	T	F	F
2	T	T	T
3	T	T	F
4	F	F	F
5	F	F	F
6	T	F	F
7	T	T	T
8	T	T	T
9	T	F	F
10	T	T	F
11	T	T	F
12	T	T	F
13	T	T	T
14	T	T	F
15	T	F	F
16	T	T	F
17	F	F	F
18	F	F	F
19	T	T	T
20	T	T	F

3. DESCRIPTION OF DATA

In this section we present the data collected during various phases of development and testing of the Launch Interceptor Programs. Since the main objective of the study was reliability comparison, most of the data is about errors detected during testing of the programs. Before presenting the error data, however, we give the details of test results. The output of the set of 54 test cases and the number of CMV elements computed incorrectly by each program, are given in Section 3.1. The structural coverage of the programs provided by this test set is discussed in Section 3.2

The errors found in each program during development and unit testing and during development and unit testing and during function testing by 54 test cases are given in Sections 3.2 and 3.4 respectively. Each of these sets of errors are further categorized into error symptoms, phases during which error was created, error causes and error types. Results of error error data from 120 test cases of set 2 and random testing to simulate the operational environment are described in Section 3.5. Data on the effort expended during program development, testing and debugging are given in Section 3.6. Finally, some relevant data about the two text formatting programs is described in Section 3.7

3.1 Results of Test Run

The output of the test run from the 54 test cases of set 1 is shown in Table 3.1. In this table, there are two rows of entries for each test case. The first row lists the incorrect CMV elements for a program. The three entries in the second row give respectively the number of incorrect elements for that test case, the cumulative number of incorrect elements for all the test cases so far, and the cumulative number of distinct incorrect elements for all

TABLE 3.1

ERROR DETECTION FORM

TC #	FL1	FL2	AL1	AL2	AL3
1	6F 10T 13T	3F 11T 12T 13T 15T	3F	5F 7F	10T
	3 3 3	5 5 5	1 1 1	2 2 2	1 1 1
2	6F 8T 12F 13T	3F 8T 9F 13T 15T	14T	5F 7F 15T	
	4 7 5	5 10 7	1 2 2	3 5 3	0 1 1
3	2F 6F 10T 12F	7F 8T 9F 13T 15T	9F 11F	3T 5F 7F	3T 10T
	4 11 6	5 15 8	2 4 4	3 8 4	2 3 2
4	6F 12F	3F 5T 9F 10F 13T 15T	10F 11F 14T	7F	
	2 13 6	6 21 10	3 7 5	1 9 4	0 3 2
5	6F 12F	3F 7T 9F 10F 15T	10F 14T	5F	
	2 15 6	5 26 11	2 9 5	1 10 4	0 3 2
6	6F 12F	3F 8T 9F 10F 13T 14F 15T	10F 15T	5F 7F	
	2 17 6	7 33 12	2 11 6	2 12 4	0 3 2
7	6F 12F	3F 9F 10F 13T 15T	10F	5F 7F	
	2 19 6	5 38 12	1 12 6	2 14 4	0 3 2
8	2F 6F 12F	3F 7T 8T 11T 13T 15T	2F 3F	5F	
	3 22 6	6 44 12	2 14 7	1 15 4	0 3 2
9	6F 12F	3F 10F 11T 15T	2F 3F 7F	5F 7F	
	2 24 6	4 48 12	3 17 8	2 17 4	0 3 2
10	2F 6F 12F	10F 11T 13T 15T	2F 3F 7F 10F	5F 7F	
	3 27 6	4 52 12	4 21 8	2 19 4	0 3 2
11	6F 12F	3F 7T 8T 10F 11T 13T 15T	3F	5F	
	2 29 6	7 59 12	1 22 8	1 20 4	0 3 2
12	6F 12F 13F	2F 3F 10F 15T	10F	5F 7F 9T	9T
	3 32 7	4 63 13	1 23 8	3 23 5	1 4 3
13	6F 12F 14T 15F	5T 8F 9F 10F	7F 10F 14T	7F	
	4 36 9	4 67 14	3 26 8	1 24 5	0 1 3
14	6F 12F	5T 8F 10F 13T 15T	7F 10F	7F	2T 9T
	2 38 9	5 72 14	2 28 8	1 25 5	2 6 4
15	6F 12F	3F 5F 7T 9F 11T 13T 14F 15T	2F 9F 14F	5F	
	2 40 9	8 80 15	3 31 9	1 26 5	0 6 4
16	2F 6F 12F	3F 5T 7T 9F 11T 13T 15T	2F 9F		
	3 43 9	7 87 15	2 33 9	0 26 5	0 6 4
17	2F 6F 12F	3F 5T 7T 9F 11T 13T 14F 15T	2F 9F 14F		
	3 46 9	8 95 15	3 36 9	0 26 5	0 6 1
18	6F 7T 12F 13F 15F	2F 7T 10F	3F 10F	5F	
	5 51 10	3 98 15	2 38 9	1 27 5	0 6 4

TABLE 3.1 (continued)

TC #	FL1	FL2	AL1	AL2	AL3
19	6F 12F 13T	3F 5T 9F 10F 13T 14F 15T	9F 10F 14F	7F	
	3 54 10	7 105 15	3 41 9	1 28 5	0 6 4
20	6F 12F 13F	3F 10F 15T	11F	5F 7F 11F	2T
	3 57 10	3 108 15	1 42 9	3 31 6	1 7 4
21	6F 12F 13F	3F 5T 10F 15T		7F 9T 14T	9T 14T
	3 60 10	4 112 15	0 42 9	3 34 7	2 9 5
22	6F 15F	2F 3F 10F 12T 13T	10F 15F	5F 7F	
	2 62 10	5 117 15	2 44 10	2 36 7	0 9 5
23	6F 10T 12F	3T 5T 8F 9F 13T 14F 15T	7F 11F	7F	3T 10T 14F
	3 65 10	7 124 16	2 46 10	1 37 7	3 12 6
24	6F 8T 12F	3T 8T 11T 13T 15T		5F 7F	
	3 68 10	5 129 16	0 46 10	2 39 7	0 12 6
25	3F 6F 12F	5T 11T 15T	3F	2T 7F	2T
	3 71 11	3 132 16	1 47 10	2 41 8	1 13 6
26	3F 6F 10F 12F	3F 10F 11T 13T 15T		2T 5F	2T
	4 75 12	5 137 16	0 47 10	2 43 8	1 14 6
27	6F 12F 13T 15F	3F 5T 7T 8F 13T	15F	10T	2T 9T 10T
	4 79 12	5 142 16	1 48 10	1 44 9	3 17 6
28	6F 12F 14T 15F	3F 8F 9F	7F 14T	7F 10T	10T
	4 83 12	3 145 16	2 50 10	2 46 9	1 18 6
29	6F 12F 13T	3F 9F 13T 15T		5F 7F 10T	10T
	3 86 12	4 149 16	0 50 10	3 49 9	1 19 6
30	4T 6F 11T 12F	3F 4T 5F 8F 9F 11T 15T	7F 14T	5F 7F 10T	10T
	4 90 14	7 156 17	2 52 10	3 52 9	1 20 6
31	6F 12F	3F 5F 9F 13T 15T	11F	5F 7F 10T 11F	10T
	2 92 14	5 161 17	1 53 10	4 56 9	1 21 6
32	6F 7T 12F	3F 7T 13T 15T	10T 11F	5F 11F	10T
	3 95 14	4 165 17	2 55 11	2 58 9	1 22 6
33	6F 12F 13T 14T 15F	3F 5T 9F 13T	9F 11F 15F	7F 14T	
	5 100 14	4 169 17	3 58 11	2 60 9	0 22 6
34	6F 12F	3F 9F 13T 14F 15T	9F 11F 14F	5F 7F	
	2 102 14	5 174 17	3 61 11	2 62 9	0 22 6
35	4T 6F 7T 11T 12F	3F 4T 5F 7T 9F 11T 13T 15T	9F	5F	
	5 107 14	8 182 17	1 62 11	1 63 9	0 22 6
36	6F 12F 15F	3F 5F 8F 9F	2F 9F	5F 7F 10T	10T 11F 15F
	3 110 14	4 186 17	2 64 11	3 66 9	3 25 8

TABLE 3.1 (continued)

TC #	FL1	FL2	AL1	AL2	AL3
37	6F 8T 12F 13T	3F 9F 13T 15T	15T		11F
	4 114 14	4 190 17	1 65 11	0 66 9	1 26 8
38	4T 6F 7T 12F	2T 3F 4T 5F 7T 8F 13T 15T	11F	5F 10T 15T	2T 10T 11F
	4 118 14	8 198 18	1 66 11	3 69 9	3 29 8
39	4T 6F 12F	3F 4T 5F 9F 10F 13T 14F 15T	11F	5F 7F	11F 14F
	3 121 14	8 206 18	1 67 11	2 71 9	2 31 8
40	4T 6F 11T 12F	3F 4T 5F 11T 13T 15T	3F	5F 7F	
	4 125 14	6 212 18	1 68 11	2 73 9	0 31 8
41	6F 12F 15F	3F 7T 9F 10F 13T			11F 15F
	3 128 14	5 217 18	0 68 11	0 73 9	2 33 8
42	6F 7T 12F	3F 10F 13T 15T	2F 10F 11F	5F	9T 15T
	3 131 14	4 221 18	3 71 11	1 74 9	2 35 9
43	6F 12F	3F 8F 9F 10F 15T	9F	5F	15T
	2 133 14	5 226 18	1 72 11	1 75 9	1 36 9
44	6F 12F	3F 7F 8T 11T 13T 15T		5F 7F	
	2 135 14	6 232 18	0 72 11	2 77 9	0 36 9
45	2F 6F 8T 13T	5T 7F 8T 11T 12T 13T 15T		7F	
	1 139 14	7 239 18	0 72 11	1 78 9	0 36 9
46	6F 8F 12F 13F	3F 8F 9F 10F 15T	10F 11F	5F	15T
	4 143 15	5 244 18	2 74 11	1 79 9	1 37 9
47	6F 7T 12F 15F	5T 10F 13T	10F 15F	2T	2T 7T 9T
	4 147 15	3 247 18	2 76 11	1 80 9	3 40 10
48	10T	3F 6T 7T 9F 11T 12T 15T	3F 6T 14T	5F	10T
	1 148 15	7 254 19	3 79 12	1 81 9	1 41 10
49	6F 12F	1T 3F 8T 11T 13T 15T		5F 7F	
	2 150 15	6 260 20	0 79 12	2 83 9	0 41 10
50	6F	3F 5T 7T 9F 11T 12T 15T	3F 14T		
	1 151 15	7 267 20	2 81 12	0 83 9	0 41 10
51	6F 12F 13T 15F	3F 8F 9F 10F 13T	3F 7F 10F	5F 7F	
	4 155 15	5 272 20	3 84 12	2 85 9	0 41 10
52	6F 12F 13T 15F	2T 5T 7T 8F 10F 13T	3F 10F 11F 15F		2T 9T
	4 159 15	6 278 20	4 88 12	0 85 9	2 43 10
53	6F 12F	3F 8F 9F 11T 13T 15T	14T	5F 7F	
	2 161 15	6 284 20	1 89 12	2 87 9	0 43 10
54	6F 12F	3F 5T 7T 8F 9F 13T 14F 15T	9F 11F 14F		
	2 163 15	8 292 20	3 92 12	0 87 9	0 43 10

the test cases upto and including the current one. For example, for test case number 2 and program FL1, there are four incorrect computations, viz, 6F (false), 8T (true), 12F (false) and 13T (true) while the correct values are 6T, 8F, 12T and 13F, respectively. The total number of incorrect CMV elements in test cases 1 and 2 is 7 (the above four and 6F, 10T and 13T in test case #1). The total number of distinct incorrect values upto and including test case 2 is five (6F, 10T, 13T, 8T, and 12F).

The total number of distinct incorrect CMV computations is shown in the last row of Table 3.1. This number is 15 for FL1, 20 for FL2, 12 for AL1, 9 for AL2, and 10 for AL3. A summary of the conditions computed by each of the programs is given in Table 3.2. The entries in this Table give the total number of CMV values computed incorrectly by each program. For example, for FL1 CMV (2) is incorrectly computed as false for a total of six times, CMV (13) is incorrectly computed as true 10 times and as false 5 times.

The information in Table 3.1 and 3.2 is very useful in assessing the correctness of the programs as well as in debugging as testing progresses.

3.2 Structural Coverage of the Programs

One measure of test data adequacy is the degree of structural coverage of the programs. In this experiment such coverage of the Fortran and Ada programs was monitored with every test case. As an example, consider the Ada program AL3. Various relevent metrics for this program are listed in Table 3.3. The cumulative numbers and cumulative percentages of several structural metrics of this program were recorded as a function of test case number.

The metrics monitored were subprograms, branches, statements (executions), in-

TABLE 3.2

Number of CMV Elements Computed Incorrectly for each Program

Program FL1			
CMV Number	TRUE	FALSE	TOTAL
CMV: 2	0	6	6
CMV: 3	0	2	2
CMV: 4	5	0	5
CMV: 6	0	53	53
CMV: 7	6	0	6
CMV: 8	4	1	5
CMV:10	4	1	5
CMV:11	3	0	3
CMV:12	0	49	49
CMV:13	10	5	15
CMV:14	3	0	3
CMV:15	0	11	11
TOTAL	35	128	163

Program FL2			
CMV Number	TRUE	FALSE	TOTAL
CMV: 1	1	0	1
CMV: 2	2	3	5
CMV: 3	2	43	45
CMV: 4	5	0	5
CMV: 5	16	8	24
CMV: 6	1	0	1
CMV: 7	16	3	19
CMV: 8	9	14	23
CMV: 9	0	30	30
CMV:10	0	24	24
CMV:11	20	0	20
CMV:12	5	0	5
CMV:13	39	0	39
CMV:14	0	8	8
CMV:15	43	0	43
TOTAL	159	133	292

Table 3.2 (continued)

Program AL1			
CMV Number	TRUE	FALSE	TOTAL
CMV: 2	0	8	8
CMV: 3	0	12	12
CMV: 6	1	0	1
CMV: 7	0	8	8
CMV: 9	0	11	11
CMV:10	1	16	17
CMV:11	0	14	14
CMV:14	9	5	14
CMV:15	2	5	7
TOTAL	13	79	92

Program AL2			
CMV Number	TRUE	FALSE	TOTAL
CMV: 2	3	0	3
CMV: 3	1	0	1
CMV: 5	0	35	35
CMV: 7	0	32	32
CMV: 9	2	0	2
CMV:10	7	0	7
CMV:11	0	3	3
CMV:14	2	0	2
CMV:15	2	0	2
TOTAL	17	70	87

Program AL3			
CMV Number	TRUE	FALSE	TOTAL
CMV: 2	8	0	8
CMV: 3	2	0	2
CMV: 7	1	0	1
CMV: 9	7	0	7
CMV:10	12	0	12
CMV:11	0	5	5
CMV:14	1	2	3
CMV:15	3	2	5
TOTAL	34	9	43

TABLE 3.3

Several Metrics Program AL3

The total number of lines is:	808
The total number of lines with comments is:	245
The total number of lines without comments is:	563
The total number of blocks for the program being tested is:	31
The total number of branches is:	105
The total number of statements is:	136
The total number of input/output:	0
The total number of IF THEN:	50
The total number of IF ELSE:	12
The total number of CASE:	18
The total number of LOOP:	0
The total number of WHILE:	0
The total number of FOR:	25
The total number of EXIT WHEN:	0

put/output statements, IF THEN statements,IF ELSE statements, CASE statements, LOOPS, WHILE statements, FOR statements and EXIT statements.

The results ot the structural coverage obtained by the 54 test cases for program AL3 are summarized in Table 3.4. The entries in this table represent actual coverage values and percentages of the totals in the program. For example, test case number 1 executes 31 subprograms (100% of the total number of subprograms), 70 branches (66% of the total) and 89 executable ststements (65% of total). Other entries in the table can be similarly interpreted.

It is interesting to note that a very high level of structural coverage is obtained by a very few test cases. The reason for this lies in the fact that computations of the LIP programs involve most of the program and hence a small number of test cases can provide high coverage.

3.3 Error Data During Development and Unit Testing

In the development and unit testing phase, errors were found via compilation attempts, code-reading and unit test cases. The data collected for each program consists of three sets of values, viz, number of compilation attempts, number of syntax errors and number of other errors. These values for the six programs are given in Table 3.5.

Errors listed as other errors were further classified into several categories using a scheme given in [Selby 85]. These are error symptoms, phases during which errors were created, error causes, and error types. A breakdown of the other errors in Table 3.5 into these four categories is shown in Table 3.6.

3.4 Error Data During Function Testing

TABLE 3.4

(Ada Program AL3 Test Set 1)

CUMULATIVE COVERAGE OF EXECUTED CODE

TC#	SUB, P	BRA, P	STA, P	IOT, P	IFT, P	IFE, P	CAS, P	LOP, P	WHI, P	FOR, P	EXI, P
1	31,100	70, 66	89, 65	0, 0	25, 50	7, 58	18,100	0, 0	0, 0	20, 80	0, 0
2	31,100	83, 79	107, 78	0, 0	33, 66	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
3	31,100	83, 79	107, 78	0, 0	33, 66	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
4	31,100	83, 79	109, 80	0, 0	33, 66	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
5	31,100	84, 80	115, 84	0, 0	34, 68	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
6	31,100	85, 80	115, 84	0, 0	35, 70	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
7	31,100	85, 80	115, 84	0, 0	35, 70	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
8	31,100	85, 80	115, 84	0, 0	35, 70	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
9	31,100	85, 80	116, 85	0, 0	35, 70	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
10	31,100	85, 80	116, 85	0, 0	35, 70	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
11	31,100	85, 80	116, 85	0, 0	35, 70	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
12	31,100	85, 80	116, 85	0, 0	35, 70	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
13	31,100	86, 81	117, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
14	31,100	86, 81	118, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
15	31,100	86, 81	118, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
16	31,100	86, 81	118, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
17	31,100	86, 81	118, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
18	31,100	86, 81	118, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
19	31,100	86, 81	118, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
20	31,100	86, 81	118, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
21	31,100	86, 81	118, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
22	31,100	86, 81	118, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
23	31,100	86, 81	118, 86	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
24	31,100	86, 81	121, 88	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
25	31,100	86, 81	121, 88	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
26	31,100	86, 81	121, 88	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	21, 96	0, 0
27	31,100	86, 81	121, 88	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	21, 96	0, 0

TABLE 3.4 (continued)

TC#	SUB, P	BRA, P	STA, P	IOT, P	IFT, P	IFE, P	CAS, P	LOP, P	WHI, P	FOR, P	EXI, P
28	31,100	86, 81	121, 88	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
29	31,100	86, 81	121, 88	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
30	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
31	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
32	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
33	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
34	31,100	86, 81	122, 89	0, 0	36. 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
35	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
36	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
37	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
38	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
39	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
40	31,100	86, 81	122, 89	0. 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0. 0
41	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
42	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
43	31,100	86, 81	122, 89	0, 0	36, 72	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
44	31,100	89, 84	122, 89	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
45	31,100	89, 84	122, 89	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
46	31,100	89, 84	122, 89	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
47	31,100	89, 84	122, 89	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
48	31,100	89, 84	123, 90	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
49	31,100	89, 84	123, 90	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
50	31,100	89, 84	123, 90	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
51	31,100	89, 84	123, 90	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
52	31,100	89, 84	123, 90	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
53	31,100	89, 84	123, 90	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0
54	31,100	89, 84	123, 90	0, 0	39, 78	8, 66	18,100	0, 0	0, 0	24, 96	0, 0

SUB --> Subprograms
BRA --> Branches
STA --> Statements
IOT --> Input/Output
IFT --> IF THEN
IFE --> IF ELSE
CAS --> CASE

LOP --> LOOP
WHI --> WHILE
FOR --> FOR
EXI --> EXIT
P --> Percent

TABLE 3.5

Error Data During Compilation, Code Reading & Unit Testing

Error Data						
Category	FL1	FL2	FL3	AL1	AL2	AL3
Number of compilations	33	39	32	20	35	31
Syntax Errors	20	27	17	30	25	18
Other Errors	24	28	10	8	7	4
TOTAL Errors	44	55	27	38	32	22

TABLE 3.6

Breakdown of errors into various catregories

Development and Unit Testing							
Category	Sub-category	FL1	FL2	FL3	AL1	AL2	AL3
Error Symptoms	Overflow/Underflow	5	3	2	0	0	2
	Access Violation	0	0	0	0	0	0
	Infinite Loop	0	0	0	3	0	0
	Wrong Result	19	23	8	5	7	0
	Constraint Error	0	2	0	0	0	2
Step During which Error Created	Spec. Reading	0	0	0	0	0	0
	Design	2	13	4	4	0	0
	Coding	22	15	6	4	7	4
Error Causes	Programmer Error	11	5	3	0	3	4
	Previous Fix	3	5	0	0	0	0
	Misunderstanding Language	3	1	0	0	0	0
	Communication Failure	0	0	0	0	0	0
	Incorrect Specification Implementation	6	8	2	4	2	0
	Clerical	1	5	2	4	2	0
	Spec. Unclear	0	4	3	0	0	0
Error Types	Initialization	0	0	0	1	0	0
	Computation	1	9	3	4	4	0
	Control	5	10	3	0	1	1
	Interface	8	3	2	0	0	1
	Data	10	6	2	3	2	2

Next we present the results of testing the LIP's using 54 test cases of the first test set. As mentioned earlier, correctness of a computed CMV element was determined by checking it against the expected value whenever possible. For other elements, majority of the five computations was taken to be correct. Recall that the sixth program (FL3) was tested after the other five had been corrected for errors revealed during testing.

Each program was first executed for the 54 test cases with an output as shown in Table 3.1. The errors causing the incorrect symptoms due to test case 1 were then isolated. The programs were debugged for these errors and run for the remaining 53 test cases. Errors causing incorrect symptoms for test case 2 were removed next and the programs run for the next 52 test cases. This process was continued until no incorrect symptoms were observed. The number of errors found by the various test cases were recorded as shown in Table 3.7. We note that the total number of errors found by the 54 test cases are 16, 18, 15, 5, 4 and 4 for programs FL1, FL2, FL3, AL1, AL2 and AL3 respectively. It is interesting to note that all the errors in Ada programs were found by 13 test cases while it took 37 test cases to find all the errors in the Fortran programs. These numbers will of course be different if the same 54 test cases were run in a different order than used here.

The above errors were further examined and categorized according to the classification used in Section 3.3. The breakdown into these categories and subcategories is given in Table 3.8. Further analysis of this data will be discussed in Section 4.

3.5 Results from Test Set 2 and Random Test Cases

A total of 120 test cases from test set 2 were run on the five LIP's and the results recorded in the same way as for test set 1. However, we found it unnecessary to remove

TABLE 3.7

Number of Errors found by Various Test Cases

Number of errors						
Test Case	FL1	FL2	FL3	AL1	AL2	AL3
1	4	5	5	1	3	1
2	1	2	2	1	1	
3	4			1		1
4		3	2			
9		2	2	2		
13	1	2				2
15	1	2				
18	3	1	3			
20	1		1			
25	1		1			
37		1				
TOTAL	16	18	15	5	4	4

TABLE 3.8

Breakdown of errors found during function testing into various categories

Function testing by Test Set 1							
Category	Sub-category	FL1	FL2	FL3	AL1	AL2	AL3
Error Symptoms	Overflow/Underflow	1	4	0	0	1	0
	Access Violation	0	0	0	0	0	0
	Infinite Loop	0	0	0	0	1	0
	Wrong Result	15	14	15	5	2	4
	Constraint Error	0	0	0	0	0	0
Step During which Error Created	Spec. Reading	0	0	0	0	0	0
	Design	5	7	4	3	3	3
	Coding	11	11	9	2	1	1
Error Causes	Programmer Error	4	10	9	0	1	0
	Previous Fix	2	0	2	0	0	0
	Misunderstanding Language	1	0	0	0	0	0
	Communication Failure	0	0	0	0	0	0
	Implementation of Specifications	6	4	2	2	3	3
	Clerical	3	4	2	2	0	1
	Spec. Unclear	0	0	0	1	0	0
Error Types	Initialization	0	0	0	0	0	0
	Computation	4	6	5	2	1	4
	Control	6	4	4	0	2	0
	Interface	5	1	1	0	0	0
	Data	1	7	5	3	1	0

errors from the programs since most of the incorrect CMV values were caused by the same errors that had already been detected by 54 test cases of set 1. In order to get maximum information about the error content of the programs, we decided to run 120 test cases on the five LIP versions that had been corrected for all errors detected by test set 1. This run revealed 1 new error each in FL1 and FL2. No additional errors were found in the Ada programs.

Two sets of random test cases were run on the five LIP versions after removing all errors indicated by 54 test cases of set 1. The objective was to simulate the operational environment for these programs and study their reliability.

The first set consisted of 100 random test cases. These were generated by assuming a uniform distribution of the number of points over the range 2 to 100. Various other values of the number of intermediate and number of consecutive points were generated by assuming a uniform distribution over the corresponding range. For example, suppose the generated value of the number of points for a test case is 10. Then the value of N-PTS was chosen from a uniform distribution over (3-10).

The values of (x,y) coordinates and other input variables, such as LENGTH1, were generated from uniform distributions on the range (0.0-100.0).

The set of 1000 random test cases was generated from uniform distributions similar to those for 100 random test cases, except that the ranges of (x,y) and variables such as LENGTH1 were taken to be (0.0-10.0).

Results of executing the six LIP's for the 100 and the 1000 random test cases are given in Table 3.9.

TABLE 3.9

Number of Errors Detected by Random Test Cases

Number of Incorrect Unique CMV Elements						
Test Set	FL1	FL2	FL3	AL1	AL2	AL3
100 Cases	2	0	1	1	0	0
1000 Cases	2	0	1	0	0	0
TOTAL	4	0	2	1	0	0

TABLE 3.10

Time spent during development, Unit Testing and Debugging

Time spent, hours						
Activity	FL1	FL2	FL3	AL1	AL2	AL3
Specification Reading	5	6	6	10	10	6
Design	15	10	12	16	14	19
Coding	27	25	22	20	18	16
Subtotal	47	41	40	46	42	41
Unit testing and debugging	41	31	19	10	16	18
Unit Test Case Generation	45	53	25	20	15	30
Subtotal	86	84	44	30	31	48
TOTAL	133	125	84	76	73	89

3.6 Effort Data During Program Development

Data on the amount of effort expended during various phases of LIP development were collected for each program version and are summarized in Table 3.10.

In each case, the figure for specification reading is the total time spent in reading and re-reading the specification document. The effort reported for design activity includes the development of high-level and detailed designs. Entries in this category include the time for the development of algorithms for various geometric functions but exclude the time spent in reviewing the basic formulae for such calculations.

Coding time is the number of hours spent in writing the code from a detailed design. It does not include the time required to review or learn the features of Fortran or Ada languages. However, editing time is included in this category.

Unit testing and debugging time encompasses compilation time and debugging of syntax errors, execution of unit test cases and the time taken to remove the errors exposed by these test cases. Time for occasional code- reading and removal of errors found during code-reading is also included. Note that the times for FL1 and FL2 are substantially larger than for the other versions. This is so because structure-based testing was done for these two programs while the other versions were tested using function-based tests.

The time spent in generating unit cases for each program is also given in Table 3.10. Again we note that the time taken for FL1 and FL2 is substantially higher than for the other programs.

Testing and Debugging Effort: Test Set 1

Next, we present data on the amount of effort required to isolate and remove the

errors detected by the 54 test cases of test set 1 The time required to develop these test cases was 105 hours.

The total effort requirted for each program is given in Table 3.11. Breakdown of the effort for error isolation and removal is given in Table 3.12 and 3.13 respectively. As seen from these tables, most of the errors were detected and fixed in less than one hour each. In general, error detection times are much longer than the corresponding error removal times.

3.7 Data on Text Formatter

Two versions of the text formatter problem were implemented, one each in Fortran and Ada. Some relevant metrics about the two programs are summarized in Table 3.14. We note that these are fairly small programs and make use of the usual Fortran and Ada features.

Both of these programs were tested using black-box or structure independent testing. The number of errors found during unit and function testing in the two programs are given in Table 3.15. Since the number of errors is very small, further analyses of this data are not likely to provide additional useful information about the relative effects of Fortran and Ada on program reliability and hence were not pursued.

TABLE 3.11

Testing and Debugging Effort

Testing and Debugging effort						
Effort(hours)	FL1	FL2	FL3	AL1	AL2	AL3
Error Isolation	35	22	18	8	5	5
Error Removal	11	12	8	3	2	2
TOTAL	46	34	26	11	7	7

TABLE 3.12

Distribution of Time to Isolate Errors

Detected by Test Set 1

Number of Errors in LIP						
Duration*	FL1	FL2	FL3	AL1	AL2	AL3
0–1	10	12	9	4	2	2
1–4	3	5	4	1	2	2
4–8	2	1	2	0	0	0
8+	1	0	0	0	0	0

* (in hours)

TABLE 3.13

Distribution of Time to Fix Errors

Detected by Test Set 1

Number of Errors in LIP						
Duration*	FL1	FL2	FL3	AL1	AL2	AL3
0–1	14	16	14	5	4	4
1–4	2	2	1	0	0	0
4–8	0	0	0	0	0	0
8+	0	0	0	0	0	0

* (in hours)

TABLE 3.14

Some Metrics of Text Formatter Programs

Source Lines	70	126
Lines with comments	22	10
Lines w/o comments	48	116
Packages	*	0
Subprograms	1	6
Subroutines	0	
Procedures	1	6
Functions	0	0
Tasks		0
Executable Statements	33	40
IF	1	0
IF THEN	3	13
IF ELSE	0	2
CASE		0
LOOP		0
WHILE Loop		0
FOR Loop		5
EXIT WHEN		0
Exceptions		0
Raise		0
Goto	1	0
Do Loop	6	

TABLE 3.15

Error Data on Text Formatted Programs

Category	Fortran	Ada
Unit Testing	5	3
Function Testing	5	3
TOTAL	10	6

4. DATA ANALYSES AND RELIABILITY COMPARISON

In this section we compare the reliability of the Fortran and Ada programs based on the data given in Section 3. We first discuss the program characteristics, development environment and the adequacy of test cases used in this study. Then, we provide detailed analysis of the data to assess and compare program reliability.

Specifically, characteristics of the program are discussed in Section 4.1, adequacy of the test cases in Section 4.2 and the effect of errors on the number of incorrect CMVE's in Section 4.3.

A discussion of the reliability measures relevant to this study is given in Section 4.4. The key measure is the number of errors detected during various phases. A comparative assessment of the Fortran and Ada program reliability based on this measure is presented in Section 4.5. A further comparison for selected error categories is given in Section 4.6. An analysis of the effort data on development, testing and debugging times is given in Section 4.7. Finally, a summary of the comparison results is given in Section 4.8.

4.1 Program characteristics and development environment

As noted in Section 2, the programs in this study were developed in a university environment by graduate students in computer engineering. The main computer system used was a VAX 11/785 running under a VMS operating system. Both the Fortran and Ada compilers were supplied by DEC. No software development tools were available. A software package, MATCHER, was developed for checking the correctness of CMV elements during testing. Also, an Ada structural coverage program was used in this study.

The programs averaged about 640 lines in length with an average of 195 executable

statements. Several Ada language features, such as packages and exceptions were used in the Ada versions (see Table 2.2).

The programmers had experience in various programming languages for a total of 5 to 9 years (see Table 2.1). The intermediate Fortran programmers had an average experience of about 1.5 years in Fortran, and the intermediate Ada programmers had about 0.8 years of experience in Ada. The advanced programmers' experience in Fortran and Ada was 4 and 3 years respectively. Based on the above, it is reasonable to assume that the differences observed in the reliability characteristics of the programs written by the programmers can be attributed to the languages.

4.2 Test Case Adequacy

The hybrid functional-strucural approach used for LIP testing was intended to provide a good coverage of the program codes as well as their functions. All the programs were tested for the 54 test cases of Set 1 as discussed in Section 2.5 . One measure of adequacy of a test set is the number of different CMVE combinations tested and another measure is the code related coverage. The adequacy of this test set in terms of these measures is seen from Table 2.5. We note that the 54 test cases executed 46 different combinations of CMVE's. Further, as seen in Table 3.4 , the structural coverage provided by these test cases is very high. As an example, for program AL3 88% of executable statements, 81% of branches are executed by this test set. Similar high coverages were obtained for other programs. Based on such high values of functional and structural coverage, we feel that the test case selection approach developed in this study was very effective in revealing errors.

4.3 Effect of Error Removal on CMVE's

The results of program testing, as indicated by the number of incorrect CMVE's computed by each test were given in Table 3.1 for test set 1. As indicated in that Table, the number of incorrect CMVE's based on test case 1 for say, FL1 was 3 and for AL2 was 2. The total incorrect CMVE's, including repetitions, were 163, 292, 92, 87 and 43 for FL1, FL2, AL1, AL2 and AL3, respectively. A plot of the cumulative number of incorrect CMVE's versus test case number for FL1 is shown in Figure 4.1. It seems to follow a homogeneous Poisson process with respect to test case number.

Debugging of the programs was undertaken until all CMVE's were correctly computed by each program. For example, four errors were found in FL1 which were causing three incorrect CMVE's for test case 1. Similarly, three errors in AL2 were giving two incorrect CMVE's. The number of errors removed for each program to correct the CMVE's for test case 1 were given in Table 3.7 . Each program was run for the 54 test cases after removal of errors for this test case. The resulting incorrect CMVE's for various test cases are shown in Table 4.1. A plot of the cumulative number of incorrect CMVE's for FL1 versus test case number is given in Figure 4.2. We note that for this program, removal of four errors resulted in a reduction of incorrect CMVE's from 163 to 50. In other words, a large number of CMVE's were being incorrectly computed due to the presence of these four errors. Similar results were found for the other programs.

The above results can be used to determine the effect of debugging on the number of subsequent incorrect computations.

As indicated in Table 3.7, all CMVE's were computed correctly after removing 16, 18, 15, 5, 4 and 4 errors in FL1, FL2, FL3, AL1, AL2 and AL3, respectively.

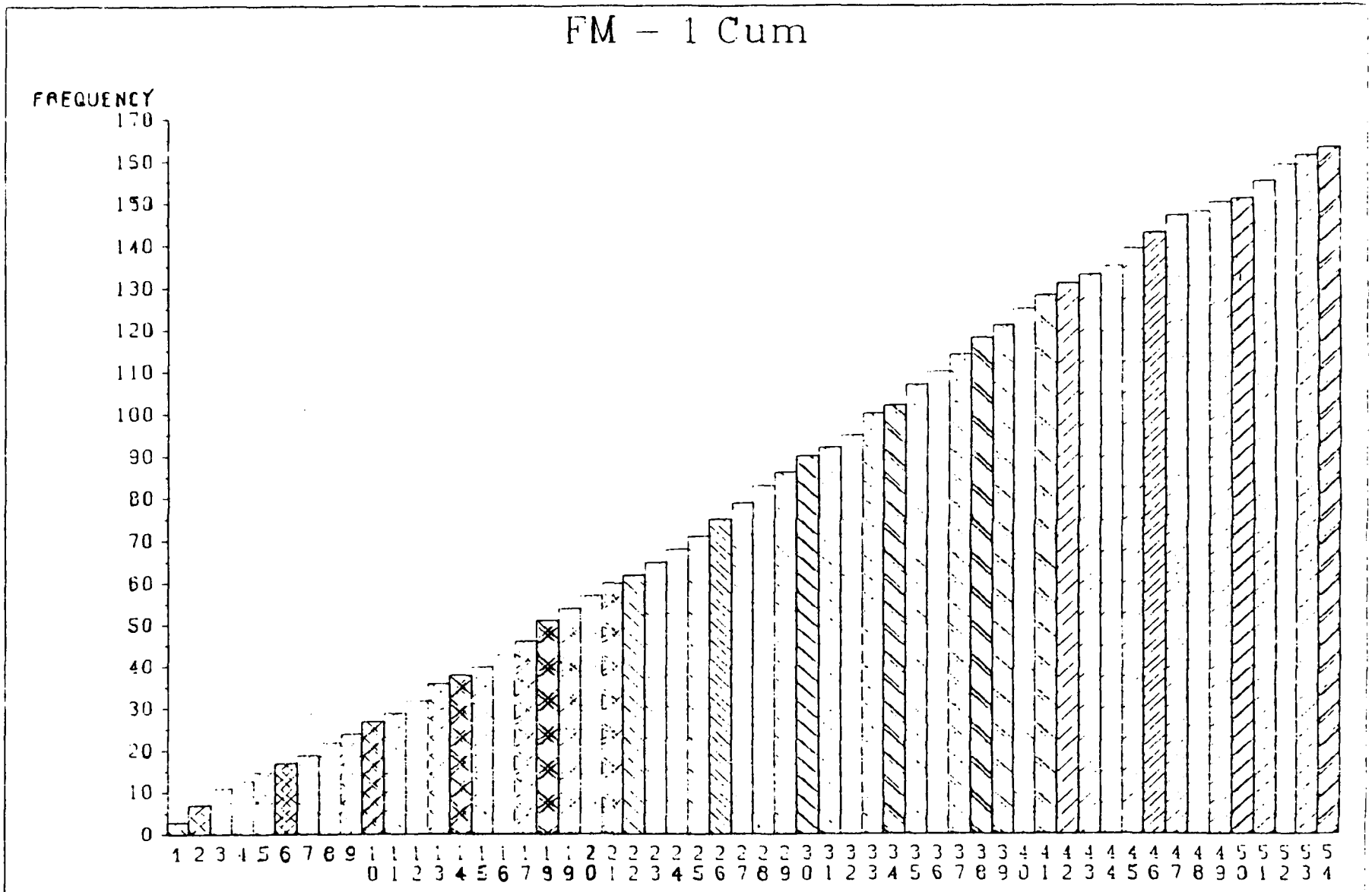

Figure 4.1. Cumulative Number of Incorrect CMVEs versus Test Case

(number prior to error removal)

Table 4.1

ERROR DETECTION FORM

TC #	FL1	FL2	AL1	AL2	AL3
1	12T				
	1 1	0 0	0 0	0 0	0 0
2		8T 9F 11F 12F 15T	14T	15T	
	0 1	5 5	1 1	1 1	0 0
3	2F 3F	7F 8T 9F 11F 15T	9F 11F		
	2 3	5 10	2 3	0 1	0 0
4		5T 9F 10F 11F 15T	10F 11F 14T		
	0 3	5 15	3 6	0 1	0 0
5		7T 9F 10F 11F 15T	10F 14T		
	0 3	5 20	2 8	0 1	0 0
6		8T 9F 10F 11F 13T 14F 15T	10F 15T		
	0 3	7 27	2 10	0 1	0 0
7		9F 10F 11F 13T 15T	10F		
	0 3	5 32	1 11	0 1	0 0
8	2F	7T 8T 13T	2F		
	1 4	3 35	1 12	0 1	0 0
9		7F 10F	2F 7F		
	0 4	2 37	2 14	0 1	0 0
10	2F	10F 13T	2F 7F		
	1 5	2 39	2 16	0 1	0 0
11		7T 8T 10F 13T			
	0 5	4 43	0 16	0 1	0 0
12		10F 11F 15T	10F	9T	9T
	0 5	3 46	1 17	1 2	1 1
13	14T 15F	3F 5T 9F 10F 11F	7F 14T		
	2 7	5 51	2 19	0 2	0 1
14		2T 5T 10F 11F 13T 15T	7F 10F		2T 9T
	0 7	6 57	2 21	0 2	2 3
15		7T 9F 13T 14F	2F 9F 14F		
	0 7	4 61	3 24	0 2	0 3
16	2F	5T 7T 9F 13T	2F 9F		
	1 8	4 65	2 26	0 2	0 3
17	2F	5T 7T 9F 13T 14F	2F 9F 14F		
	1 9	5 70	3 29	0 2	0 3
18	7T 15F	7T 10F 11F			
	2 11	3 73	0 29	0 2	0 3

Table 4.1 (Continued)

TC #	FM	FH	AT	AF	AE
19		5T 9F 10F 11F 13T 14F 15T	9F 10F 14F		
	0 11	7 80	3 32	0 2	0 3
20	11T	2T 10F 15T			2T 11T
	1 12	3 83	0 32	0 2	2 5
21		5T 10F 11F 15T		9T 14T	9T 14T
	0 12	4 87	0 32	2 4	2 7
22	12T 15F	10F 11F 13T	10F 15F		
	2 14	3 90	2 34	0 4	0 7
23		5T 9F 11F 13T 14F 15T	7F 11F		14F
	0 14	6 96	2 36	0 4	1 8
24		8T 13T			
	0 14	2 98	0 36	0 4	0 8
25	2F 3F	5T 7F	2F		
	2 16	2 100	1 37	0 4	0 8
26	2F 3F	5F 8T 10F 13T	2F 10F		
	2 18	4 104	2 39	0 4	0 8
27	15F	2T 5T 7T 11F 13T	15F	10T	2T 9T 10T
	1 19	5 109	1 40	1 5	3 11
28	14T 15F	5T 9F 11F	7F 14T	10T	10T
	2 21	3 112	2 42	1 6	1 12
29		9F 11F 13T 15T		10T	10T
	0 21	4 116	0 42	1 7	1 13
30	4T 11T	4T 5F 9F 15T	7F 14T	10T	10T
	2 23	4 120	2 44	1 8	1 14
31	11T	5F 9F 13T 15T		10T	10T 11T
	1 24	4 124	0 44	1 9	2 16
32	7T 11T	7T 8T 13T 15T			11T
	2 26	4 128	0 44	0 9	1 17
33	14T 15F	5T 9F 11F 13T	9F 11F 15F	14T	
	2 28	4 132	3 47	1 10	0 17
34		9F 11F 13T 14F 15T	9F 11F 14F		
	0 28	5 137	3 50	0 10	0 17
35	4T 7T 11T	4T 5F 7T 8T 9F 13T 15T	9F		
	3 31	7 144	1 51	0 10	0 17
36	15F	7F 9F 11F	2F 9F	10T	10T 11F 15F
	1 32	3 147	2 53	1 11	3 20

Table 4.1 (Continued)

TC #	FM	FH	AT	AF	AE
37		5T 8T 9F 11F 13T 15T	15T		11F
	0 32	6 153	1 54	0 11	1 21
38	4T 7T 11T	2T 4T 5F 7T 13T 15T		10T 11T 15T	2T 10T
	3 35	6 159	0 54	3 14	2 23
39	4T 11T	4T 5F 7F 8T 9F 10F 13T 14F 15T		11T	11F
	2 37	9 168	0 54	1 15	1 24
40	4T 11T	4T 5F 13T 15T			
	2 39	4 172	0 54	0 15	0 24
41	15F	5T 8T 9F 10F 11F 13T			11F 15F
	1 40	6 178	0 54	0 15	2 26
42	7T	8T 10F 11F 13T 15T	2F 11F		9T 15T
	1 41	5 183	2 56	0 15	2 28
43	10F	9F 10F 11F 15T	9F		15T
	1 42	4 187	1 57	0 15	1 29
44		7F 8T			
	0 42	2 189	0 57	0 15	0 29
45	2F 8T 13T	5T 7F 8T			
	3 45	3 192	0 57	0 15	0 29
46		8F 9F 10F 11F 15T	11F		15T
	0 45	5 197	1 58	0 15	1 30
47	2F 15F	5T 7F 8T 10F 11F 13T	2F 7F 15F		9T
	2 47	6 203	3 61	0 15	1 31
48	12T	7T 9F	9T 14T		
	1 48	2 205	2 63	0 15	0 31
49		7F 8T 13T			
	0 48	3 208	0 63	0 15	0 31
50		5T 9F	14T		
	0 48	2 210	1 64	0 15	0 31
51	15F	9F 10F 11F 13T	7F 10F		
	1 49	4 214	2 66	0 15	0 31
52	15F	2T 5T 7T 10F 11F 13T	10F 11F 15F		2T 9T
	1 50	6 220	3 69	0 15	2 33
53		9F 13T	14T		
	0 50	2 222	1 70	0 15	0 33
54		5T 7T 9F 11F 13T 14F 15T	9F 11F 14F		
	0 50	7 229	3 73	0	

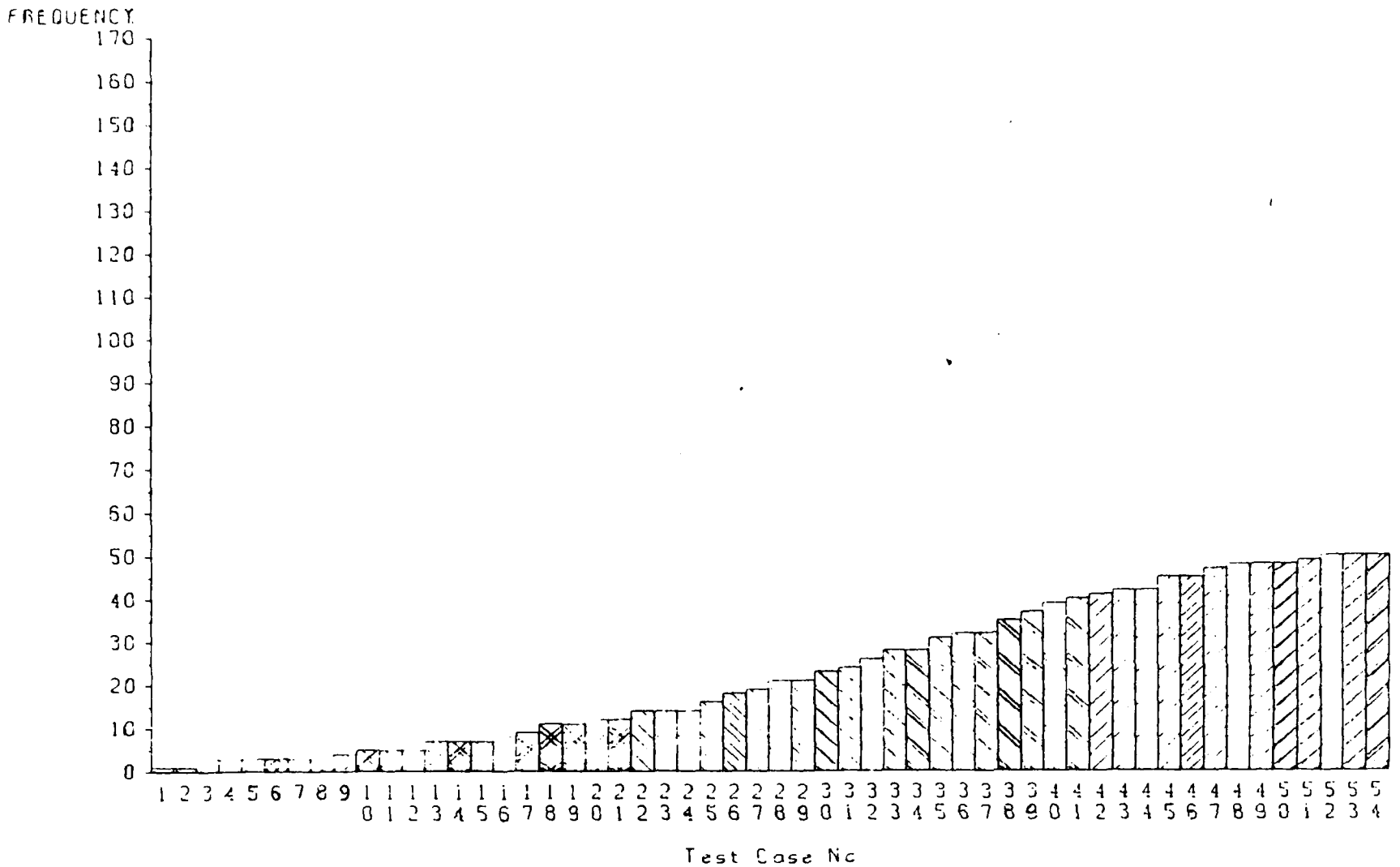

Figure 4.2. Cumulative Number of Incorrect CMVEs versus Test Case

(number after removal of error)

4.4 Reliability Measures

Reliability is generally defined as the ability of a program to perform its intended function in the specified environment. Since its true value cannot be determined, the number of errors found during development and testing can be used to assess program reliability. There are at least two ways to interpret such numbers. One is to consider a large value of detected errors as an indicator of high reliability, on the assumption that only a few errors are likely to be still remaining in the program. Another interpretation would be to consider a high value as a sign of error proneness and hence low reliability, because many more errors may not have been detected.

There is no general way to determine which, is either, of these viewpoints is correct. However, if two programs represent implementation of the same specification and are subjected to the same level of testing, the one with a larger number of detected errors is likely to be less reliable since it tends to indicate a poorer development approach and low software quality. Furthermore, error detection and removal does consume resources and hence a program with smaller number of intrinsic errors is clearly more desirable than the one with a larger error content.

In this study, we have considered several implementations of the same problem which were then subjected to the same test cases. Therefore, we can use the number of detected errors as a measure of program reliability. It is this measure that is used in Section 4.5 for reliability comparison.

Further, the structure and features of the language used can affect the number of errors due to design and coding as well as due to the programmer. Thus, a comparison

of the errors due to such causes can be used as an indicator of their relative reliability of programs in the languages. Another measure of reliability can be based on various types of errors. Specifically, a comparison of interface, control and data errors detected in these programs can provide additional useful information for purposes of reliability assessment. Reliability comparisons based on several error causes and error types are discussed in Section 4.6.

4.5 Reliability Comparison based on Total Errors

The number of errors detected during various phases of LIP development are summarized in Table 4.2. It lists the errors found during development and unit testing, function testing, testing for set 2 of 120 test cases, and testing for 100 and 1000 random tests. The data are grouped separately for intermediate and advanced programmers. We now determine the reliability characteristics of these programs based on the above data.

4.5.1 Errors in all Phases

If we assume that the errors found in various phases are of equal importance, we can use the total number of errors as a measure of program quality. Plots of the cumulative number of errors versus development phase are shown in Figures 4.3 and 4.4 for the programs by the intermediate and advanced programmers, respectively.

We note that the totals for FL1, FL2, AL1 and AL2 are 44, 47, 14, and 11, respectively. Since the programmers of these versions have almost the same level of experience, these values tend to indicate a much better quality of the programs in Ada compared to those in Fortran. A similar conclusion can be drawn from the cumulative number of errors for the advanced programmers.

TABLE 4.2

A Summary of the number of Errors found in various Phases

Number of errors						
Phase	FL1	FL2	AL1	AL2	FL3	AL3
Development & Unit Testing	24	28	8	7	10	4
Function Testing	16	18	5	4	15	4
Subtotal	40	46	13	11	25	8
Operational Testing 120 Test Cases	1	1	0	0	1	0
100 Random	1	0	1	0	1	0
1000 Random	2	0	0	0	0	0
Total after Function Testing	4	1	1	0	2	0
Total for all phases	44	47	14	11	27	8

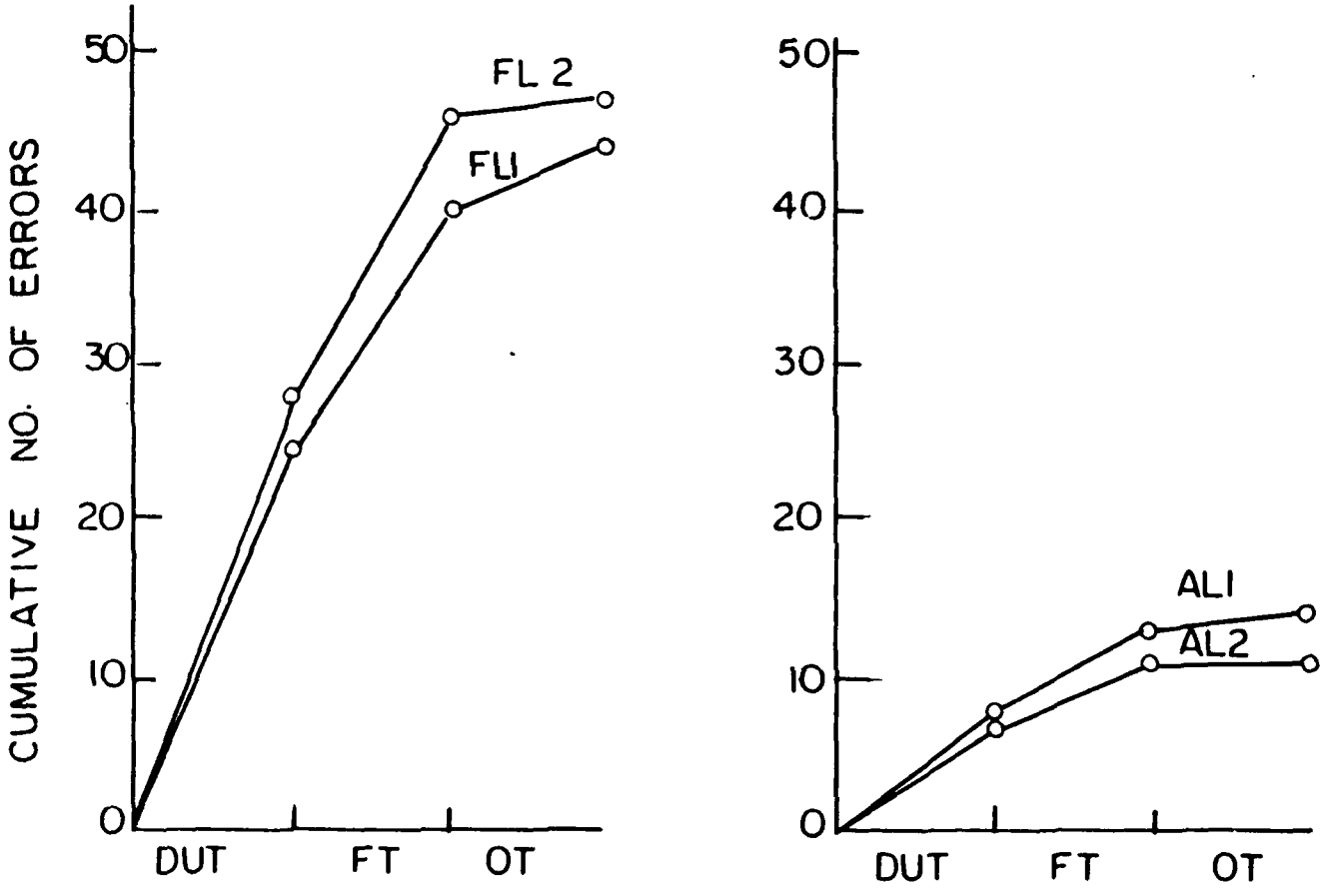

Figure 4.3. Cumulative Number of errors over Development Phases:

Intermediate Programmers

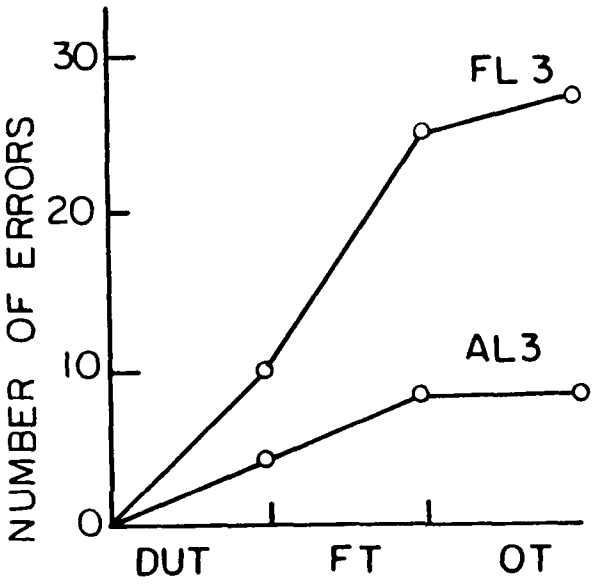

Figure 4.4. Cumulative Number of Errors over Development Phases:

Advanced Programmers

Sometimes it is useful to consider the error counts by normalizing for program size. For this purpose we compute the number of errors per 100 non-comment lines of code,i.e., the error density. These values, obtained from the size data in Table 2.2 and the error data in Table 4.2, are given in Table 4.3. A comparison of the values in the last row of this table also tends to indicate a higher quality of the Ada programs.

Next, we compute program reliability based on the average number of errors and average error density. The values of these quantities for the Fortran and Ada programs are summarized in Table 4.4. We note that on the average the Ada programs in this study have about 70% less error than the Fortran ones.

Finally, we show the error density values as the results of a 2^2 factorial design in Figure 4.5. From this we compute the main and interaction effects of language and experience level on error density as below:

Main effect of language = 1/2[(2.2 + 1.3) - (9.6 + 6.2)] = -6.15

Main effect of experience = 1/2[(6.2 + 1.3) - (9.6 + 2.2)] = -2.15

Interaction effect between languages and experience

= 1/2[(9.6 + 1.3) - (2.2 +6.2)] = 1.25

The above values are interpreted as follows. The average effect of Ada versus Fortran is to reduce error density by 6.15 units and of advanced versus intermediate programmers is to reduce error density by 2.15 units. The interaction effect between languages and experience is 1.3 units.

4.5.2 Errors during development and function testing

Since the contribution of errors during operational testing is very small, we now an-

TABLE 4.3

Number of Errors per 100 Non-comment Lines

Number of errors						
Phase	FL1	FL2	AL1	AL2	FL3	AL3
Development & Unit Testing	4.36	6.64	1.27	1.41	2.28	0.67
Function Testing	2.91	4.27	0.79	0.80	3.42	0.67
Subtotal	7.27	10.90	2.06	2.21	5.70	1.34
Operational Testing 120 Test Cases	0.18	0.24	0.0	0.0	0.23	0.0
100 Random	0.18	0.0	0.16	0.0	0.46	0.0
1000 Random	0.36	0.0	0.0	0.0	0.0	0.0
Total after Function Testing	0.72	0.24	0.16	0.0	0.46	0.0
Total for all phases	8.0	11.14	2.22	2.21	6.16	1.34

TABLE 4.4

Average of Total Errors and Error Densities

(All Phases)

Average of all Phases					
Programmers		F	A	F–A	(F–A)/Fx100
Intermediate	Average Number of Errors	45.5	12.5	33.0	72.5
	Average Error Density	9.6	2.2	7.4	77.1
Advanced	Average Number of Errors	27	8	19	70.4
	Average Error Density	6.2	1.3	4.9	79.0

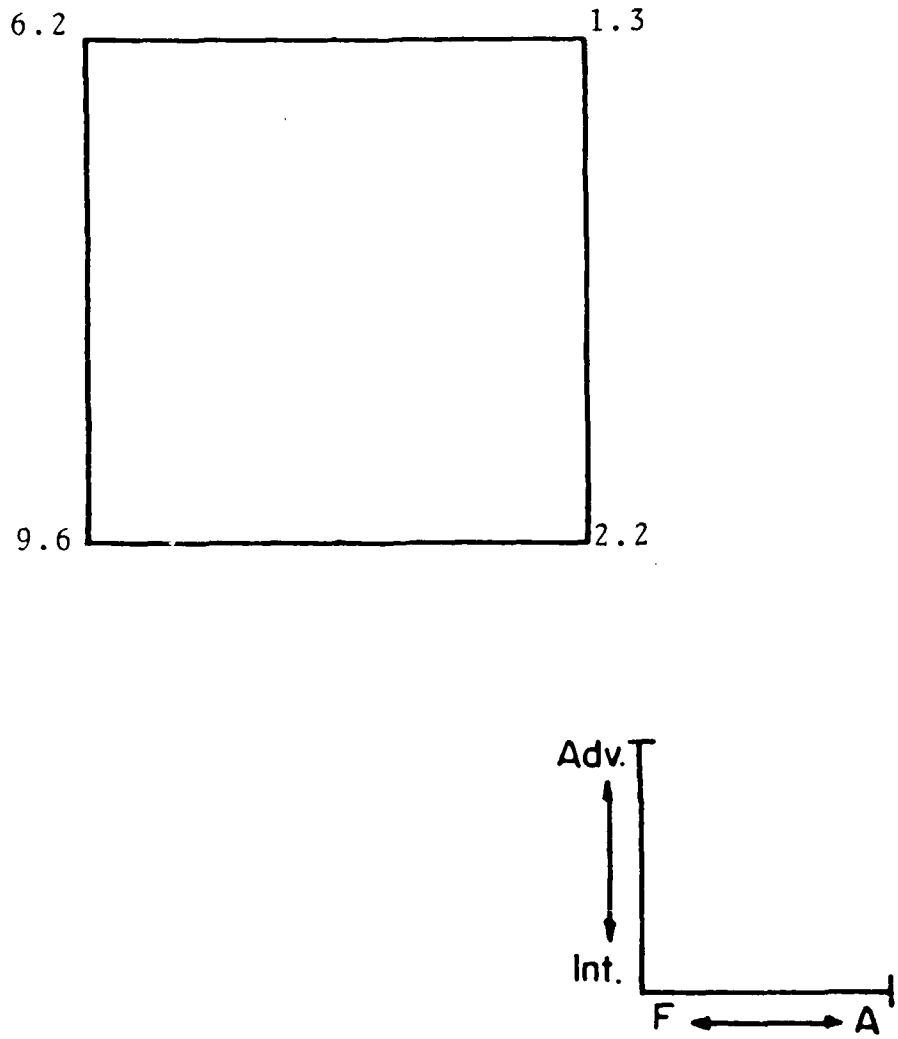

Figure 4.5. Data from a 2^2 Factorial Design for Average Error Densities: All Phases

alyze the error data from development and functional testing phases only. The average error densities for this case are shown in Figure 4.6 as the results of a 2^2 factorial design. The main and interaction effects for this case are obtained as follows:

Main effect of language = 1/2[(2.1 +1.3) - (9.1 + 5.7)] = -5.7

Main effect of experience = 1/2[(5.7+1.3) - (9.1+2.1)] = -2.1

Interaction effect of language and experience

= 1/2[(9.1 +1.3) - (2.1+5.7)] = 1.3

We can also assess the precision of these effects by obtaining an estimate of their error variance. This is done from the replicated data in Table 4.3 as follows:

V(Effect) = 3/4 V(Subtotal),

V(Subtotal) = 1/2[(10.90 - 7.27) + (2.21 - 2.06)] = 1.8

V(Effect) = 1.35 with 2 degrees of freedom

The values of the effects with one standard deviation are:

Main effect of language = -5.7 - 1.35 to -5.7+1.35

Main effect of experience = -2.1 - 1.35 to -2.1+1.35

Interaction effect between language and experience

= 1.3 - 1.35 to 1.3+1.35

Clearly from the data analyzed here, language does have a significant effect on the number of errors detected during development, unit and functional testing.

4.5.3 Errors During function and Operational Testing

It can be argued that the errors found during development and unit testing are not truly representative of program reliability and should be excluded from comparison. For

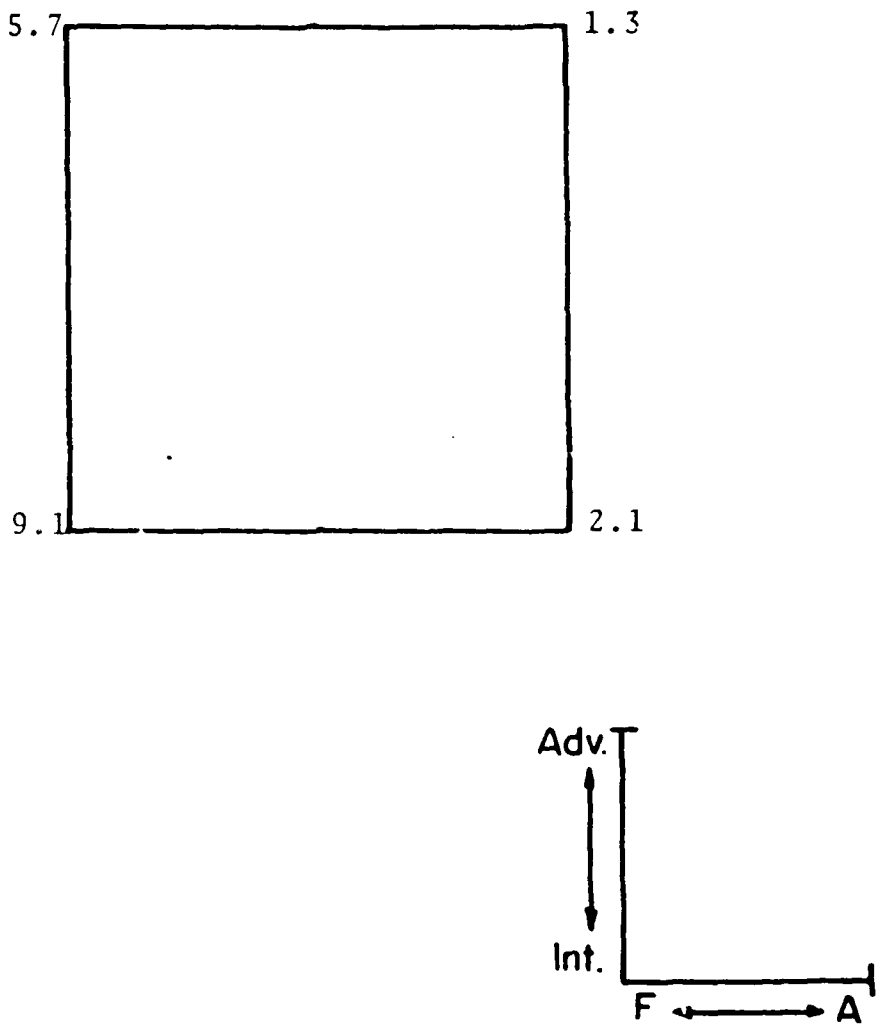

Figure 4.6. Data from a 2^2 Factorial Design for Average Error Densities: Development, Unit Testing and Functional Testing

functional and operational testing phases, the average number of errors and error densities are given in Table 4.5.

Using average error density as a measure, the results for the 2^2 factorial design are shown in Figure 4.7. From this data, the average effects are obtained as follows:

Main effect of language = 1/2[(0.88-4.07)+(0.67 -3.88)] = -3.20

Main effect of experience = 1/2[(3.88-4.07)+(0.67-0.88)] = -0.20

Interaction effect between language and experience

= 1/2[(4.07+0.67) - (0.88+3.88)] = 0.20

Based on both the data in Table 4.5 and the main effect of language, we again note that the Ada programs are about 75% less error-prone than the corresponding Fortran programs.

4.5.4 Errors during Operational Testing

Errors found during operational testing alone can be considered to be another good measure of reliability. One can correctly argue that this measure represents the true reliability of the programs in operation.

The relevant data for this case is summarized in Table 4.6. Again, the relatively higher reliability of the Ada programs is clearly seen from the data given here.

4.6 Reliability Comparison based on Error categories

While the total number of errors provides a good measure of reliability, additional information about program reliability can be obtained by analyzing errors for selected categories. In this section we compare the reliability of Fortran and Ada programs with respect to errors created during design and coding, errors due to programmers and data

TABLE 4.5

Average Number of Errors and Error Densities

(Functional and Operational Testing)

Functional Operational Testing					
Programmers		F	A	F-A	(F-A)/Fx100
Intermediate	Average Number of Errors	19.5	5.0	14.5	74.4
	Average Error Density	4.07	0.88	3.19	78.4
Advanced	Average Number of Errors	17	4	13	76.5
	Average Error Density	3.88	0.67	3.21	82.7

TABLE 4.6

Average Number of Errors and Error Densities

(Operational Testing)

Operational Testing					
Programmers		F	A	F-A	(F-A)/Fx100
Intermediate	Average Number of Errors	2.5	0.5	2.0	80.0
	Average Error Density	0.48	0.08	0.40	83.3
Advanced	Average Number of Errors	2	0	2	100
	Average Error Density	0.46	0.0	0.46	100

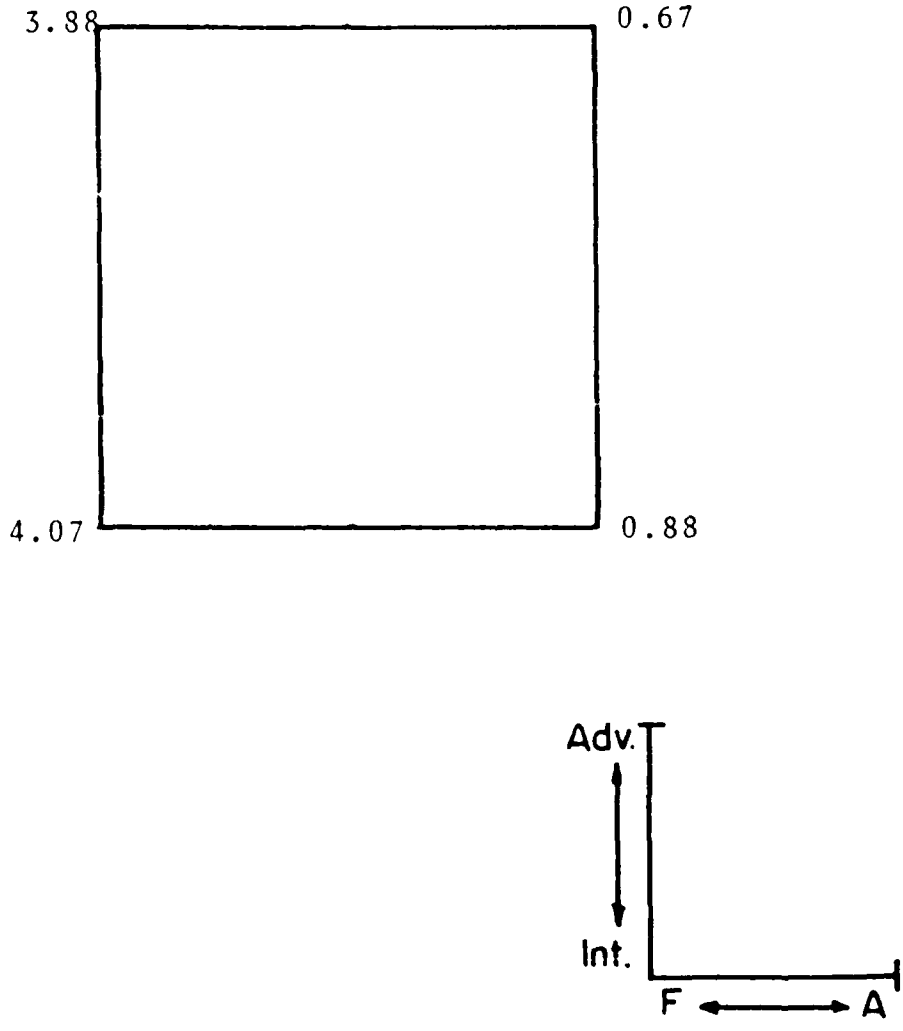

Figure 4.7. Data from a 2^2 Factorial Design for Average Error densities: Functional and Operational Testing

errors.

The errors found during unit and function testing for these and other categories were given in Tables 3.6 and 3.8 respectively. Some relevant data from these tables are summarized in Tables 4.7 and 4.8. Table 4.7 gives the total and average number of errors in stated categories for both the unit and function testing phases. For example, the total number of programmer errors during unit and function testing for programs FL1 and FL2 is 22 with an average of 11 errors per program.

Table 4.8 is a summary of error data simular to that of Table 4.7 except that the numbers here are for function testing alone.

We note that the percentage differences in the errors between Fortran and Ada programs indicate a substantially higher reliability for the programs in Ada.

4.7 Analyses of Effort Data

Two categories of effort data were collected in this study as described in Section 3.6. The first category was the development, unit testing and debugging effort. Times spent in these activities for the six programs were given in Table 3.10. The data about the second category, the testing and debugging effort, was listed in Table 3.11. Further breakdown of this data into error isolation and error removal times was given in Table 3.12 and 3.13 respectively.

Plots of the development effort data for different development phases are shown in Figures 4.8 and 4.9 for the intermediate and advanced programmers, respectively. Recall that structure based unit testing was done for many of the modules in FL1 and FL1. That is why the unit test case generation and unit testing and debugging times are so high for

TABLE 4.7

Total and Average Errors for selected categories

during Unit and Function Testing

			Error Categories			
			Design	Coding	Programmer	Data
Intermediate	Fortran	Total	27	59	30	24
		Average	13.5	29.5	15.0	12.0
	Ada	Total	10	14	4	9
		Average	5.0	7.0	2.0	4.5
Average Percentage Difference			62.9	76.3	86.7	37.5
Advanced	Fortran	Total	10	15	12	7
	Ada	Total	3	5	4	2
Percentage difference			70.0	66.7	66.7	71.4

TABLE 4.8

Total and Average Errors for selected categories

during Function Testing

			Error Categories			
			Design	Coding	Programmer	Data
Intermediate	Fortran	Total	12	22	14	8
		Average	6.0	11.0	7.0	4.0
	Ada	Total	6	3	1	4
		Average	3.0	1.5	0.5	2.0
Average Percentage Difference			50.0	86.4	92.9	50.0
Advanced	Fortran	Total	6	9	9	5
	Ada	Total	3	1	0	0
Percentage difference			50.0	88.9	100	100

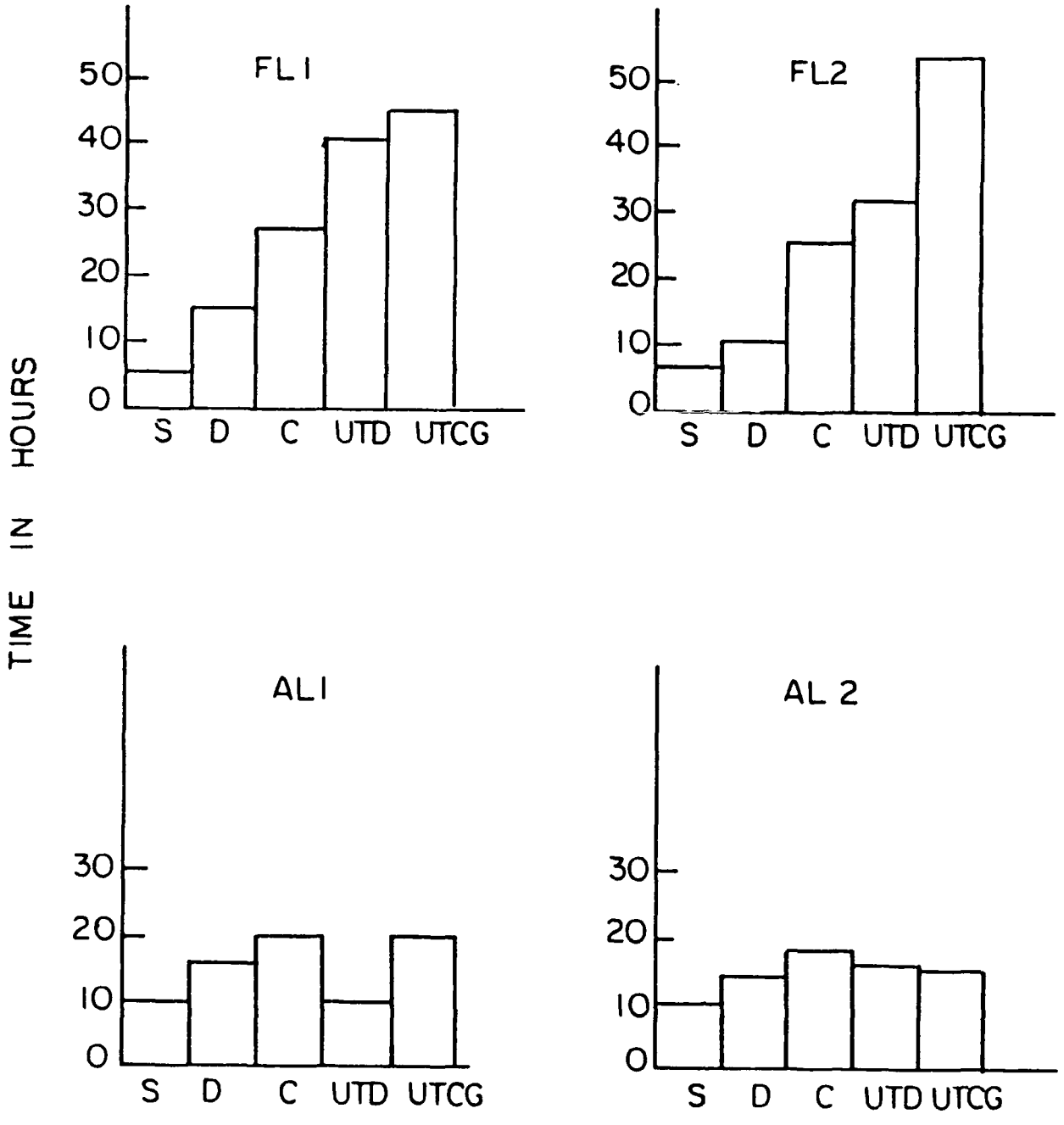

Figure 4.8. Time Spent in Various Development Phases:

Intermediate Programmers

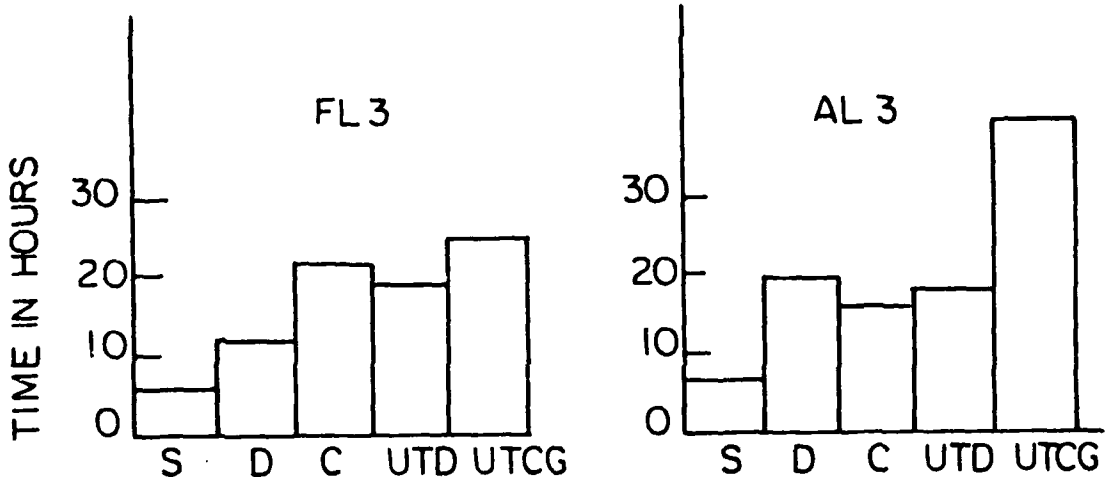

Figure 4.9. Time Spent in Various Development Phases:

Advanced Programmers

these two programs. If we ignore the effort on these two activities, we note that the times spent on specification, design and coding for all the programs was very close to each other. In other words, except for the unit testing activity, there was no significant difference in the development efforts of the Fortran and Ada programs. However, if we include unit testing, FL1 and FL2 took about 60% longer than the other programs to develop.

The function testing for all programs was done using the same test cases and hence the effort was almost identical. The times spend in isolating and removing the errors found by the 54 test cases of set number 1 varied significantly among the programs as seen in Table 3.11. These data are plotted in Figure 4.10. Clearly, the Fortran programs took much longer to correct than the Ada programs.

To compare the relative ease of detecting and correcting errors in the Fortran and Ada programs, we now analyse the data in Tables 3.11 and 3.12. These data are plotted in Figures 4.11 and 4.12,respectively. From Figure 4.11 we note that errors in Fortran took longer to isolate than in Ada. Similarly, from Figure 4.12, we see that all of the Ada errors were fixed in less than one hour each while some errors in Fortran programs took up to four hours each to fix. Overall, the testing and debugging effort for the Fortran programs was much higher than for the Ada programs.

4.8 Summary of Reliability Comparisons

In the previous Sections, we provided a comparative assessment of the relative reliability of Ada and Fortran programs. For this purpose, we employed measures such as number of errors, error density, error causes and error types. In this Section, we provide a summary of these comparisons.

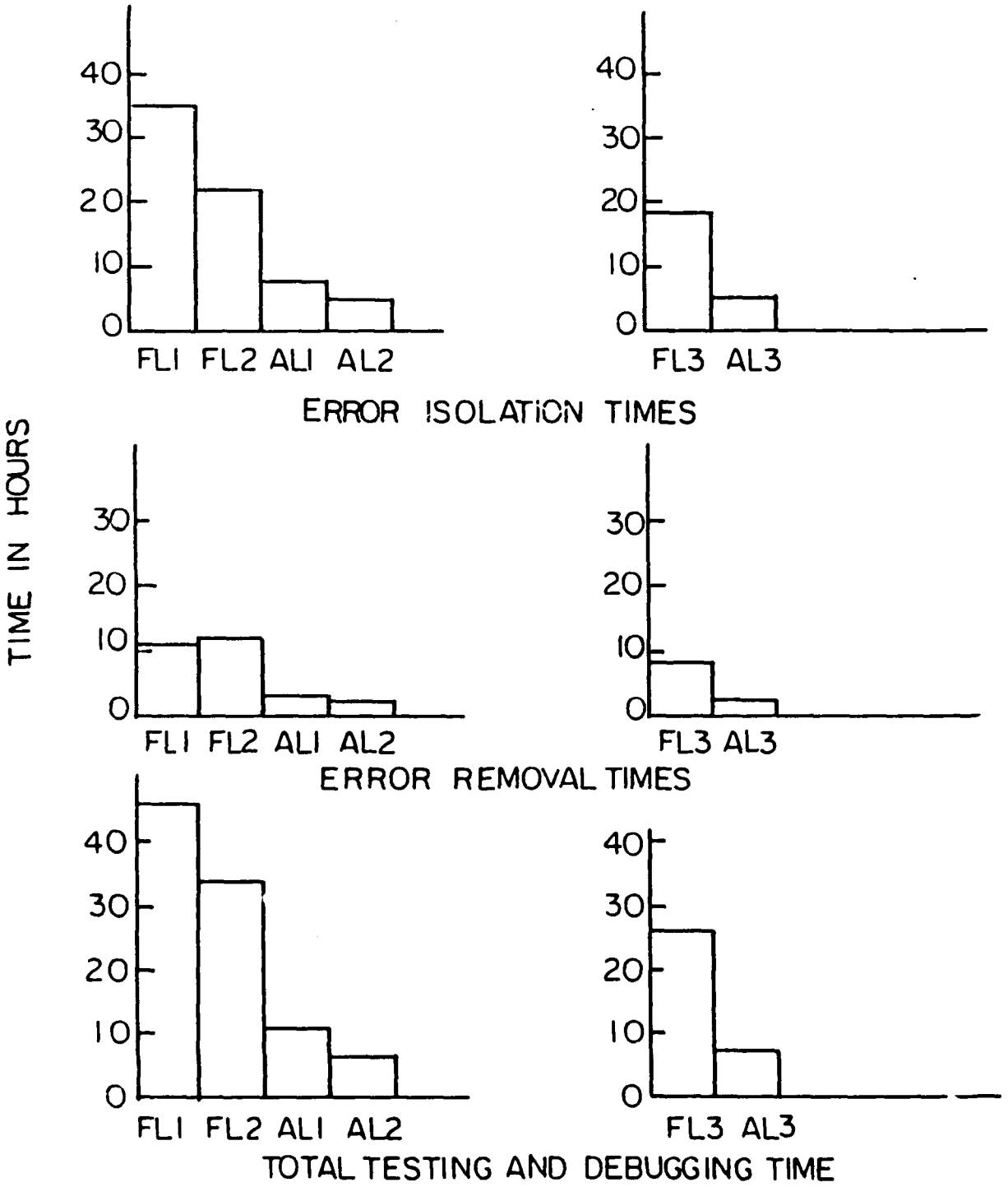

Figure 4.10. Effort Distribution During Function Testing and Debugging

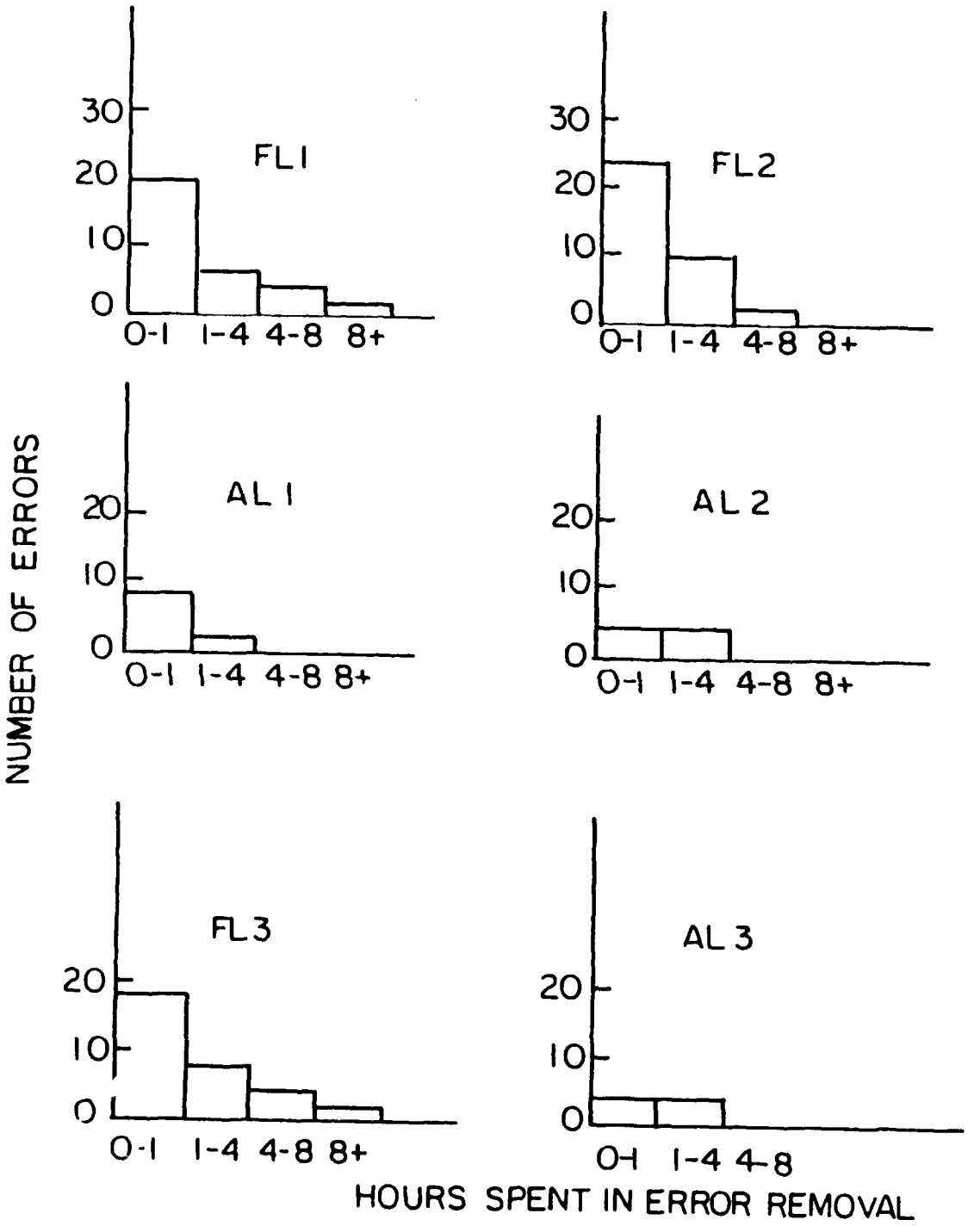

Figure 4.11. Distribution of Error Isolation Times

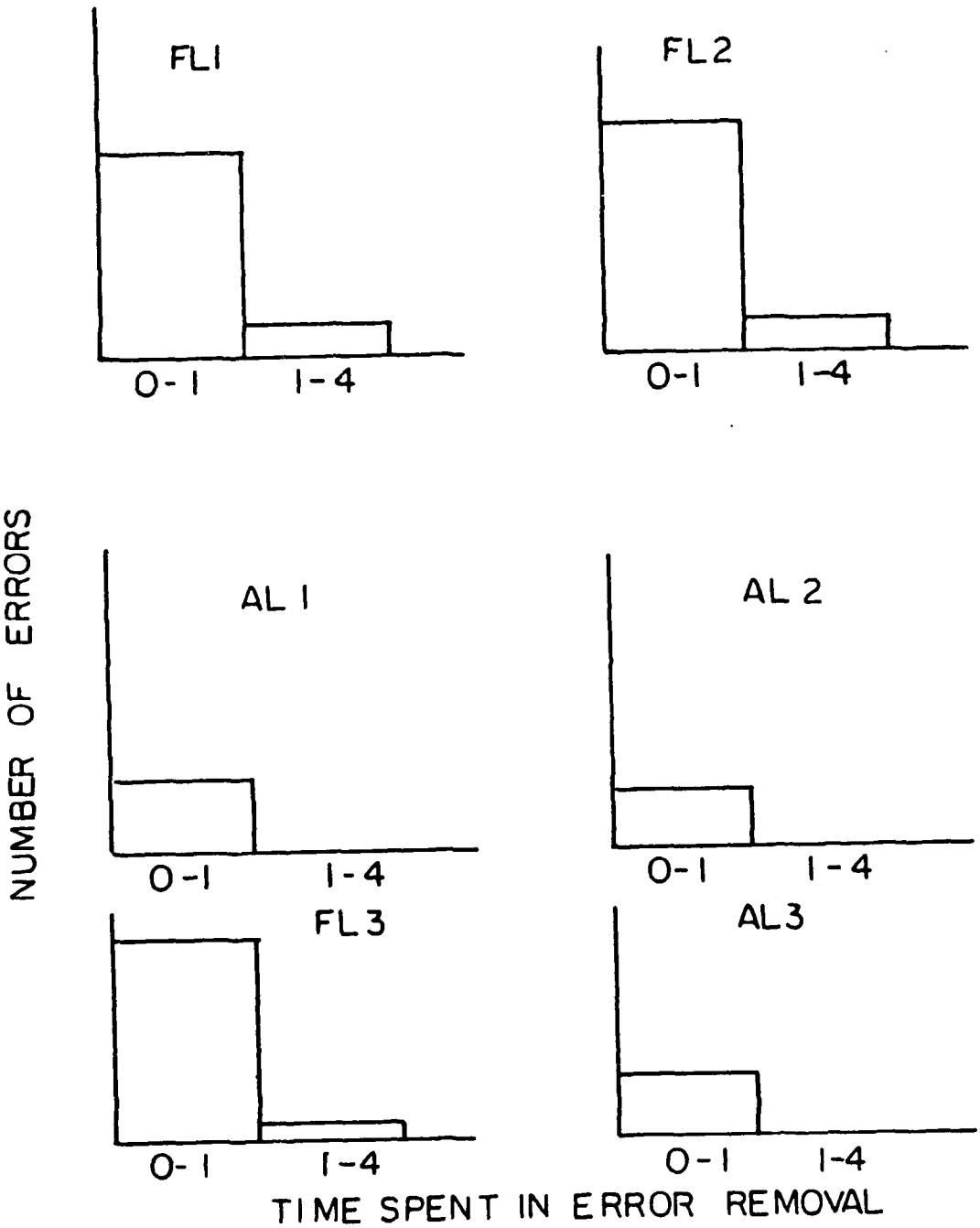

Figure 4.12. Distribution of Error Removal Times

A listing of the percentage differences between the number of errors in the Fortran and Ada programs is given in Table 4.9. The values in this Table are obtained from the analyses in Sections 4.5 and 4.6. A study of the data in this table clearly indicates that the Ada programs are about 70% less error-prone than those in Fortran.

TABLE 4.9

A Summary of Reliability Comparisons

Percentage Difference		
Category	Intermediate	Advanced
Total Errors		
All Phases	72.5	70.4
Development and Functional Testing	72.1	68.0
Functional and Operational Testing	74.4	76.5
Operational Testing	80.0	100.0
Error Density		
All Phases	76.9	77.2
Development and Functional Testing	75.9	76.5
Functional and Operational Testing	78.4	82.7
Error Causes:All phases		
Design	59.0	62.5
Coding	79.0	75.0
Programmer	86.7	66.7
Error Types:All Phases		
Control	92.0	85.7
Interface	100.0	66.7

5. CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The main objective of this study was to experimentally assess and compare the effects of Ada and Fortran languages on program reliability. Towards this objective, six versions of one problem and two of another were implemented by six different programmers; three of them programmed the Fortran and three the Ada versions.

The experimental design employed here was a subset of a 2^3 full factorial design, i.e., a design in three variables, each at two levels. The variables and their levels were language (Fortran and Ada), programmer experience (intermediate and advanced) and application type (scientific and text processing). For reasons discussed earlier, the main emphasis was on the data from a 2^2 factorial design in two variables, language and programmer experience. Some of the experimental points in this design were replicated to determine precision of the effects of language and experience on program reliability. The scientific problem used here was the Launch Interceptor Program (LIP), a simple but realistic, anti-missile system. The programmers were graduate students in computer engineering with varying degrees of experience in programming languages.

All versions of LIP were subjected to the same functional and operational test cases. For function testing, the test cases were developed manually using a hybrid of structure dependent and specification dependent techniques. Some operational test cases were developed in a similar fashion and some were randomly selected. The methodology for developing test cases used here was very effective in providing a high level of functional and structural coverage of the programs. Since it was not feasible to determine the oracle

for testing the LIP's, a variation of majority-voting technique was employed which proved to be very effective in error detection and isolation.

Data on errors were collected during development and unit testing, function testing and operational testing. Some data on design and code metrics, as well as on structural coverage during testing, were also recorded. The key measure of reliability used in this study was the number of errors found during various development phases.

Reliability comparisons between Ada and Fortran programs were based on the total number of errors, as well as on the number of errors found during development and functional testing, functional and operational testing and operational testing. Some comparisons were also based on error density, the number of errors per 100 non-comment lines of code. Further analyses were done for some error causes as well as for error types. Following are the main results of these analyses.

The total number of errors in the Ada programs was about 70% less than in the Fortran program. If errors during development and unit testing, were excluded,i.e., if only functional and operational testing data were considered, the Ada program had about 78% less errors. Similar differences were found when comparing the number of errors during different phases and due to error causes and error types.

Using error-density during development and functional testing as a measure, the average difference between Ada and Fortran program was 5.7 errors per 100 non-comment lines with a standard deviation of 1.35 units. The effect of programmers' experience was to reduce error density by 2.1 with a standard deviation of 1.35 units. If data during functional and operational testing alone is considered, then the Ada programs had 3.20

less errors per 100 non-comment lines.

Regarding the development effort, the time spent in specification reading, design and coding was almost the same for all programs. However, unit testing efforts differed due to differences in the extent of such testing. Analyses of the data on error isolation and correction indicated that the Ada programs were easier to debug than the Fortran programs.

Given above are the main important results of this study. What the above numbers mean is that there is a statistically significant evidence in support of higher reliability of Ada programs. The extent of the difference in reliability, however, is likely to vary from one application to another and across different development environments.

5.2 Recommendations

The results of this study were based primarily on several implementations of one scientific program written by student programmers. The claim of higher reliability of the Ada programs was found to have statistically significant evidence in this experiment. However, the extent of this difference is likely to change with different types of programs written by relatively more experienced programmers. It is recommended that the experiment design used here be employed to collect additional data and obtain further confirmatory evidence in support of Ada reliability.

The features of Ada used in this study were those relevant to the problem. The effect of these features, such as concurrency, on reliability should be explored for applications where they are more relevant.

Reusability and ease of maintenance are two well advertised features of Ada. The programs developed in this study can be used to explore these features and compare them

with the corresponding Fortran programs.

The use of N-version programming (NVP) for software fault-tolerance has been popular for some time. The testing results of this study provided some interesting insights into the reliability of NVP. Further experimental work on this topic is recommended for developing ultra-high reliable software.

6. REFERENCES

[Basili et al. 85] V.R. Basili, E.E. Katz, N.M. Panlilio-Yap, C.L. Ramsey, and S. Chang, "A Quantitative Characterization and Evaluation of a Software Development in Ada ", IEEE Computer,pp 53-65, September.

[Basili & Perricone 84] V.R. Basili and B.T. perricone,"Software Errors and Complexity: An Empirical Investigation", Communications of ACM, 27, 1, pp 42-52, January

[Basili & Weiss 84] V.R. Basili and D.M. Weiss, "A Methodology for Collecting Valid Software Engineering Data", Trans. Software Engr. SE-10, 6, pp 728-738, November.

[Bergland 81] G.D. Bergland, "A Guided Tour of Program Design Methodologies", IEEE Computer, pp 126-149, October.

[Booch 83] G. Booch, Software Engineering with Ada, Benjamin/Cummings Publishing Co.,1983.

[Box, Hunter, & Hunter 78] G. E. P. Box, W. G. Hunter, and J. S. Hunter, Statistics for Experimenters, John Wiley & Sons, New York, 1978.

[Di Millo et al. 87] Di Millo et al, Software Testing and Evaluation, Benjamin/Cummings Publishing Co., 1987.

[Fairley 85] R. Fairley, Software Engineering Concepts, McGraw Hill Book Co.,1985.

[Gannon & Horning 75] J. D. Gannon and J. J. Horning, "Language Design for Programming Reliability," IEEE Transactions on Software Engineering, Vol. SE-1,TNo. 2. pp. 179-191, June.

[Gannon et al. 83] J. D. Gannon, E. E. Katz, and V. R. Basili,"Characterizing Ada Programs: Packages",The Measurement of Computer Software Performance, Los Alamos

National Laboratory, August 1983.

[Gehani 83] N. Gehani, Ada: An Advanced Introduction, 1983.

[Goel 83] A. L. Goel,"A Guidebook for Software Reliability Assessment," RADC-TR-83-176, August 1983.

[Goel 83] A. L. Goel,"Software Reliability Models: Assumptions, Limitations and Applicability," IEEE Trans. on Software Engineering, Vol. SE-11, pp 1411-1423, December.

[Goel & Valdes 83] A. L. Goel and P. Valdes, "Software Testing Techniques," Technical Report, Syracuse University, 1983.

[Goodenough & Gerhart 77] J. Goodenough and S. Gerhart,"Towards a Theory of Test Data Selection Criteria," Current Trends in Programming Methodology", Vol. 2 R. Yeh (Ed), Prentice Hall, pp 44-79, 1977.

[Knight & Leveson 86] J. C. Knight and N. G. Leveson,"An Experimental Evaluation of the Assumption of Independence in Multiversion Programming", IEEE Trans. on Software Engineering, vol. SE-12, pp 96-109, January.

[Meyer 85] B. Meyer, "On Formalism in Specifications", IEEE Software pp 6-26, January.

[Naur 69] Peter Naur, "Programming by Action Clusters",BIT vol. 9, No.3, pp 250-258.

[Ostrand & Weyuker 83] T. J. Ostrand and E. J. Weyuker,"Collecting and Categorizing Software Error Data in an Industrial Environment",Dept. Com. Sci., Courant Inst. Math,Sci, New York Univ., NY, Tech. Rep. 47, August 1982 (Revised May 1983).

[Redwine 83] S. T. Redwine,"An Engineering Approach to Software Test Data De-

sign". IEEE Transactions on Software Engineering, vol. SE-9, pp. 191-200, March.

[Selby 85] R. W. Selby,Jr., Evaluation of Software Technologies: Testing CLEANROOM and Metrics, Ph. D. Dissertation, University of Maryland, College Park.

[Weyuker 82] E. Weyuker, "On Testing Non-Testable Programs", The Computer Journal Vol. 25, No.4, pp 465-470.

[Wichmann 84] B. A. Wichmann, "Is Ada Too Big ? A Designer Answers to Critics". CACM, pp 98-103, February.

[Wiener & Sincovec 84] R. Wiener and R. Sincovec, Software Engineering with Modula-2 and Ada, John Wiley, 1984.

APPENDIX A

FUNCTIONAL REQUIREMENTS FOR THE LAUNCH INTERCEPTOR PROBLEM

INPUT VARIABLES, RANGES

The following values are global variables available to the LIP as inputs.

NUMPOINTS - The number of planar data points.
(1 <= NUMPOINTS <= 100)

X - Array containing the X coordinates of data points.

Y - Array containing the Y coordinates of data points.

PARAMETERS - A record holding parameters for LIC's:

LENGTH1 - (0 <= LENGTH1)
RADIUS1 - (0 <= RADIUS1)
EPSILON - (0 <= EPSILON <= PI)
AREA1 - (0 <= AREA)
Q_PTS - (2 <=Q_PTS <= NUMPOINTS)
QUADS - (1 <= QUADS <= 3)
DIST - (0 <= DIST)
N_PTS - (3 <= N_PTS <= NUMPOINTS)
F_PTS - (1 <= F_PTS <= NUMPOINTS - 2)
A_PTS - (A_PTS + B_PTS <= NUMPOINTS - 3)
B_PTS - (A_PTS + B_PTS <= NUMPOINTS - 3)
C_PTS - (C_PTS + D_PTS <= NUMPOINTS - 3)
D_PTS - (C_PTS + D_PTS <= NUMPOINTS - 3)
E_PTS - (E_PTS + F_PTS <= NUMPOINTS - 3)
F_PTS - (E_PTS + F_PTS <= NUMPOINTS - 3)
G_PTS - (1 <= G_PTS <= NUMPOINTS - 2)
LENGTH2 - (0 <= LENGHT2)
RADIUS2 - (0 <= RADIUS2)
AREA2 - (0 <= AREA2)

LCM - Logical connector matrix. Valid values: NOTUSED, ORR, ANDD.
(LCM[LICRANGE,LICRANGE] where 1 <= LICRANGE <= 15).

PUM (diagonal elements) - Valid values: boolean TRUE, FALSE.
(PUM[LICRANGE,LICRANGE] where 1 <= LICRANGE <= 15).

OUTPUT VARIABLES

The following values are global variables available to the LIP as outputs.

PUM (off diagonal elements) - Valid values: boolean TRUE, FALSE.
(PUM[LICRANGE,LICRANGE] where 1 <= LICRANGE <= 15).

CMV - Valid values: boolean TRUE, FALSE.
(CMV[LICRANGE] where 1 <= LICRANGE <= 15).

PUV - Valid values: boolean TRUE, FALSE.

(FUV[LICRANGE] where 1 <= LICRANGE <= 15).

LAUNCH - Valid values: boolean. TRUE launches.

CONSTANTS

PI - 3.1415926535

EXTERNAL FUNCTION CALLS

REALCOMPARE - Compares real numbers. Invocation: REALCOMPARE (A,B), returns LT: A < B, EQ: A = B, GT: A > B.

REQUIRED COMPUTATIONS

It is assumed that all input (and output) variables have consistent units, and require no unit conversion.

CMV Generation

The CMV elements are set according to the results of the LIC calculations. There are fifteen LIC calculations. The corresponding CMV element is set to TRUE if the LIC is satisfied, and FALSE if it is not.

LIC Calculations

1. There exists at least one set of two consecutive data points that are a distance greater than LENGTH1 apart.

2. There exists at least one set of three consecutive data points that cannot all be contained on or within a circle of radius RADIUS1.

3. There exists at least one set of three consecutive data points which form an angle such that:

angle < (PI - EPSILON), or angle > (PI + EPSILON).

The second of the three data points is the vertex. If either the first or third of the three data points coincides with the second, the angle is undefined, and the LIC is not satisfied by those three points.

4. There exists at least one set of three consecutive data points that are the vertices of a triangle with area greater than AREA1.

5. There exists at least one set of Q_PTS consecutive data points that lie in more tha QUADS quadrants. Where there is ambiguity as to which quadrant contains a given point, priority of decision will be by quadrant number, that is, 1, 2, 3, 4. The data point (-1.0) is in quadrant 2, the point (0,-1)is in quadrant 3, and the points (0,0), (0,1), and (1,0) are in quadrant 1.

6. There exists at least one set of two consecutive data points (X[i],Y[i]) and X[j],Y[j], such that X[j] - X[i] ≠ 0, where i = j-1.

7. There exists at least one set of N_PTS consecutive data points such that at least one of the points lies a distance greater than DIST from the line joining the first and last of these N_PTS points. If the fist and last of these N_PTS are identical, then the calculated distance to compare with DIST will be the distance from the coincident point to all other points of the N_PTS consecutive points. The condition is noit met when NUMPOINTS < 3.

8. There exists at least one set of two data points separated by exactly F_PTS consecutive intervening points that are a distance greater than the length, LENGHT1, apart. This condition is not met when NUMPOINTS < 3.

9. There exists at least one set of three data points separated by exactly A_PTS and B_PTS consecutive intervening points, respectively, that cannot be contained within or on a circle of radius RADIUS1. The condition is not met when NUMPOINTS < 5.

10. There exists at least one set of three data points separated by exactly C_PTS and D_PTS consecutive intervening points, respectively, that form an angle such that:

angle < (PI - EPSILON), or angle > (PI + EPSILON).

The second of the three data points is the vertex. If either the first or third of the three data points coincides with the second, the angle is undefined, and the LIC is not satisfied by those three points. This LIC is not satisfied if NUMPOINTS < 5.

11. There exists at least one set of three data points separated by exactly E_PTS and F_PTS consecutive intervening points, respectively, that are the vertices of a triangle with area greater than AREA1. The condition is not met when NUMPOINTS < 5.

12. There exists at least one set of two data points, X[i], Y[i] and X[j], Y[j], separated by exactly G_PTS consecutive intervening points, such that X[j] - X[i] ≠0, where i < j. The condition is not met when NUMPOINTS < 3.

13. There exists at least one set of two data points, separated by exactly H_PTS consecutive intervening points, which are a distance greater than length, LENGTH1, apart. In addition, there exists at least one set of two data points separated by exactly I_PTS consecutive intervening points, that are a distance less than length, LENGTH2, apart. The condition is not met when NUMPOINTS < 3.

14. There exists at least one set of three data points, separated

by exactly A_PTS and B_PTS consecutive intervening points, respectively, that cannot be contained on or within a circle of radius RADIUS1. In addition, there exists at least one set of three data points separates by exactly A_PTS and B_PTS consecutive intervening points, respectively, that can be contained on or within a circle of radius RADIUS2. This condition is not met when NUMPOINTS < 5.

15. There exists at least one set of three data points, separated by exactly E_PTS and F_PTS consecutive intervening points, respectively, that are the vertices of a triangle with area greater than AREA1. In addition, there exist three data points separated by exactly E_PTS and F_PTS consecutive intervening points, respectively, that are the vertices of a triangle with area less than AREA2. The condition is not met when NUMPOINTS < 5.

PUM Off-Diagonal Element Generation

The PUM off diagonal elements are generated by operation of the LCM on the CMV. The entries in the LCM represent the logical connectors to be used between pairs of LIC's to determine the corresponding entry in the PUM. LCM[i,j] represents the boolean operator to be applied to CMV[i] and CMV[j], the result being placed in PUM[i,j]. The operators are represented by ANDD, ORR, and NOTUSED. ANDD and ORR are to be used as boolean operators AND and OR, respectively. The operation by NOTUSED will always result in TRUE.

PUV Generation

The PUV is produced from the PUM. An element in the PUV is set to TRUE if one of two conditions are satisfied by the PUM. If a diagonal element of the PUM is false, its corresponding element in the PUV is set to TRUE, i.e., if PUM[i,i] is FALSE, then PUV[i] is set to TRUE. If all of the elements in row i of the PUM are true, the corresponding element in the PUV is set to TRUE, i.e., if PUM[i,j] is TRUE for j = 1 to 15, then PUV[i] is TRUE.

LAUNCH Generation

The LAUNCH signal is generated from the PUV. LAUNCH is set to TRUE if all elements of the PUV are TRUE, otherwise it is FALSE, i.e., if PUV[i] is TRUE for i = 1 to 15, then LAUNCH is TRUE.

APPENDIX B

LISTING OF TEST CASES (SET NO. 1)

A Listing of the 54 test cases for test set number 1 is given in this Appendix. The listed values represent the input data as explained below.

Data Items

1 Test case number

2 NUMPOINTS

3 LENGTH1, RADIUS1, EPSILON, AREA1

4 QPTS , QUADS

5 DIST

6 N K A B C D E F G (PTS)

7 LENGTH2, RADIUS2, AREA2

8 25 Elements of the LCM: given here are the elements of the upper left hand 5x5 submatrix. Other values are not used.

(0 - not used,1 -ORR, 2- ANDD)

9 Diagonal elements of PUM: 0 -FALSE, 1- TRUE

10 Starting with line 10 are the values of (x,y) pairs;one pair on each line.

```
 1
 6
       6.10         7.43         2.40         1.90
 4  2
       2.90
 3  3  2  2  2  1  2  2  1
       7.30         2.80         0.10
2 2 0 1 1 1 0 2 1 0 0 1 2 1 0 2 0 0 1 0 1 2 0 1 2
0 1 0 0 1 1 0 1 1 0 1 0 1 0 1
     -2.900       -3.700
     -8.300        1.600
      1.200        4.300
     -4.700        0.300
      5.200       -3.200
      1.100        3.100
 2
 6
      12.40         0.30         1.30         0.70
 3  2
       3.80
 3  3  1  1  3  2  1  2  1
       6.40         1.20         7.40
1 0 1 2 0 1 0 2 0 1 0 1 0 2 0 1 0 1 0 1 0 2 0 1 0
0 1 0 1 0 1 0 1 0 1 0 0 0 1 0
     -2.900       -3.700
     -8.300        1.600
      1.200        4.300
     -4.700        0.300
      5.200       -3.200
      1.100        3.100
 3
 6
      16.70         4.70         2.50         9.30
 3  1
       1.50
 3  3  1  1  2  1  2  1  2
      10.10         0.30         1.40
0 1 0 2 0 1 1 0 1 2 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1
1 0 1 1 0 1 0 1 1 0 0 1 0 1 0
     -2.900       -3.700
     -8.300        1.600
      1.200        4.300
     -4.700        0.300
      5.200       -3.200
      1.100        3.100
```

```
4
6
      4.50         2.70         0.20         1.60
4  3
      1.10
3  3  1  1  2  1  2  1  2
      0.50         3.50         2.10
0 1 0 2 0 1 1 0 1 2 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1
1 0 1 1 0 1 0 1 1 0 0 1 0 1 0
     -2.900       -3.700
     -8.300        1.600
      1.200        4.300
     -4.700        0.300
      5.200       -3.200
      1.100        3.100
5
8
      3.20         0.30         1.70         4.30
4  3
      8.90
3  3  2  2  2  1  2  2  1
     12.40         2.80         0.10
2 2 0 1 1 1 0 2 1 0 0 1 2 0 1 1 2 0 1 0 0 0 0 0 0
0 0 1 0 1 0 0 1 1 1 1 1 0 1 0
     -0.300        2.400
      9.800        1.600
      7.200       -6.400
     -3.400        1.700
      2.600        6.400
     -3.400        2.100
      0.900       -7.300
     -2.900       -3.700
6
8
     14.70         2.10         0.40         1.60
4  2
      1.80
3  3  2  2  2  1  2  2  1
      3.10         6.40         9.20
1 0 0 2 1 0 1 0 1 2 1 0 1 0 1 0 1 0 1 0 1 2 1 1 0
1 0 1 0 1 1 0 1 0 0 0 1 1 0 1
     -0.300        2.400
      9.800        1.600
      7.200       -6.400
     -3.400        1.700
```

```
      2.600        6.400
     -3.400        2.100
      0.900       -7.300
     -2.900       -3.700
 7
 8
       5.60         1.60         1.20         2.80
 4  3
       2.80
 3  3  2  2  2  1  2  2  1
       2.30         1.30         0.20
1 0 0 2 1 0 1 0 1 2 1 0 1 0 1 0 1 0 1 0 1 2 1 1 0
1 0 1 0 1 1 0 1 0 0 0 1 1 0 1
     -0.300        2.400
      9.800        1.600
      7.200       -6.400
     -3.400        1.700
      2.600        6.400
     -3.400        2.100
      0.900       -7.300
     -2.900       -3.700
 8
 5
       7.30         6.40         0.50         2.70
 4  2
       8.90
 3  3  2  2  2  1  2  2  1
       2.70         1.20         5.50
2 2 0 1 1 1 0 2 1 0 0 1 2 0 1 1 2 0 1 0 0 0 0 0 0
0 0 1 0 1 0 0 1 1 1 1 1 0 1 0
      1.300        0.400
     -3.800        3.500
      8.400       -5.200
      0.400        1.600
      0.500       -4.800
 9
 5
       0.90         2.10         0.50         8.10
 3  1
       1.80
 3  3  2  2  1  1  3  1  3
       5.30         0.40         1.20
0 1 0 2 0 1 1 0 1 1 1 0 0 1 2 0 1 0 2 1 0 1 2 1 2
0 1 0 1 0 1 1 0 0 1 1 0 1 0 1
      1.300        0.400
```

```
      -3.800          3.500
       8.400         -5.200
       0.400          1.600
       0.500         -4.800
10
 5
         1.20           7.30           2.40           0.10
 2  1
         2.80
 3  3  2  2  1  1  3  1  3
         0.20           3.20           7.50
1 2 0 1 0 1 0 0 1 0 1 0 1 0 2 2 0 1 1 0 0 1 2 1 2
1 1 0 1 1 0 1 1 1 0 1 0 1 0 1
        1.300          0.400
       -3.800          3.500
        8.400         -5.200
        0.400          1.600
        0.500         -4.800
11
 5
        17.30           0.40           1.20           5.40
 2  1
         7.50
 3  3  2  2  1  1  3  1  3
         3.60           2.10           5.40
1 2 0 1 0 1 0 0 1 0 1 0 1 0 2 2 0 1 1 0 0 1 2 1 2
1 1 0 1 1 0 1 1 1 0 1 0 1 0 1
        1.300          0.400
       -3.800          3.500
        8.400         -5.200
        0.400          1.600
        0.500         -4.800
12
 8
         3.10           2.90           0.10           0.30
 4  1
         0.10
 3  3  2  2  2  1  2  2  1
         1.90           1.60           0.10
0 1 0 1 0 2 1 0 1 2 1 0 1 0 1 0 1 2 0 1 0 2 1 1 0
1 0 0 1 0 1 0 0 1 0 0 1 1 0 1
        0.200          1.200
       -0.300          2.100
       -0.100          4.100
        5.200         -3.200
```

```
      -1.100        -3.200
       0.600         6.500
       1.100         3.100
       4.400         5.400
13
 8
         2.90          1.40          2.80          0.60
 2  3
         2.30
 3  3  2  2  1  2  1  1  2
         2.70          3.10          6.70
1 0 1 1 2 0 1 0 1 0 1 2 0 1 0 2 0 2 1 0 0 2 1 2 1
0 1 0 1 1 1 0 1 0 1 0 1 0 1 1
       0.200         1.200
      -0.300         2.100
      -0.100         4.100
       5.200        -3.200
      -1.100        -3.200
       0.600         6.500
       1.100         3.100
       4.400         5.400
14
 8
         4.80          6.30          0.60          1.20
 2  2
         3.90
 3  3  2  2  2  1  3  1  3
         1.10          0.70          0.30
0 1 2 0 1 1 0 1 0 1 0 0 1 0 0 1 0 1 0 2 0 2 1 2 1
0 1 0 0 0 1 1 0 1 0 1 0 1 0 1
       0.200         1.200
      -0.300         2.100
      -0.100         4.100
       5.200        -3.200
      -1.100        -3.200
       0.600         6.500
       1.100         3.100
       4.400         5.400
15
 5
         2.90          3.20          1.20          9.30
 4  1
         8.10
 3  3  1  1  2  1  2  2  1
         2.10          5.30          6.90
```

```
1-0 2 1 2 2 1 0 1 0 0 1 2 0 1 1 2 0 1 0 0 0 0 0 0
1 0 0 1 0 1 0 0 0 1 1 1 0 1 0
      0.300        9.000
      2.300        0.300
      4.400        2.100
      9.900        2.200
     -3.900        1.200
16
 5
       0.80         4.20         1.40         1.10
 2  3
       3.90
 3  2  1  1  1  2  1  2  3
       1.00         3.10         0.40
0 1 0 1 0 1 0 1 2 1 2 1 0 1 0 1 0 1 2 0 1 2 1 2 1
0 1 0 1 1 0 1 0 1 0 1 1 0 1 1
      0.300        9.000
      2.300        0.300
      4.400        2.100
      9.900        2.200
     -3.900        1.200
17
 5
       3.80         4.35         1.50         0.80
 3  2
       2.80
 3  2  1  1  2  1  2  1  2
       0.20         6.20         2.10
0 1 1 0 2 0 1 2 0 1 0 1 2 0 1 2 0 1 0 1 0 1 2 1 0
0 0 1 1 0 1 0 0 1 0 1 0 1 0 1
      0.300        9.000
      2.300        0.300
      4.400        2.100
      9.900        2.200
     -3.900        1.200
18
 7
       0.20         5.30         2.10         1.20
 4  1
       3.30
 3  3  2  1  1  2  1  1  2
       2.00         3.70         5.30
1 0 2 1 2 2 1 0 1 0 0 1 2 0 1 1 2 0 1 0 0 0 0 0 0
1 0 0 1 0 1 0 0 0 1 1 1 0 1 0
      3.900       -3.300
```

```
     -4.400       -2.200
     -4.300        1.100
      7.300        2.200
     -3.800        1.100
     -1.200        0.300
     -3.900        1.200
19
 7
        1.30         2.60         0.30         4.50
 2  3
        2.20
 3  2  2  1  2  1  2  2  1
        1.40         4.50         3.80
0 1 0 0 1 0 2 0 2 1 2 2 1 0 1 2 2 2 0 1 2 1 2 1 0
1 0 1 0 1 0 1 1 0 1 0 1 1 1 1
       3.900       -3.300
     -4.400       -2.200
     -4.300        1.100
      7.300        2.200
     -3.800        1.100
     -1.200        0.300
     -3.900        1.200
20
 7
        0.40         7.80         1.32         3.90
 3  2
        1.30
 3  1  2  1  1  3  1  3  1
        0.30         0.90         0.20
0 1 0 0 1 2 0 1 2 0 1 0 1 0 1 2 0 2 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1 0 0 1 0 0
       3.900       -3.300
     -4.400       -2.200
     -4.300        1.100
      7.300        2.200
     -3.800        1.100
     -1.200        0.300
     -3.900        1.200
21
 7
        3.20         4.30         0.20         1.40
 3  3
        0.40
 3  1  2  1  1  3  2  2  2
        1.10         2.30         1.20
```

```
1 0 1 2 0 1 0 2 1 0 1 1 2 1 0 2 0 1 0 2 1 2 0 1 1
0 1 0 1 0 0 1 0 1 0 1 0 1 1 1
      3.900       -3.300
     -4.400       -2.200
     -4.300        1.100
      7.300        2.200
     -3.800        1.100
     -1.200        0.300
     -3.900        1.200
22
 7
       1.00         6.40         1.70         2.70
 4  2
       1.30
 3  3  2  2  2  1  1  1  1
       0.40         1.20         5.50
2 2 0 1 1 1 0 2 1 0 0 1 2 0 1 1 2 0 1 0 0 0 0 0 0
0 0 1 0 1 0 0 1 1 1 1 1 0 1 0
      0.300       -4.300
     -5.200       -7.500
      0.400        1.800
      3.700        3.400
      0.800        3.700
      9.800        6.400
      2.700       -5.400
23
 7
       3.40         1.50         2.90         4.30
 4  3
       9.30
 3  3  1  1  2  1  2  1  2
       1.10         5.40         1.30
0 1 0 2 1 0 1 0 2 1 2 0 1 0 0 2 2 2 0 1 0 2 1 2 1
1 0 1 1 1 1 0 1 0 0 1 1 0 1 1
      0.300       -4.300
     -5.200       -7.500
      0.400        1.800
      3.700        3.400
      0.800        3.700
      9.800        6.400
      2.700       -5.400
24
 5
       6.40         1.90         2.87         0.20
 2  1
```

```
        2.05
 3   3   1   2   1   1   1   2   2
        1.90          0.30          9.40
2 2 0 1 1 1 0 2 1 0 0 1 2 0 0 1 1 2 0 1 1 2 0 1 1
0 1 1 1 0 1 1 0 1 0 1 0 1 1 0
       6.300         0.200
       0.400        -3.200
       0.900         2.400
      -3.700        -1.500
       9.800         4.300
25
 5
        3.50          3.70          2.80          2.54
 2   2
        4.39
 3   3   1   2   1   1   1   2   2
       10.30          5.30          7.20
1 1 1 0 1 2 0 1 0 1 0 1 0 2 1 1 1 0 1 0 1 1 1 0 1
0 1 1 1 0 1 1 0 1 0 1 0 1 1 0
       6.300         0.200
       0.400        -3.200
       0.900         2.400
      -3.700        -1.500
       9.800         4.300
26
 5
       12.90          3.90          1.40         11.90
 2   1
       13.90
 3   3   1   2   1   1   2   1   2
       19.80          1.90          6.50
1 1 1 0 1 2 0 1 0 1 0 1 0 2 1 1 1 0 1 0 1 1 1 0 1
0 1 1 1 0 1 1 0 1 0 1 0 1 1 0
       6.300         0.200
       0.400        -3.200
       0.900         2.400
      -3.700        -1.500
       9.800         4.300
27
 9
       12.10          9.30          1.30          9.40
 5   3
       19.30
 3   3   2   2   3   2   2   2   2
        1.80          5.40          7.60
```

```
1-2 0 1 0 1 0 1 2 2 0 1 1 0 1 1 0 1 0 1 0 1 2 1 2
1 0 1 0 1 0 1 0 1 1 0 1 0 1 1
      0.400        1.200
     -4.900       -3.700
     -0.200       -1.100
      3.400        5.200
      5.400        1.200
     -3.900        9.400
     -8.400       -2.100
      4.600        0.100
     -5.200       -7.500
28
 9
       2.90         1.60         1.80        14.70
 5  3
       8.20
 3  3  1  1  3  2  1  1  1
      12.80         2.70        10.30
2 2 0 1 1 1 0 2 1 0 0 1 2 0 1 1 1 0 1 2 0 1 0 1 2
1 0 1 0 1 1 1 0 0 1 0 1 1 0 1
      0.400        1.200
     -4.900       -3.700
     -0.200       -1.100
      3.400        5.200
      5.400        1.200
     -3.900        9.400
     -8.400       -2.100
      4.600        0.100
     -5.200       -7.500
29
 9
      10.80         2.10         1.20         5.10
 6  2
       1.80
 3  3  1  1  3  2  1  1  1
       1.70         0.20         1.20
2 2 0 1 1 1 0 2 1 0 0 1 2 0 1 1 1 0 1 2 0 1 0 1 2
1 0 1 0 1 1 1 0 0 1 0 1 1 0 1
      0.400        1.200
     -4.900       -3.700
     -0.200       -1.100
      3.400        5.200
      5.400        1.200
     -3.900        9.400
     -8.400       -2.100
```

```
      4.600        0.100
     -5.200       -7.500
30
 9
       6.80         1.30         0.40        80.50
 6  1
       9.10
 3  3  1  1  3  2  1  1  1
       5.90         1.30         7.50
2 2 0 1 1 1 0 2 1 0 0 1 2 1 0 1 0 1 2 0 1 0 1 0 1
1 0 1 0 1 1 1 1 0 1 1 0 0 1 0
      0.400        1.200
     -4.900       -3.700
     -0.200       -1.100
      3.400        5.200
      5.400        1.200
     -3.900        9.400
     -8.400       -2.100
      4.600        0.100
     -5.200       -7.500
31
 9
       1.20         0.30         1.80        57.30
 5  2
       1.60
 3  3  1  2  3  2  1  1  3
       1.30         0.30         2.70
0 1 0 2 1 0 1 0 1 0 1 0 2 1 0 1 2 0 1 0 2 1 1 2 0
0 1 0 1 1 0 1 0 1 1 0 1 1 0 1
      0.400        1.200
     -4.900       -3.700
     -0.200       -1.100
      3.400        5.200
      5.400        1.200
     -3.900        9.400
     -8.400       -2.100
      4.600        0.100
     -5.200       -7.500
32
 6
      12.90         0.10         1.40        11.70
 3  2
      11.02
 3  3  1  3  1  2  2  1  1
       1.00         0.30         3.40
```

```
1 0 1 0 1 0 1 0 1 2 2 0 1 0 1 2 0 2 0 2 0 2 1 1 2
1 0 1 1 0 1 0 1 0 1 0 1 0 0 1
      2.800       -4.300
     -8.900       -3.800
      0.100       -0.300
      3.900        2.500
     -9.400       -1.800
      0.900        4.300
33
 6
       1.90         3.50         1.30        10.90
 3  3
       1.09
 3  3  2  1  3  2  2  1  1
       8.40         4.50        24.10
1 2 0 1 0 1 0 1 0 1 0 2 1 0 1 0 1 1 2 0 1 2 0 0 2
1 1 0 1 1 0 1 0 0 1 0 1 0 0 1
      2.800       -4.300
     -8.900       -3.800
      0.100       -0.300
      3.900        2.500
     -9.400       -1.800
      0.900        4.300
34
 6
       0.20         4.23         0.30         7.60
 3  1
       0.39
 3  3  2  1  3  2  2  1  1
       1.02        11.40         6.20
0 1 0 1 0 1 0 1 0 1 0 2 1 0 1 0 1 1 1 0 1 1 2 2 0
0 1 0 1 1 1 0 1 1 0 0 1 1 1 0
      2.800       -4.300
     -8.900       -3.800
      0.100       -0.300
      3.900        2.500
     -9.400       -1.800
      0.900        4.300
35
 6
       12.90         0.30         1.09        30.70
 4  2
       12.85
 3  3  2  1  3  2  2  1  1
       10.80         0.30         2.10
```

```
1-0 1 0 1 2 1 2 1 2 0 1 1 1 0 2 2 0 1 0 1 2 0 2 1
1 0 1 1 1 0 1 0 1 0 0 1 1 1 0
       2.800       -4.300
      -8.900       -3.800
       0.100       -0.300
       3.900        2.500
      -9.400       -1.800
       0.900        4.300
36
 8
        0.20         5.40         1.30        10.50
 4  2
        1.10
 3  3  2  2  3  2  1  3  1
       10.50         0.40        45.70
1 0 1 0 1 1 0 2 1 0 0 1 2 0 1 1 2 0 1 2 0 1 2 1 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
       9.900        2.800
      -0.400       -3.800
       2.900       -2.200
       0.230       -5.500
       9.300       -2.100
      -0.400        1.300
      -4.900        3.800
       5.700       -1.000
37
 8
       11.90         0.30         1.80         4.60
 5  3
       12.80
 3  3  2  2  3  2  1  3  1
       15.76         2.70         2.50
1 0 1 1 0 1 0 2 1 0 0 1 2 0 1 1 2 0 1 0 1 0 2 1 0
0 1 0 1 0 1 0 1 1 0 1 0 1 0 1
      9.900        2.800
     -0.400       -3.800
      2.900       -2.200
      0.230       -5.500
      9.300       -2.100
     -0.400        1.300
     -4.900        3.800
      5.700       -1.000
38
 8
       0.90        11.90         1.70        33.10
```

```
 4  2
       10.80
 3  3  2  2  3  2  1  3  1
        1.09          5.40          51.40
1 0 1 1 0 1 0 2 1 0 0 1 2 0 1 1 2 0 1 0 1 0 2 1 0
0 1 0 1 0 1 0 1 1 0 1 0 1 0 1
       9.900         2.800
      -0.400        -3.800
       2.900        -2.200
       0.230        -5.500
       9.300        -2.100
      -0.400         1.300
      -4.900         3.800
       5.700        -1.000
39
 8
       12.90          0.30          1.11          38.50
 6  3
        0.98
 3  3  2  3  2  2  1  3  2
       11.90         12.70          4.10
1 0 1 1 0 1 0 2 1 0 0 1 2 0 1 1 2 0 1 0 1 0 2 1 0
0 1 0 1 0 1 0 1 1 0 1 0 1 0 1
       9.900         2.800
      -0.400        -3.800
       2.900        -2.200
       0.230        -5.500
       9.300        -2.100
      -0.400         1.300
      -4.900         3.800
       5.700        -1.000
40
 8
        1.00          4.50          2.89          42.70
 4  2
        0.43
 3  3  3  2  4  2  1  3  1
        0.90         11.50          3.90
0 1 0 2 1 2 1 2 1 1 2 1 1 2 1 0 1 0 2 2 1 0 1 1 2
1 0 1 0 1 0 1 0 1 0 1 1 1 1 0
       9.900         2.800
      -0.400        -3.800
       2.900        -2.200
       0.230        -5.500
       9.300        -2.100
```

```
 -    -0.400        1.300
      -4.900        3.800
       5.700       -1.000
41
 8
       56.40         1.90         0.20        17.30
 4  3
       12.90
 3  3  2  2  1  1  1  3  2
       11.40         0.30        43.70
0 1 0 2 1 2 1 2 1 1 2 1 1 2 1 0 1 0 2 2 1 0 1 1 2
1 0 1 0 1 0 1 0 1 0 1 1 1 1 0
       9.900        2.800
      -0.400       -3.800
       2.900       -2.200
       0.230       -5.500
       9.300       -2.100
      -0.400        1.300
      -4.900        3.800
       5.700       -1.000
42
 9
       12.90         6.40         0.90         2.70
 5  1
       10.20
 3  3  2  2  1  1  3  2  2
       11.80         0.40         5.50
2 2 0 1 1 1 0 2 1 0 0 1 2 0 1 1 2 0 0 1 2 1 2 1 1
0 0 1 0 1 1 0 1 0 1 0 1 1 1 0
       4.300       -2.300
       3.900        1.700
       4.300       -0.800
       0.200        1.200
       9.800        0.200
      -3.800       -3.700
      -0.900        2.100
       9.300       -2.100
       0.500        2.600
43
 9
        1.90         0.30         1.50         0.10
 5  2
       23.90
 3  3  3  2  2  2  1  3  1
       10.30         1.30         0.90
```

```
0-1 2 2 0 1 0 1 0 1 0 2 1 1 0 1 0 2 1 0 1 0 1 0 1
1 0 1 1 0 1 0 1 1 1 1 0 0 1 1
       4.300        -2.300
       3.900         1.700
       4.300        -0.800
       0.200         1.200
       9.800         0.200
      -3.800        -3.700
      -0.900         2.100
       9.300        -2.100
       0.500         2.600
44
 4
       12.80          0.30          1.91          1.38
 2  1
       11.90
 3  3  2  1  2  3  1  2  2
       23.90          0.30         12.03
1 1 1 0 2 2 1 0 1 1 1 1 0 2 2 1 0 1 1 1 1 2 0 2 1
1 0 1 1 0 1 1 0 1 1 0 1 0 1 0
       3.900        -2.100
      -9.700         1.200
       7.400         2.400
      -0.800        -3.200
45
 4
        0.20          4.90          2.90          0.30
 2  3
       11.30
 3  3  2  2  3  2  1  3  1
       12.01          0.30          5.50
2 2 0 1 1 1 0 2 1 0 0 1 2 0 1 1 1 0 1 2 1 0 1 1 2
1 0 1 1 0 1 1 0 1 0 1 0 1 0 1
       3.900        -2.100
      -9.700         1.200
       7.400         2.400
      -0.800        -3.200
46
 7
       10.30          0.40          0.40          0.40
 3  2
       23.90
 3  3  3  1  1  1  2  1  2
       11.03          0.40          0.60
2 2 0 1 1 1 0 2 1 0 0 1 2 0 0 1 0 1 2 1 1 1 2 2 1
```

```
1 0 1 1 0 1 1 0 1 0 1 0 1 0 1
      2.100        0.800
     -0.200        2.700
      0.100        1.600
      0.400       -3.200
     -8.300        3.700
      1.100       -0.800
     -0.900       -7.400
47
 7
      11.00         6.30         1.30         0.30
 3  3
      10.20
 3  3  2  2  1  1  1  1  2
      18.60         0.40         3.20
1 0 1 0 2 1 1 0 1 1 1 0 1 2 2 2 1 1 2 1 1 2 1 2 1
1 0 1 0 1 1 1 0 1 0 1 0 1 0 1
      2.100        0.800
     -0.200        2.700
      0.100        1.600
      0.400       -3.200
     -8.300        3.700
      1.100       -0.800
     -0.900       -7.400
48
 6
       0.30         0.40         1.01         9.30
 4  1
      11.80
 3  1  1  1  2  1  2  3  1
      13.20         4.30         2.10
2 2 1 2 0 2 2 1 1 0 0 1 2 0 1 1 2 0 1 1 2 1 2 1 1
1 0 1 1 1 0 1 1 1 1 0 1 1 1 1
    -40.200        0.340
    -38.900        2.930
    -12.800       -2.400
      1.260        3.450
      3.400       -2.400
     13.400        2.340
49
 5
      13.80         0.30         1.30         0.40
 3  2
       3.10
 3  3  2  2  3  2  1  3  1
```

```
      11.02         6.70         4.30
1 0 1 1 0 1 0 1 2 1 1 2 1 0 1 1 1 1 0 1 2 1 2 2 0
1 0 0 1 0 1 1 0 1 1 1 0 1 1 1
      3.800        0.300
     -4.700        1.300
      0.900       -3.400
      5.770       -6.650
      4.500       -8.700
50
 5
       1.90         0.30         2.10        12.90
 4  3
      10.20
 3  3  1  1  2  1  1  2  2
      11.92         3.50         0.30
1 2 0 1 0 1 1 0 2 2 2 1 0 1 1 2 2 1 2 1 2 1 1 1 2
0 1 1 0 1 1 1 0 0 1 0 1 0 1 0
      3.800        0.300
     -4.700        1.300
      0.900       -3.400
      5.770       -6.650
      4.500       -8.700
51
 8
       2.40         0.40         0.20         0.10
 4  2
       7.80
 4  3  2  2  3  2  1  3  1
       2.39         0.30        10.30
1 0 1 2 0 1 0 1 0 2 2 1 2 0 1 2 2 2 2 1 0 1 1 1 0
1 0 1 1 1 1 0 1 0 1 0 1 1 0 1
      4.300       -5.300
      0.400       -4.460
     -5.387        3.420
     -3.520        3.030
      0.320       -4.550
      3.800        0.300
      0.900       -3.400
      4.500       -8.700
52
 8
       0.30         7.30         1.32         0.30
 5  3
      12.40
 4  3  1  1  2  1  2  1  2
```

```
       0.20         0.60        10.30
1 0 1 2 1 1 1 1 2 2 2 2 1 1 1 0 1 1 2 2 1 0 1 1 2
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
      4.300       -5.300
      0.400       -4.460
     -5.387        3.420
     -3.520        3.030
      0.320       -4.550
      3.800        0.300
      0.900       -3.400
      4.500       -8.700
53
 6
       2.30         0.40         1.40         0.70
 3  1
       7.80
 5  3  1  1  2  2  3  1  2
      10.90         3.70         0.40
1 2 0 1 2 2 1 0 1 1 1 0 1 0 2 2 2 0 1 2 2 2 1 0 1
1 1 0 1 1 0 1 0 1 0 1 0 1 1 0
      9.600        6.300
     -0.400        7.400
      0.530       -5.800
     -5.600        4.100
     -9.400       -5.800
      0.300       -4.600
54
 6
      12.90         5.40         1.30         6.50
 3  3
      18.20
 5  3  1  1  2  3  2  1  2
       1.90        12.20         0.30
1 0 1 0 2 1 1 2 0 1 2 2 1 0 1 1 2 2 2 1 0 2 1 2 1
1 0 1 1 1 1 1 0 1 1 0 1 0 1 1
      9.600        6.300
     -0.4         7.400
      0.530       -5.800
     -5.600        4.100
     -9.400       -5.800
      0.300       -4.600
```

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.