NASA CR-172,553

NASA Contractor Report 172553

NASA-CR-172553
19860020075

AN EXPERIMENT IN SOFTWARE RELIABILITY

Janet R. Dunham
John L. Pierce

Software Research and Development
Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

Contract NAS1-16489
Task Assignment Nos. 12 and 14
March 1985

Revised May 1986

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

NASA Contractor Report 172553

AN EXPERIMENT IN SOFTWARE RELIABILITY

Janet R. Dunham
John L. Pierce

Software Research and Development
Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

Contract NAS1-16489
Task Assignment Nos. 12 and 14

March 1985

Revised May 1986

N86-29547 #

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF PLOTS

APPENDICES

ACKNOWLEDGMENT

The authors acknowledge the technical guidance of Earle Migneault of NASA - Langley Research Center during this investigation. We also thank Phyllis Nagel and Jim Skrivan of Boeing whose report spurred our initial interest in this area and to Bev Littlewood of City University of London, Larry Lee of NASA-LaRC, and Doug Miller of George Washington University for their advice on statistical issues. Our appreciation is also extended to the project participants, J. Dunn, Bill Ingogly, Barry Koster, Bill Hargrove, Paul Morris, and Sam McBride for their hard work and determination. Finally, we thank Brian Lupton of the Flight Controls Systems Branch of NASA-LaRC who was the contract monitor for this project and our manager Jim Clary for his unfailing enthusiasm about our work.

Janet R. Dunham
March 1985

1. INTRODUCTION AND OVERVIEW

1.1. Background

Digital computers are having an increasingly important role in process control applications, particularly those in which human life may be endangered, such as radar tracking, flight-control, and medical life-support systems. Reliability requirements for these systems are very stringent as system failure may incur extensive costs. For example, NASA-Langley Research Center (NASA-LaRC) has used a requirement of a system probability of failure of 10^{-9} for a ten hour flight as a working figure. [1]

Software is an integral component of these crucial digital systems, yet we presently lack the appropriate methods for ensuring that the software is meeting the reliability and performance requirements specified. Acknowledgement of the insufficiency of development methods for avoiding and removing faults in software has led to the proposal of different designs for tolerating them. As advocated by Avizieniz [2] and further discussed by Anderson and Lee [3], one software fault-tolerant design is n-version programming. Empirical evaluations of the reliability of systems implemented using this and other software fault-tolerant designs for realistic applications are needed to determine their potential usefulness.

This document reports the goals, design, and results of a research project evaluating the reliability of software modules. The research involved the independent development of three versions of a module for a radar tracking application and the execution of these versions in an n-version system using inputs which simulate the operational environment for the application. First, a brief history of the project is given, and the reasons for its inception are discussed. This discussion highlights the differences between the desired scope of the project and the scope which could be realistically implemented. Second, elements of the project are described: the emulation of a realistic software engineering environment, the *a priori* definition of the data to be collected and analyzed, the process used for selecting the participants, the development of data collection and evaluation tools and collection of the data. Third, the results of the data analysis are presented. An appendix contains the failure data for each of the three software modules under test.

1.2. Project Goals

The primary goal was to gather more information on the failure processes for software modules. The information collected consists of development and failure data for the component modules. This information will be used to aid in the evaluation of software reliability modeling techniques. A secondary goal was to independently verify a portion of the research performed by Nagel and Skrivan of Boeing Corporation [4] by using their repetitive run modeling approach but a different detection mechanism (i.e., n-version programming).

This study is part of a long term objective to determine procedures for evaluating the reliability of flight control systems. Hence, this study also includes a pitch axis control problem not reported herein. The pitch axis control problem was chosen to investigate the phenomena of error bursts [5] for a real-time system with a feed-back loop.

1.3. Summary of Study and Conclusions

This study in software reliability was designed to emulate the software development process. It was performed in a laboratory environment using the remote computing facilities of AIRLAB at NASA-Langley Research Center. The experiment can be viewed as a case study which is being performed in a controlled laboratory setting.

As a part of the study, three moderate skill level programmers were given two problems to code and debug. The problem requirements were written in English, and the programmers worked independently to code these problems in FORTRAN IV. The programmers were permitted to compile and execute their programs freely prior to providing them for acceptance testing. A user was available to answer programmer questions. When a program for a given problem was completed, it was submitted to a series of acceptance tests. Acceptance testing involved executing the program with a pre-defined set of input data and comparing the program output with the pre-defined correct output. Acceptance test failures (i.e., discrepancies between program outputs and pre-defined outputs) were noted, and the program returned to the programmer for additional debugging. After all acceptance tests were passed by all programs for a given problem, the programs were executed in an N-version system with inputs randomly generated from a specified probability distribution on the input space. Program outputs were voted on to determine output correctness. Replicated trials were obtained using the repetitive run modeling approach described in 2.4.

Summary of Observations

The results obtained agree with those found in the Boeing study. The sequence of program error rates followed a log-linear pattern, and the individual "bugs" occurred with significantly different rates. Additional data is needed to investigate the impact of dependencies among faults.

Recommendations

A sequence of smaller scale experiments coordinated among a set of collaborating researchers appears to be the preferred mode of research. This preference is a result of the extensive cost of collecting software failure data for reliability modeling and the nature of unanticipated problems which occur during the conduct of this type of experiment. Experiment data integrity requires a much greater extent of automation than one would anticipate. Finally, while it would not be sensible to advocate more complex statistical techniques where simpler ones suffice, it appears that software reliability assessment demands computationally intensive techniques and that simpler ones will not suffice.

2. PROJECT DESIGN

2.1. History Provides Perspective

Knowledge about the design history of a project provides perspective as it contains information about project development. For example, an important design decision made for this project resulted in the use of three advanced skill level programmers for a period of three months instead of using six moderate skill level programmers for one and a half months. The ability to recruit and manage three programmers for a summer project determined this design. Unfortunately, the accompanying reduction in sample size precluded performing regression analyses aimed at correlating program failure data with other programmer, development, and program characteristic data.

The paradigms used in designing this research project are analogous to those recommended by F.P. Brooks for software design [6] and those recommended by Basili and Weiss for software engineering data collection. [7] More specifically, the project design was considered a prototype design with the inevitability of change planned for and the need for automated tools recognized at an early stage. In addition, a great deal of emphasis was placed on carefully defining the experimental goals and determining guidelines for weighing design decisions. This meticulousness, although burdensome at times, in our opinion, resulted in fewer difficulties during implementation. Design decision guidelines include keeping the investigation small scale and exploratory, defining the data analysis methodology in advance, trying to collect as much data as possible without jeopardizing the goals of the project, supporting other work done in this field, and not constraining the initial project design by implementation decisions.

In attempting to design the project to collect as much potentially useful data as possible, an initial list of the data elements to be recorded was drawn up. This layout resulted in a logical record length of 10,000 bytes per input case for the radar tracking problem. A quick computation estimated the storage requirements at 10,000 bytes per input case. Executing an average of 200,000 input cases per replication for 50 replications would require $1x10^{11}$ bytes of storage space per problem. This space requirement resulted in a decision to record only input cases for which an application task or system failure occurred. In other words, the decision to delete some potentially important dynamic data for each input case (e.g., inputs generated, application task outputs, and system outputs) was made only after much consideration was given to the desired data elements. Other implementation decisions were made with respect to the project goals, design guidelines, and impact upon previous design decisions. Approximately 2.5 person months of effort were spent formulating the project design and 1 person year of effort was spent preparing for the experiment (e.g., selecting the programmers and developing the necessary tools and forms).

2.2. Transitioning from a 2^3 Factorial Experiment to a Case Study

The foremost design decision was that the experiment be performed in a controlled environment which emulated the production environment a software engineer would face if he or she were developing real-time software for an n-version system. This decision was made in an attempt to counter the criticisms surrounding the lack of "realism" in software engineering experiments [8] and to thwart the influence of certain uncontrollable and immeasurable exogenous factors in actual development environments.

A 2^3 factorial experimental design was proposed in which programmer aptitude and problem difficulty were chosen as the two factors to be varied. Programmer skill was to be grouped into basic, intermediate, and advanced categories; the problems to be coded were based on three levels of difficulty. However, the sheer cost of obtaining a sufficient sample size for the 2^3 factorial design resulted in the selection of three programmers of an advanced skill level and their implementation of two problems. This reduction in scope did not obviate the already carefully thought out design decisions. For example, programmer selection considered the criteria reported by Moher and Schneider. [9] A form was designed to screen applicants (see Figure 1), and those considered were exposed to a series of rigorous interviews by the project staff. Although students were employed, they had previous programming experience in full-time positions and were enrolled in Master of Science degree programs offered by accredited Computer Science Departments. A competitive salary was paid to attract the best qualified programmers. The students were hired as programmer/analysts as part of a summer internship program.

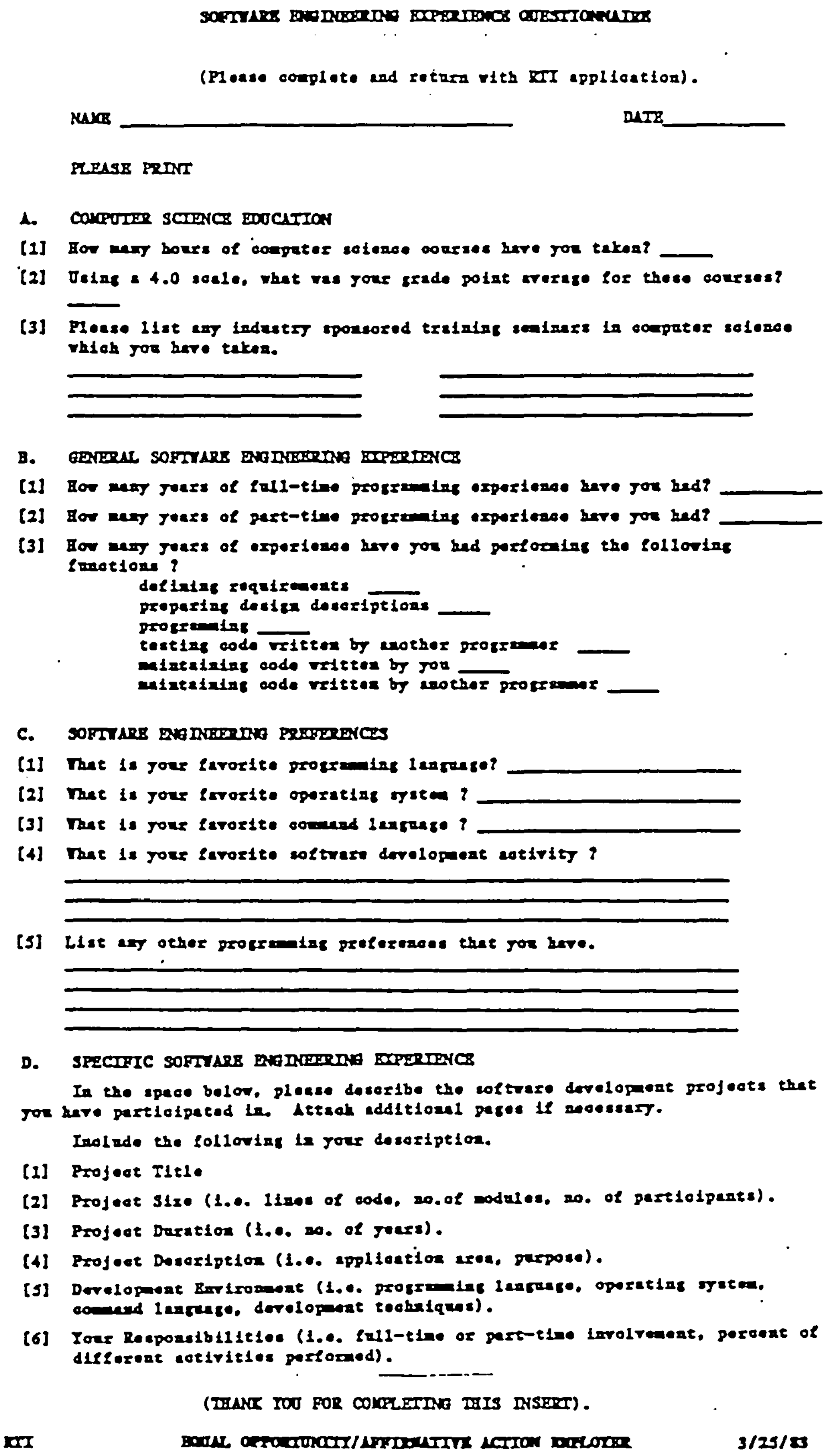
The reduction in scope from a 2^3 factorial experiment to a research project which employed three moderate skill level programmers did change the project from a formal, statistically designed experiment to a controlled laboratory investigation. Essentially, the external validity of inferences based on these results would be even more suspect than those made using the 2^3 factorial design. Increasing the sample size by repeating the experiment, however, would permit one to make inferences about other projects. It is important to note that most theories should be verified by multiple corroboration, empirical repetition, and independent validation. Statistical significance is not the only important attribute of a good experiment.

2.3. N-Version Simulation

After the programmers had completed a given problem, their programs or application tasks (ATs) were executed in parallel using n-version simulation for error detection. During n-version simulation, an input case was generated, and n-version outputs were selected using three different decision rules. The AT outputs and the n-version outputs were then compared with the output of an extensively tested version coded by a senior analyst. In addition, the output for each of the ATs was pairwise compared. When an AT output discrepancy was noted, the n-version simulation was halted, the discrepancy analyzed, and an electronic mail message describing the input case and the error was sent to the subject programmer. This process was repeated a number of times using the Repetitive Run Modeling Approach.

2.4. Repetitive Run Modeling

The Repetitive Run Modeling Approach was first advocated by Nagel and Skrivan of Boeing Computer Services. [4] This approach was developed to provide information about the probabilistic impact of detected software faults on subsequent fault detection. It involves repetitively executing a software module from an initial state or design stage through to the detection and correction of n faults. Repetitive Run Modeling provides "better" estimates of the individual error rates. On subsequent runs or replications, the


SOFTWARE ENGINEERING EXPERIENCE QUESTIONNAIRE

(Please complete and return with KTI application).

NAME ______________________________ DATE __________

PLEASE PRINT

A. COMPUTER SCIENCE EDUCATION

[1] How many hours of computer science courses have you taken? _____

[2] Using a 4.0 scale, what was your grade point average for these courses? _____

[3] Please list any industry sponsored training seminars in computer science which you have taken.
______________________ ______________________
______________________ ______________________
______________________ ______________________

B. GENERAL SOFTWARE ENGINEERING EXPERIENCE

[1] How many years of full-time programming experience have you had? _______

[2] How many years of part-time programming experience have you had? _______

[3] How many years of experience have you had performing the following functions ?
defining requirements _____
preparing design descriptions _____
programming _____
testing code written by another programmer _____
maintaining code written by you _____
maintaining code written by another programmer _____

C. SOFTWARE ENGINEERING PREFERENCES

[1] What is your favorite programming language? ______________

[2] What is your favorite operating system ? ______________

[3] What is your favorite command language ? ______________

[4] What is your favorite software development activity ?
____
____
____

[5] List any other programming preferences that you have.
____
____
____
____

D. SPECIFIC SOFTWARE ENGINEERING EXPERIENCE

In the space below, please describe the software development projects that you have participated in. Attach additional pages if necessary.

Include the following in your description.

[1] Project Title

[2] Project Size (i.e. lines of code, no.of modules, no. of participants).

[3] Project Duration (i.e. no. of years).

[4] Project Description (i.e. application area, purpose).

[5] Development Environment (i.e. programming language, operating system, command language, development techniques).

[6] Your Responsibilities (i.e. full-time or part-time involvement, percent of different activities performed).

(THANK YOU FOR COMPLETING THIS INSERT).

KTI EQUAL OPPORTUNITY/AFFIRMATIVE ACTION EMPLOYER 3/25/83

Figure 1. Software Engineering Experience Questionnaire

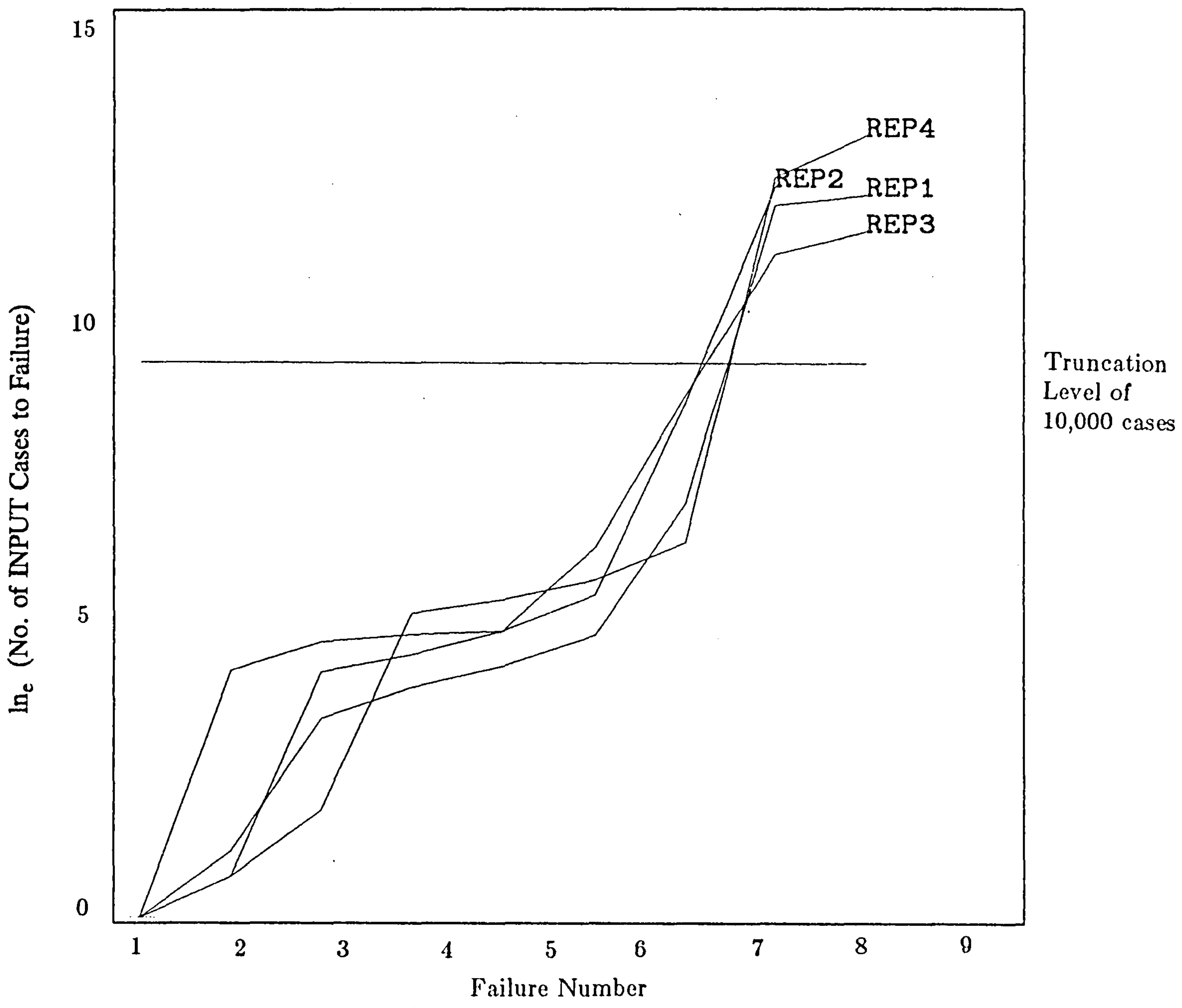
program is returned to its initial design stage, and the same corrections are applied to the failed program. Replications continue until enough observations have been collected to achieve the desired level of statistical accuracy for the maximum likelihood estimate of the program failure rates.

Replication Length

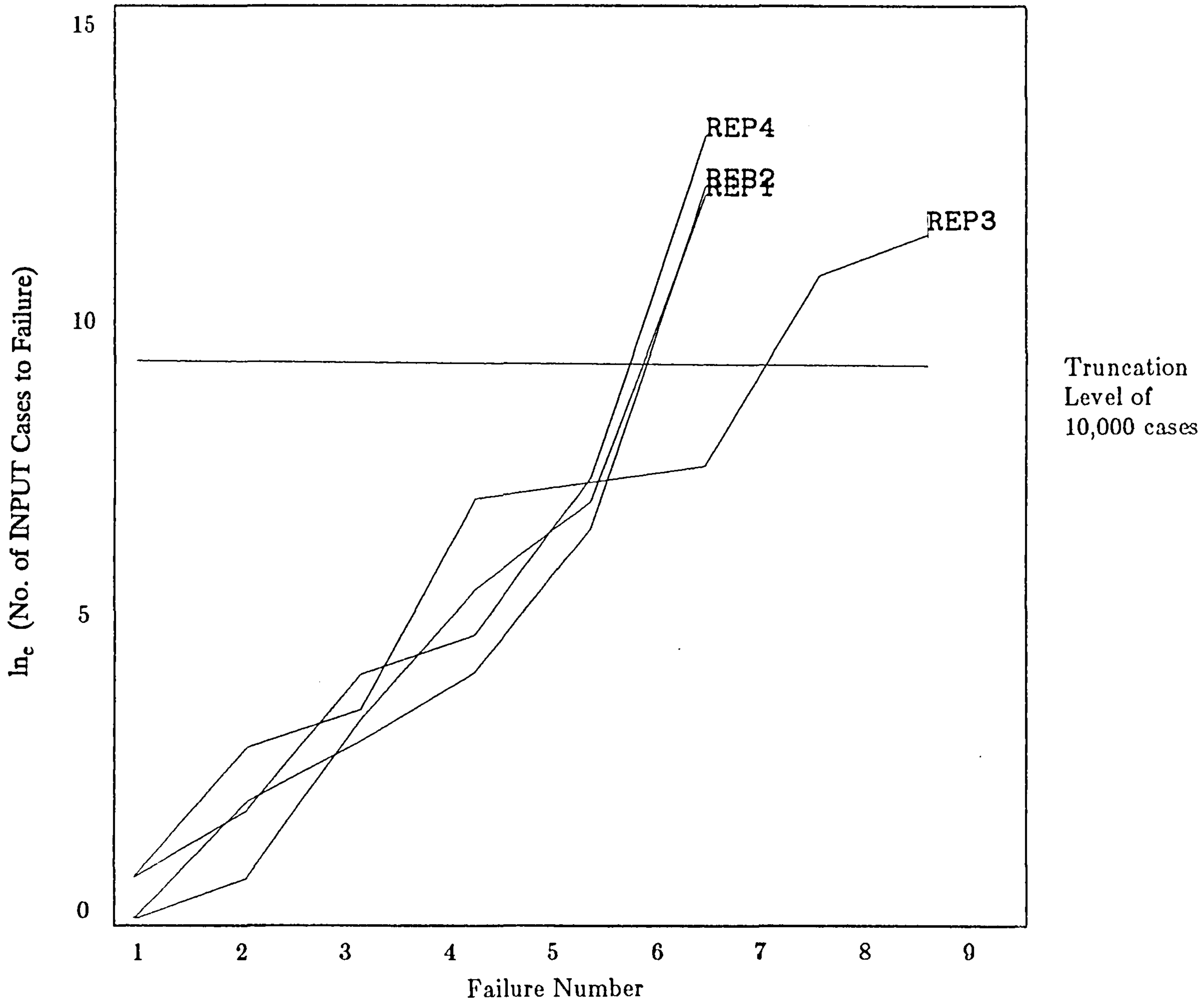
The length of a replication was determined by first executing the N-version system for 500,000 input cases for four replications and then reducing the replication length to 10,000 input cases for the remaining number of replications. The choice of this replication length was based on the observation that most of the known program errors occurred before executing 10,000 input cases as depicted in Plots 1 and 2. Table 1 gives the actual failure times observed. These failure times are based on a number of faults initially perceived: 11 faults in AT1, 1 fault in AT2, and 20 faults in AT3. For certain input combinations, several faults can occur at the same time. This fixed length censoring simplifies modeling and analysis by yielding a set of planned truncated observations. Using input cases instead of execution time simplifies comparison of the reliabilities of the independent implementations.

Table 1. Input Cases to Failure for Replications of 500,000 Input Cases

Code	Rep	Input Cases to Failure								
AT1	1	1	3	27	46	65	110	953	135873	162506
	2	1	60	97	110	116	210	5137	186106	
	3	1	2	58	78	116	468	5786	60450	88535
	4	1	2	6	155	197	273	504	216484	424374
AT2	1	1								
	2	1								
	3	2								
	4	2								
AT3	1	1	2	27	234	1031	162506			
	2	1	7	19	60	655	186106			
	3	2	17	32	1057	1889	43594	88535		
	4	2	6	57	111	1505	424374			



Plot 1. Input Cases to Failure for AT1



Plot 2. Input Cases to Failure for AT3

Replication Sample Size

Choosing the sample size involves determining k, the number of replicated observations of failures needed to accurately estimate p, the probability that the program will fail (due to a specific fault) on a given execution. We assume a geometric distribution,

$$P(x) = (1-p)^{x-1}p \text{ , } 0 \leq p \leq 1 \text{ and } 0 \leq x \leq 10{,}000$$

where P(x) is the probability that the program fails on input case x. The maximum likelihood estimate (M.L.E), $\hat{p}$, is obtained by computing the maximum of

$$L(p; x_1, x_2, \cdots, x_k) = \prod_{j=1}^{k} (1-p)^{x_j - 1} p \quad \text{for } 1 \leq j \leq k,$$

where $L(p; x_1, x_2, \ldots, x_k)$ is the joint likelihood function for k observations of x.

The M.L.E. is $\hat{p} = \frac{k}{\sum_{j=1}^{k} x_j}$, which is a biased estimator with an asymptotic variance (using the Cramer-Rao Inequality [11]) of Var ($\hat{p}$)$\geq \frac{p^2(1-p)}{k}$.

In determining the sample size we wish to control the relative error, r, in the estimated failure probability, $\hat{p}$, with a risk of $1-\alpha$. That is, $\Pr(| \hat{p} - p | \geq rp) \leq 1-\alpha$. Thus, setting

$$rp = Z_{1-\alpha/2} \times \sqrt{\left(\frac{p^2(1-p)}{k}\right)}$$

gives

$$k = \frac{Z^2_{1-\alpha/2}}{r^2} \times (1 - p)$$

where $Z_{1-\alpha/2}$ is the $1-\alpha/2$ point of the standard normal distribution.

Table 2 shows the number of replicated observations required for selected values α, r, and p. Since we are interested in accurately estimating the reliability of programs which have been debugged until p is sufficiently small, we should choose k based on p small. But the allowable risk, $1-\alpha$, must presumably be compatible with the failure probability, p. This poses a problem because k increases as the allowable risk decreases, and therefore, indirectly k increases as p decreases. This report is based on the current number of k=25 replicates at the end of Task 14. We are gathering more data under an additional task.

TABLE 2. NUMBER OF REPLICATED OBSERVATIONS

	p=.05			p=.01			p=.005			p=.001			p=.0001			p=.00001		
r	.5	.25	.10	.5	.25	.10	.5	.25	.10	.5	.25	.10	.5	.25	.01	.5	.25	.01
1- α = .90	11	42	259	11	44	270	11	44	271	11	44	272	11	44	272	11	44	272
1- α = .95	15	59	365	16	61	381	16	62	383	16	62	384	16	62	384	16	62	384
1- α = .99	26	102	633	27	106	660	27	107	663	27	107	666	27	107	666	27	107	666
1- α = .9999	43	169	1047	45	176	1091	45	177	1097	45	177	1101	45	177	1102	45	177	1102

3. THE CODE UNDER TEST

3.1. The Launch Interceptor Condition Application Task

The Launch Interceptor Condition problem was originally reported in a 1973 study by TRW [11] and was used by Nagel and Skrivan in their study. The problem requirements specification can be found in Appendix A. The following paragraphs provide an overview of the problem.

The launch interceptor condition problem specifies a part of a radar tracking system that generates a launch interceptor signal based upon input radar tracking information available at the instant the module is called. Values of input parameters determine which combination of individual launch interceptor conditions are relevant to the immediate situation. The interceptor launch key is normally considered locked; only if all relevant combinations of launch conditions are met will the button unlocking signal be issued.

The launch interceptor condition module determines whether each of 15 parameterized launch interceptor conditions (LIC's) are met for a set of up to 100 2-dimensional points provided as input. Additional inputs dictate how to combine the outcomes of these determinations to form a final unlocking matrix (FUM) from which the launch/no launch signal will be computed. That is, the 15 individual launch conditions specified are examined; a boolean decision is reached as to whether each condition is met, thereby forming the conditions met matrix (CMM), a 15 element vector (one element per condition). The input Logical Connector Matrix (LCM), a 15 by 15 symmetric matrix, is consulted to see which individual conditions must be considered jointly in some way. CMM values are combined as indicated by the LCM to form the off-diagonal elements of the preliminary unlocking matrix (PUM), also a 15 by 15 symmetric matrix. The PUM's diagonal elements, given as input, represent which launch interceptor conditions actually matter in this particular launch determination. Each diagonal element dictates how to combine the off-diagonal values in the same PUM row to form the corresponding element of the FUM. If, and only if, all the values in the FUM indicate a "go" condition is the launch signal generated. Figure 2 provides a schematic representation of the problem. Implementation of the problem resulted in 419, 464, and 585 lines of uncommented FORTRAN code with all known bugs corrected for AT1, AT2, and AT3 respectively.

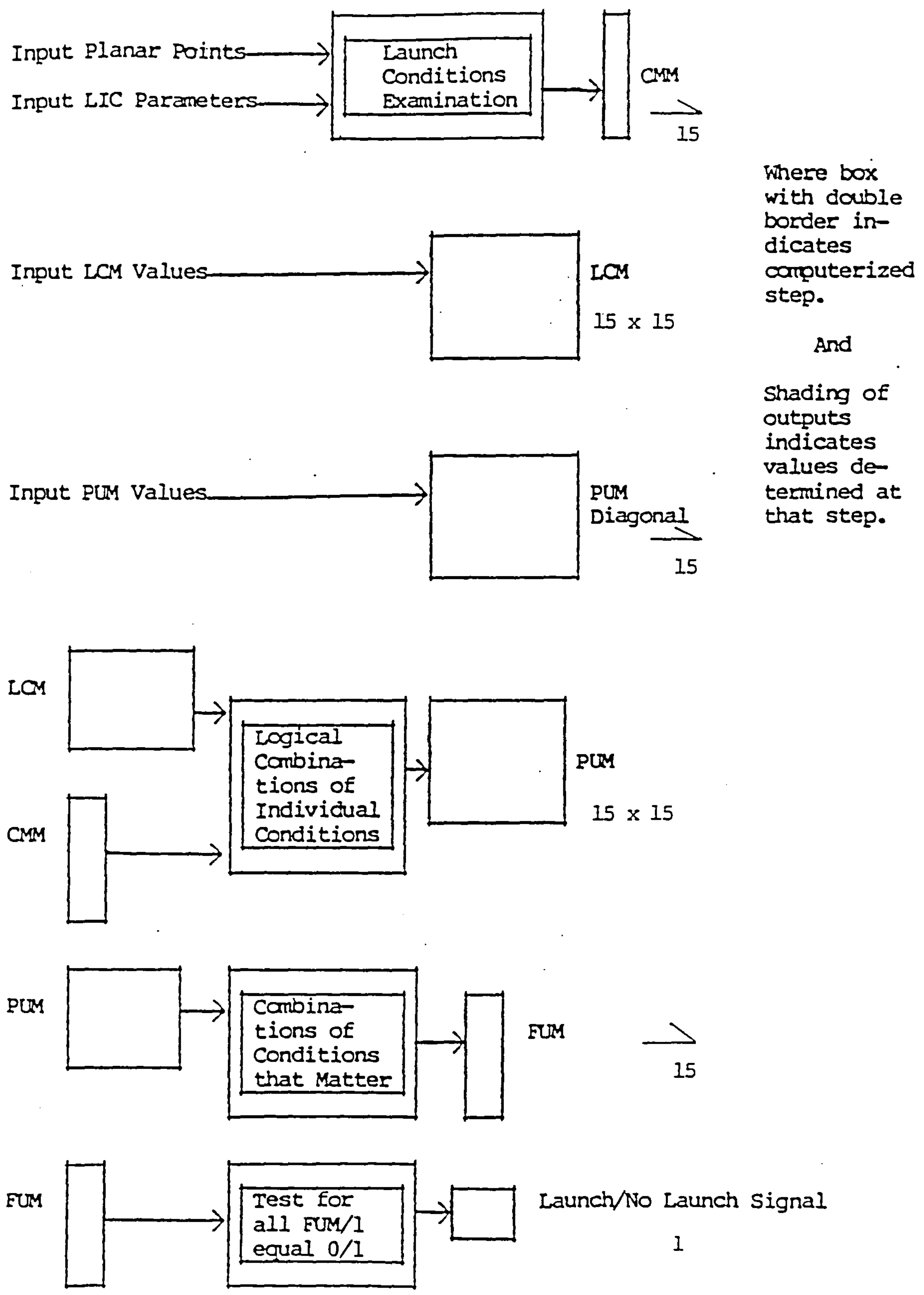

Input Planar Points
Input LIC Parameters
Launch Conditions Examination
CMM
15
Where box with double border indicates computerized step.
And
Shading of outputs indicates values determined at that step.
Input LCM Values
LCM
15 x 15
Input PUM Values
PUM Diagonal
15
LCM
CMM
Logical Combinations of Individual Conditions
PUM
15 x 15
PUM
Combinations of Conditions that Matter
FUM
15
FUM
Test for all FUM/1 equal 0/1
Launch/No Launch Signal
1

Figure 2. A Schematic Representation of the Launch Interceptor Problem

3.2. Specification Failures

During the acceptance testing phase and the n-version simulation of the code under test, a few problems with the specification became apparent. The following list summarizes the salient problems encountered with the specification provided to the programmers.

(1) The words "all" and "any" confused the programmers.

(2) The computation of the area for Launch Interceptor Conditions 4, 11, and 15 resulted in a problem due to round-off.

(3) If one or both of the endpoints coincides with the vertex in Launch Interceptor Conditions 3 and 10, the angle cannot be computed if one uses the definition of an angle given in the VNR Concise Encyclopedia of Mathematics. [12] The problem specification should contain statements about the precision of all variables.

Using a formal specification language provides more complete, consistent, and executable specifications. The formal specification approach does not, however, eliminate errors which result from misinterpreting the specification. Until advances are made in improving the performance of executable specifications or in transforming programs, using redundancy may be a viable technique for surfacing interpretation errors.

3.3. The Usage Distribution

The input data for the Launch Interceptor Condition Problem were generated according to the usage distributions specified in Appendix B.

Several of the input data values were modified so that the usage of the programs would include pathological input cases. The changes were made to the values used in the Nagel & Skrivan study [4] and are identified in the last section, Input Data Modification, of Appendix B. Since an objective of this research project was to test the unequal error likelihood hypothesis and not to discriminate between the actual magnitude of the fault rates, these modifications do not have an impact on this research objective. In retrospect, it appears that, rather than particular values of the parameters (in Appendix B) having been chosen, the entire parameter set should have been made random. The latter task is easily accomplished using the N-VERSION CONTROLLER INTERFACE tool which is discussed in Section 4.3.2 of this report.

3.4. The Pseudo-Random Number Generator

The results of this study depend on the operational testing performed using the N-VERSION CONTROLLER. A major component of this testing is the generation of data according to the usage distribution specified in Section 3.3. Generation of this usage distribution relies on a random number generation package. The GGUBT package [13] was selected for its computational efficiency and for its superior performance on a battery of tests which detect departures from randomness. [14] GGUBT generates a sequence of $2^{31}-2$ uniform random deviates. It uses a linear congruential multiplicative modulo $2^{31}-1$ recurrence relationship with multiplier 397204094.

To ensure that the replications did not exhaust the random number sequence, an upper bound for the number of deviates required was computed. Each input case generated requires 6p + 240 deviates, where p is the number of (x,y) coordinates which describes the radar track. Since max p = 100, the maximum number of deviates required per input case is 840. Thus, for 25 replications of 10,000 cases this simulation requires in the worst case $2.1\times10^8 \leq 2^{31}-2$ deviates. Appendix C lists the random number seeds for the beginning of each replication.

4. STEPS IN THE STUDY

4.1. Emulating an Actual Software Development Environment

The three programmers coded and debugged a radar tracking problem adapted from the Boeing study [4] and a pitch axis control algorithm for a Piper PA28 aircraft. The problem requirements were specified in English, and the programmers independently constructed their software designs and implemented them using VAX-11 FORTRAN. Programmer questions related to the problem specification were directed to an RTI staff member who functioned as a user for the duration of the project. This staff member, who is a mathematician, had over 17 years of experience in FORTRAN programming for applied mathematics applications. He coded and extensively tested his own version of each problem and was, therefore, knowledgeable about the problem. He provided information relating to programming style, documented all questions about the specification, and answered similar questions as consistently as possible.

Another RTI staff member functioned as the project manager. This staff member, who is a research software engineer, had over seven years programming experience and five years program management experience. He developed the automated development data collection tools and designed and oversaw the development of the N-VERSION CONTROLLER and its associated INTERFACE. The programmers were instructed to deal with the project manager in the event of system difficulties and personnel/management problems. The project manager provided the programmers with a system primer and a tutorial problem (i.e., a pseudo-mathematical specification of a majority rule voter) so that they could become acquainted with the system.

Although acceptance tests and a fairly lenient deadline for program completion were specified in advance, the programmers were not restricted to the amount of program compilation and execution permitted before delivering the programs to management. They were instructed to optimize program reliability first and program efficiency second. They were informed that their programs were to be used in a prototype n-version system for avionics research and that the reliability of the overall n-version system would be evaluated. The programmers were also given special assignments and were told that the special assignments had a lower priority than the development and maintenance of the problem assigned.

Software tools were developed to support the experiment. These tools included a programming environment, an N-VERSION CONTROLLER, and an N-VERSION CONTROLLER INTERFACE. Information was captured both manually and with the automated tools developed throughout the project.

4.2. Collecting Data During Software Development and Repair

4.2.1. Manual Data Collection

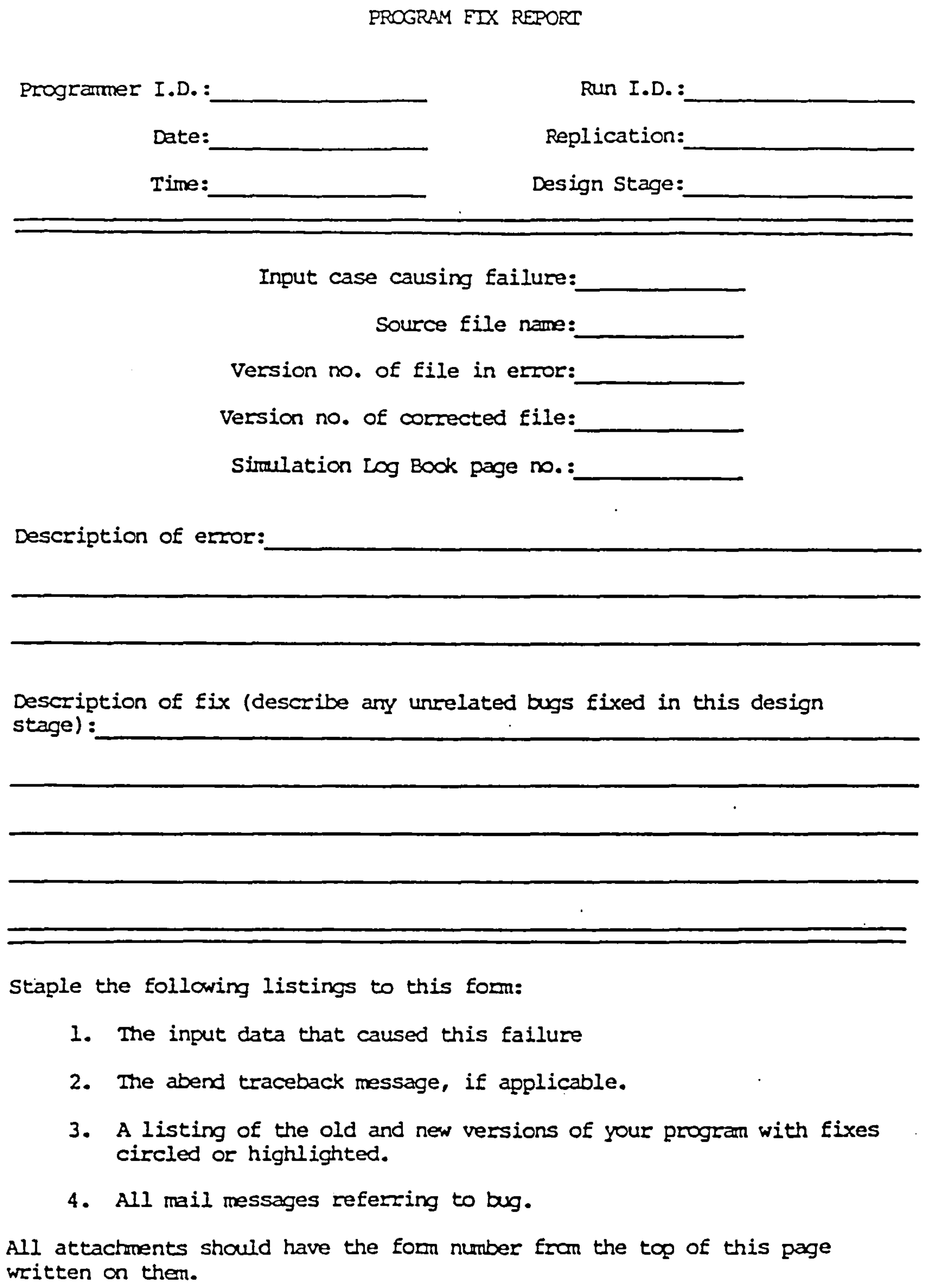
The programmers were asked to record their daily activities in an activity notebook. These books were monitored to ensure that the appropriate information was recorded. When a program failed during simulation, the programmer was notified by electronic mail that his or her program had failed. The mail message indicated whether or not an abend occurred or which bits in the CMM were in error. If an abend occurred, the traceback message was provided. The input case which the program failed to execute successfully was always provided. Changes made in the failed program were annotated by using a special notation in the code. The notation used was a fixed format program comment, and these comments contained a fix number, a fix type, a fix type identifier, and the associated program change report form number. The fix type field marked the beginning of a fix (Type B), the addition of a line of code (Type A), the modification of a line of code (Type C), and the deletion of a line of code (Type D). Deleted lines of code were retained by making them comments. The fix type identifier for a Type B comment contained three 3-digit numbers which indicated the number of Type A, C, and D changes associated with the single fix. The fix type identifier for all the other comment types contained a three digit number which uniquely identified the change as a part of the fix. Figure 3 depicts the program fix report used to manually document fix information. The programmers received a set of guidelines for completing this form.

4.2.2. Automated Data Collection

Automated data collection during development and repair was achieved by instrumenting the environment used by the programmers. The programming environment provides a restricted subset of VAX/VMS commands for file manipulation, file editing, and program compilation and execution. This environment automatically records information about the programmer's activities during program development, including the dates and times at which the activities occurred. Files cannot be deleted by the programmers, so a detailed record of all file modifications is available. Figure 4 shows the organization of the programming environment.

This system operates by means of a command line interpreter which resides between the programmer and the VAX-VMS command line interpreter. Each command line is parsed and checked for errors or disallowed actions. If the command is legitimate it is executed. This execution is referred to as an event. The commands implemented are:

FORTRAN	HELP
LINK	TYPE
RUN	PRINT
TESTED	TCAI
DIRECTORY	LOGIN
MAIL	LOGOUT
EDIT	



PROGRAM FIX REPORT

Programmer I.D.: ____________	Run I.D.: ____________
Date: ____________	Replication: ____________
Time: ____________	Design Stage: ____________

Input case causing failure: __________

Source file name: __________

Version no. of file in error: __________

Version no. of corrected file: __________

Simulation Log Book page no.: __________

Description of error: ______________________________

Description of fix (describe any unrelated bugs fixed in this design stage): ______________________________

Staple the following listings to this form:

1. The input data that caused this failure
2. The abend traceback message, if applicable.
3. A listing of the old and new versions of your program with fixes circled or highlighted.
4. All mail messages referring to bug.

All attachments should have the form number from the top of this page written on them.

Figure 3. Fix Report Form

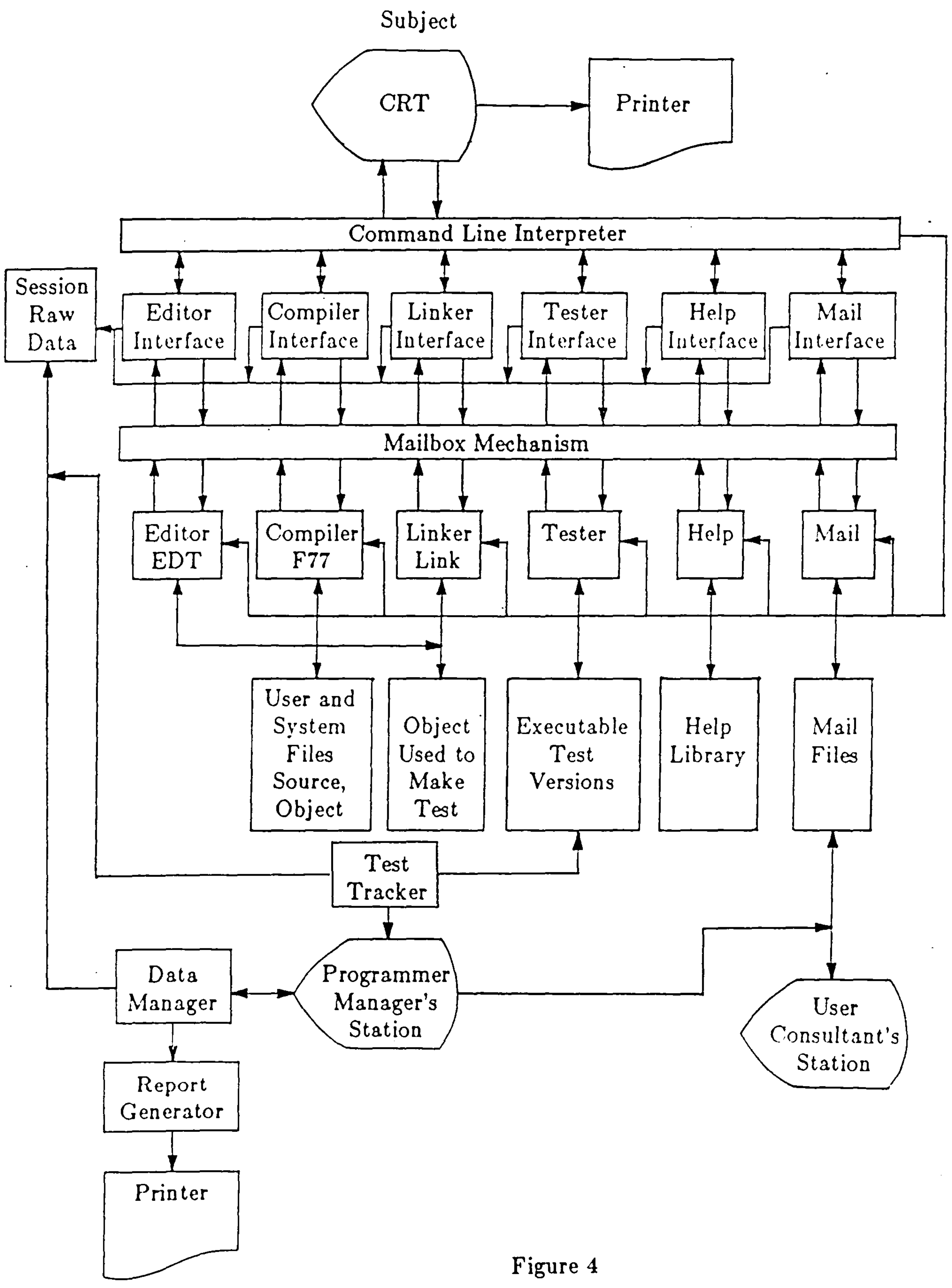


Figure 4

INSTRUMENTED SOFTWARE DEVELOPMENT ENVIRONMENT

A command language primer [15] which describes the environment is available and was provided to the programmers along with the VAX-11 FORTRAN manual for their use during program development. The statistics and text gathered for each event are:

command line	program number
pointers to the files referred to	page faults
	process page counts
error messages	virtual peak
cpu time	working set size
buffered i/o's	elapsed clock time
direct i/o's	user name
subject number	terminal

The text data stored is dependent on the command executed. Edit commands cause a new file, a pointer to the old file, and all of the editor commands to be stored. Compile commands cause a pointer to the source compiled to be stored. Compiler errors, linker errors and the errors generated by other commands are also part of the text data that is stored.

4.3. Generating the Error Data

4.3.1. The N-VERSION CONTROLLER

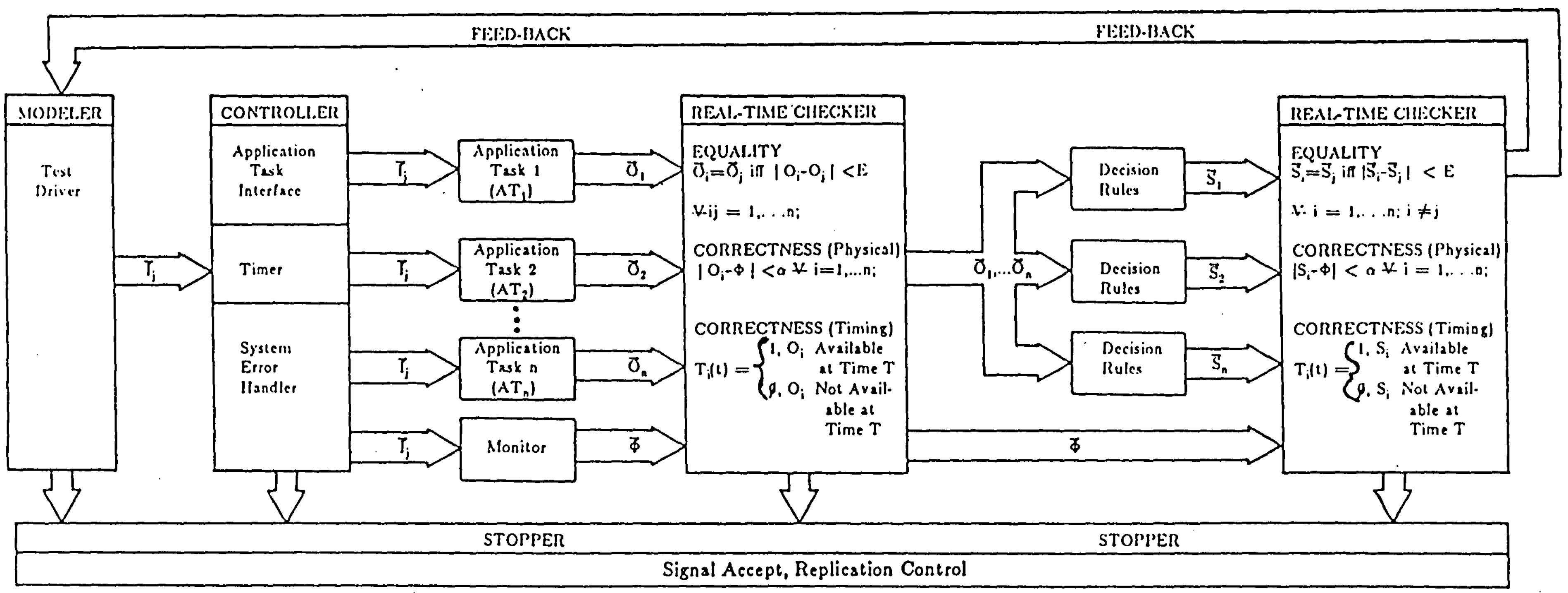
After all three programmers had completed a given problem, their programs were executed using a variant of the N-VERSION CONTROLLER shown in Figure 5. The N-VERSION CONTROLLER models the usage distribution described in Section 3.3. Its major components include a modeler, which generates a set of random problem inputs on each simulation pass using parameters which are specified by the experimenter when the simulation is initiated; a controller, which executes N versions of the program against the problem inputs and passes the results to a set of selection programs; and a stopper, which halts the simulation when one of the following occurs:

- The N application tasks are not in agreement
- One or more of the application tasks has abended
- A predetermined stopping rule has been satisfied
- One or more of the application tasks has overwritten shared memory

When the simulation is halted, a snapshot of the system state is written to a data file for later analysis. This snapshot includes the inputs that caused the failure, the outputs of all application tasks and voters, simulation parameters at time of failure, and a number of flags indicating the cause of the failure. Snapshot file records are indexed by simulation run, replication, design stage, and input case. The current values of all variables saved in a snapshot are accessible dynamically during simulation through the simulation INTERFACE. A back-up log of simulation activity is kept manually.

4.3.2. The USER INTERFACE

The INTERFACE uses table-driven menus for data display and simulation control. It is described in greater detail in the INTERFACE User's Guide [15]. The set of menus


FEED-BACK
FEED-BACK
MODELER
Test Driver
CONTROLLER
Application Task Interface
Timer
System Error Handler
Application Task 1 (AT_1)
Application Task 2 (AT_2)
Application Task n (AT_n)
Monitor
T_j
$\bar{O}_1$
$\bar{O}_2$
$\bar{O}_n$
Φ
REAL-TIME CHECKER
EQUALITY
$\bar{O}_i = \bar{O}_j$ iff $|O_i - O_j| < E$
$\forall ij = 1, \ldots n;$
CORRECTNESS (Physical)
$|O_i - \Phi| < \alpha \;\forall\; i = 1, \ldots n;$
CORRECTNESS (Timing)
$T_i(t) = \begin{cases} 1, O_i & \text{Available at Time T} \\ \emptyset, O_i & \text{Not Available at Time T} \end{cases}$
$\bar{O}_1 \ldots \bar{O}_n$
Decision Rules
$\bar{S}_1$
$\bar{S}_2$
$\bar{S}_n$
REAL-TIME CHECKER
EQUALITY
$\bar{S}_i = \bar{S}_j$ iff $|\bar{S}_i - \bar{S}_j| < E$
$\forall i = 1, \ldots n; i \neq j$
CORRECTNESS (Physical)
$|S_i - \Phi| < \alpha \;\forall\; i = 1, \ldots n;$
CORRECTNESS (Timing)
$T_i(t) = \begin{cases} 1, S_i & \text{Available at Time T} \\ \emptyset, S_i & \text{Not Available at Time T} \end{cases}$
STOPPER
STOPPER
Signal Accept, Replication Control

DESIGN OF N-VERSION CONTROLLER

Figure 5

is kept small for the convenience and the experienced user alike. The organization of the menu/data display for the INTERFACE developed for the Launch Interceptor Condition Problem is shown in Figure 6.

The INTERFACE has commands for the examination of the simulation variables and parameters during simulation and after a failure has occurred and for controlling execution of the N-VERSION CONTROLLER. Snapshots of past failures can be examined with the INTERFACE whether or not the CONTROLLER is executing. The INTERFACE currently provides the following types of commands:

- Select menu - go to a different menu in the graph.
- Examine data print the values of the variables in the display selected on the terminal screen
- Next page - go to the next page of a multi-page display. This command is a carriage return. The last page of a display is circularly linked to the first page of the display.
- Modify parameters - change the parameters controlling the simulation. Data modification is password-protected, and certain parameters can only be changed at the start of a run.
- Stop/start simulation - force the simulation to halt, or restart a halted simulation from where it left off.
- Read snapshot - read a snapshot from the snapshot data file and display its variables instead of the active simulation variables. A snapshot can be selected by replication number and design stage, or by failure number.
- Next snapshot - read the first snapshot in the next replication on the snapshot data file.
- Initialize simulation - start a new run/replication/design stage by compiling the specified application tasks, linking them with the N-VERSION, and starting execution.

Thus all major simulation functions and variables are under the control of the experimenter through the INTERFACE. The N-VERSION CONTROLLER can be restarted from any past failure by directing to read the simulation failure state from the snapshot data file. The parameter and stopping condition modification is password protected to prevent inadvertent damage to the data during a simulation run.

Since the INTERFACE code is modular and table-driven, the INTERFACE can be adapted to a variety of simulation problems with a minimal amount of effort. Menu items can be added, deleted, or modified by changing fields in a FORTRAN common block that describes nodes and arcs in the menu graph. Individual data displays can be added, deleted, or modified by changing fields in another common block that describes the format and screen position of data elements as subnodes on a display page; display page graphs are arranged in a circularly linked list of pages to provide a method for rapidly stepping through pages in a display.

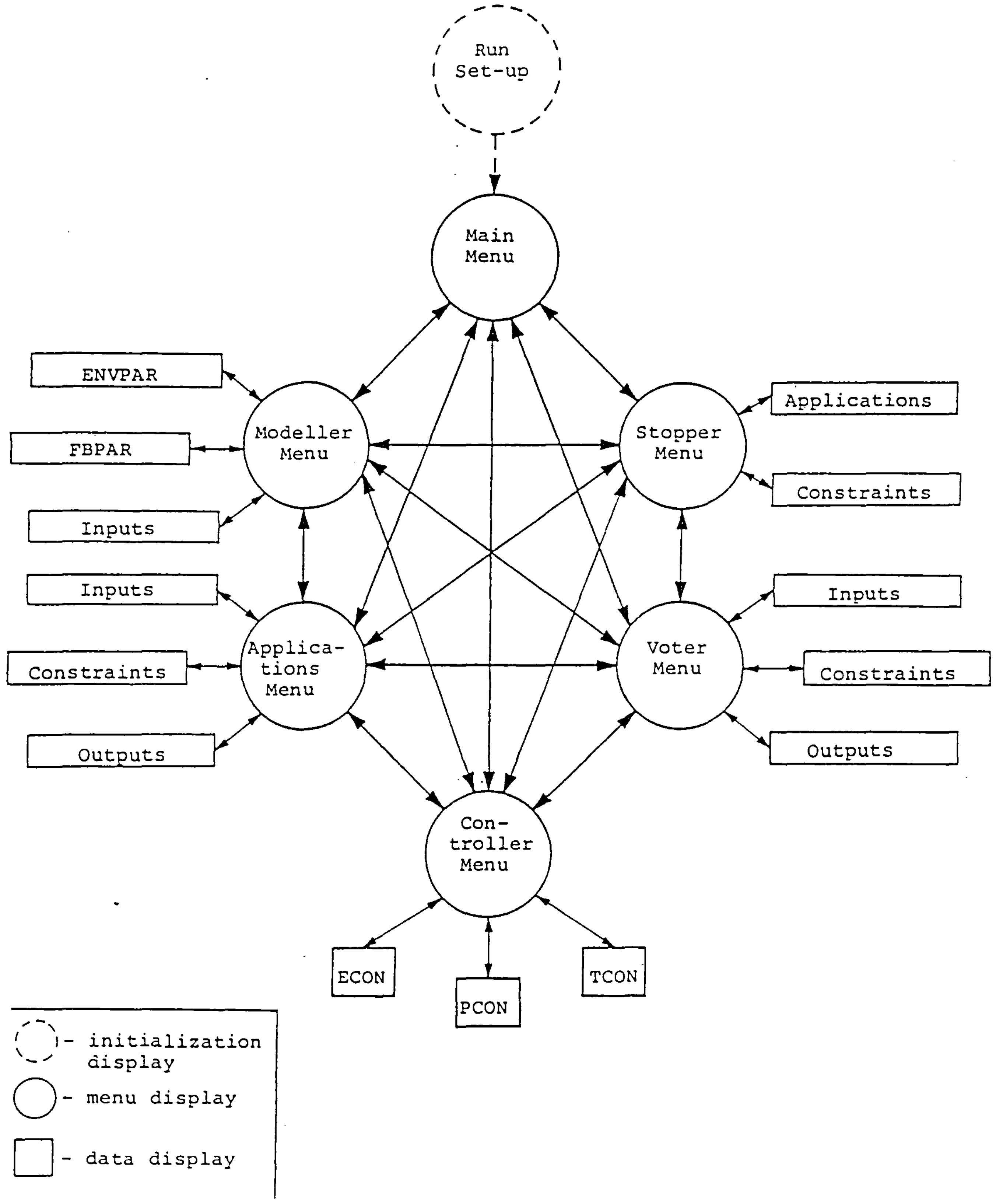


Figure 6. Menu/Display Interconnection Graph.

Communication between the N-VERSION CONTROLLER and its associated INTERFACE is accomplished with a set of event flags and a memory region that is shared by both programs. When the INTERFACE examines the simulation variables dynamically during simulation, it samples the shared memory region periodically, and data are displayed from a local copy of the shared region. The N-VERSION CONTROLLER writes a new set of data into the shared region at each simulation pass. If the experimenter issues a modify data request from the INTERFACE during simulation, an event flag is cleared to indicate that a modify request is pending, and the INTERFACE suspends execution until the N-VERSION CONTROLLER sets the flag to indicate that it has suspended execution so the INTERFACE can write into the shared region. The INTERFACE must set another event flag to start up the N-VERSION CONTROLLER after the data modification is complete.

5. DATA

5.1. Software Failures

Understanding the terms failure, error, fix, and fault is essential to the interpretation of the results obtained. For this problem, an application task failure is detected when a task disagrees with any other unfailed application tasks or the extensively tested version on its FUM, CMM, PUM, and Launch/No Launch output during simulation. This disagreement event constitutes a failure, the incorrect element(s) of the output variables (is/are) the error(s), and the minimum code change required to correct a single error is a fix. The fault is the conceptual flaw in the program which is corrected by a fix. Table 3 describes the faults in the three application tasks and provides the associated fix number. A few program abends and a memory overwrite were observed. However, most of the errors involved incorrectly setting the bits of the CMM. With the exception of the fault identified by fix number 1 in AT2, there were no observations of faults in computing the PUM, the FUM, and the FC. Fixes applied to FORTRAN functions frequently caused errors in more than one bit of the CMM.

Appendix D shows the input cases to failure by fix number for each of the respective application tasks. The granularity used in reporting this failure information was not immediately clear, but after some investigation, a decision was made to report the failure data by fix where fix is associated with the finest error decomposition — in this case an error in a bit of the CMM. The fault corrected by fix 10 of AT1 is a valid fault. Its rarity (it occurred around 200,000 input cases on the long replications) provides a reasonable explanation as to its non-observation during these 25 replications. Fixes 3, 4, and 7 for AT3 are invalid fixes. They constitute changes to code for a "perceived" fault which was non-existent. The logic incorporated by these code changes is isomorphic to the logic prior to the code changes. The fault corrected by fix 20 for AT3 was observed on input case 43,594 on the third replication of 500,000 input cases. If one assumes that input cases for each replication can be added, one would "expect" 1,043,594 input cases for this fault to manifest itself. Note that there was no application of Fix 20 during the 25 replications of 10,000 input cases.

After running the simulation for the replications of 500,000 input cases, it became evident that the fix application procedure was error prone and checks for avoiding these errors should be implemented. One check resulted from the observation that the installation of a fix may correct the error but may not be the appropriate fix. This possibility is an artifact of the boolean outputs. Checking all candidate fixes (i.e., fixes associated with the error), provided some assurance that the correct fix was applied. On the second long replication, it became apparent that for certain failures it was impossible to convincingly chose the correct fix without a great deal of analysis. Since dependencies among faults may be important for the modeling of the software failure process, we intend to investigate the impact of this type of imperfect detection on the failure times by repeating these replications and applying all combinations of fixes.

The application task output sometimes disagreed as a result of floating point computations. These round-off errors resulted from the programmers using different formulas in computing the mathematical calculations required for deciding whether or not a bit of the CMM was met. Changes to the application tasks were not made when a round-off error occurred. Observations of round-off errors were recorded since it was of interest to observe the frequency of round-off errors. Tallying these errors per input case yields a rate of 9.3×10^{-6}. As previously mentioned, observing round-off errors resulted from an omission from the problem specification. Future specifications of the problem should provide precision specifications for all variables and a utility for scaling all floating point variables used in comparisons.

TABLE 3. FAULT/FIX DESCRIPTIONS

Code	Fix No.	Description
AT1	1	Function ANGLEA overwrote data in common region by making assignments to input variables.
	2	CMM(5) - wrong data point was discarded due to erroneous index specification in a loop.
	3	CMM(7) - three erroneous index specifications.
	4	CMM(5) - did not specify logic that if $M \leq 1$, CMM(5) cannot be met.
	5	Function RAD program abended due to an out-of-bound argument when calling the FORTRAN library routine which computes the arccosine.
	6	CMM(8) - specified LT. on bound instead of LE.
	7	Function ANGLEA failed to complete FIX DO 1 by not changing all variable names.
	8	Function ANGLEA program abended due to an out-of-bound argument when calling the FORTRAN library routine which computes the arccosine.
	9	CMM(1) - used wrong formula to compute the difference between 2 points.
	10	CMM(3) - inconsistent definition of a null vector with other ATs.
	11	CMM(1) - inconsistent definition of a null vector with other ATs.
AT2	1	Used integer variable instead of boolean variable when setting the FC.
AT3	1	CMM(7) fix for misinterpretation of any
	2	CMM(13) fix for wrong variable N6 $\rightarrow$ N1
	3 $\rightarrow$ 6	CMM(2) $\rightarrow$ CMM(5) inappropriate handling of computation when the No. of data points is small.
	7 $\rightarrow$ 15	CMM(7) $\rightarrow$ CMM(15) inappropriate handling of computation when the No. of data points is small.
	16	CMM(7) - the upper bound of a do loop was incorrectly set.
	17	Function AGLCOS program abended when trying to compute cosine.
	18	Program abended due to a division by zero in function RADCIR. DELETES an entire subroutine [ERRSTP]
	19	CMM(3) - Program returned wrong value from AGLCOS. This fix is related to fix 17.
	20	Function PERDIS program abended when all 3 points were the same.

6.0. ANALYSIS

6.1. Time Independent Estimation

A simple time independent estimate of reliability can be calculated by viewing the program executions as a sequence of Bernouilli trials as done in [16]. Each replication consists of 10,000 trials and the reliability per replication is the relative frequency of successful trials. This simple estimate remains fairly constant for different replications and uses little of the information in the data. In our case where we are repairing the programs, this simple measure does not take reliability growth into account. It provides, however, a quick comparison of the reliabilities of the different programs. Table 4 gives this estimate for AT1 and AT3 averaged over the 25 replications.

TABLE 4. TIME INDEPENDENT RELIABILITY ESTIMATES

Code	AT1	AT2	AT3
Reliability	.9992	.9999	.9985

6.2. Time Dependent Estimation

6.2.1. Log-Linear Pattern of Error Rates

One goal of this research was to determine if the error rates of the program design stages, as the faults were discovered and removed, exhibited the log-linear pattern suggested in the study by Nagel and Skrivan [4]. (The program design stage error rates in this report correspond to the stage probabilities of the Nagel and Skrivan study). That is, can an estimate of the error rate, λ_j, of the jth design stage of a program be approximately represented by a function of the form

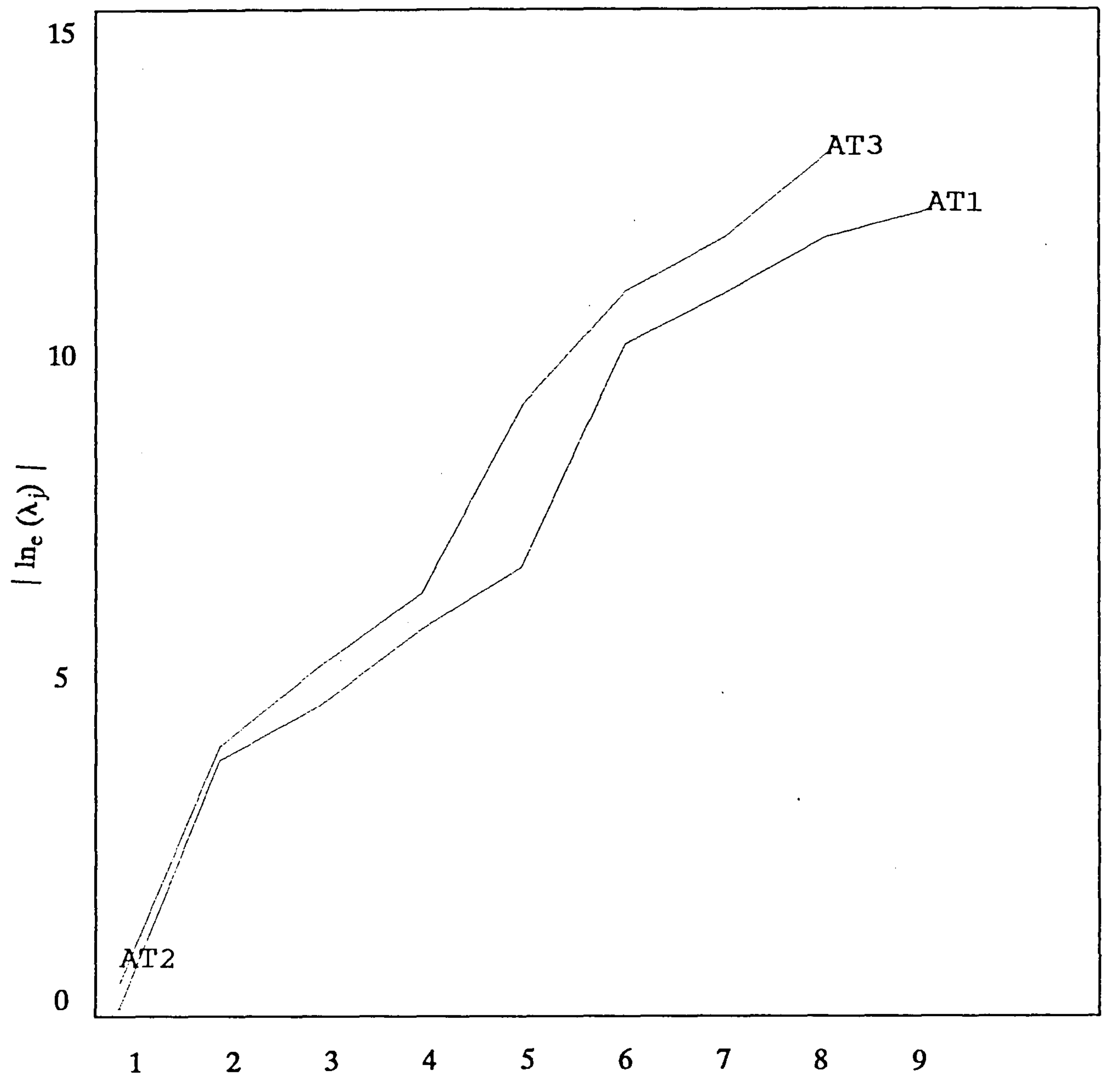
$$\lambda_j = e^{\alpha+\beta j}$$

where j is the design stage number and α, β are coefficients to be determined. If there is such a λ_j, then the reciprocal of the mean time to error of design stage j over all replicates containing at least j design stages is an estimate. The absolute value of the logarithm of this error rate estimate is tabulated by design stage number and test program in Table 5 and plotted in Plot 3. The graphs exhibit a linear trend consistent with the Nagel and Skrivan study. The larger the magnitude of the y intercept indicates the longer mean time to observation of the first failure. The observed differences between all slopes are a manifestation of the programmer effect and the use of a different error detection mechanism.

TABLE 5. LOGARITHMS OF ERROR RATE ESTIMATES BY DESIGN STAGE NUMBER

Code	Design Stage Number (j)	k_j	$\sum_{i=1}^{k_j} t_{ij}$	$\| \ln\lambda_j \|$
AT1	1	25	25	0
	2	25	307	2.51
	3	25	538	3.07
	4	25	1174	3.85
	5	25	2208	4.48
	6	25	21525	6.73
	7	20	29598	7.26
	8	7	16338	7.82
	9	2	6414	8.11
AT2	1	25	39	.44
AT3	1	25	32	.25
	2	25	353	2.65
	3	25	809	3.48
	4	25	1665	4.20
	5	24	10792	6.12
	6	21	28529	7.26
	7	9	20364	7.82
	8	4	5777	8.66

where k_j is the number of replicates containing a j^{th} design stage, and $t_{i,j}$ is the time to observation of an error of the j^{th} design stage during replicate i. (Note that time, $t_{i,j}$, is measured from the start of the j^{th} design stage during replicate i, not from the start of the i^{th} replicate).



Design Stage Number

Plot 3. Error Rate Estimates by Design Stage Number

6.2.2. Decreasing Error Rates

Use of a software reliability model implies the existence of a trend in the failure data. To test for trend we use the method of analysis described in Cox and Lewis. [18, pp. 44-51] This test assumes that we are observing a time-dependent Poisson process for which the rate of occurrence takes an exponential form. This assumption appears reasonable based on the findings of the preceding section. To test for trend, we test the null hypothesis of no trend

$$H_o:\beta = 0$$

using the following test statistic which combines information from all replications:

$$U = \frac{\sum_{i=1}^{k} \left\{ \sum_{j=1}^{n_i} \tau_{i,j} - \frac{1}{2} n_i \tau_{i,0} \right\}}{\sqrt{\frac{1}{12} \sum_{i=1}^{k} n_i \tau_{i,0}^2}}$$

where k is the number of replicates,

n_i is the number of design stages less one during the ith replicate,

$\tau_{i,j}$ is the time of the error observed in the jth design stage during the ith replicate (note that here time is measured from the start of the replicate), and

$\tau_{i,0}$ is the duration time of the ith replicate.

This test statistic compares the centroid of the observed times with the mid-point of the period of observation. The test statistic is approximately distributed as $\eta(0,1)$. Table 6 gives the test statistics for AT1 and AT3 which both reject the null hypothesis of no trend at the 95% significance level.

TABLE 6. TREND TEST STATISTICS

Code	Numerator	Denominator	U	RESULT
AT1	-140288	49450	2.84	REJECT
AT3	-95701	41950	2.28	REJECT

6.2.3. Unequal Error Rates Associated with Individual Fixes

Testing the unequal error probability hypothesis involves testing for the homogeneity of the failure rates of the different errors observed. Table 7 shows the absolute value of the logarithm of the error rate by fix for AT1 and AT3. These rates are plotted in increasing order in Plots 4 and Plot 5. To test the hypothesis of unequal error rates we use the Maximum Likelihood Ratio Test described in Cox and Lewis. [18] The null hypothesis of equal rates can be stated as

$$H_o: \lambda_1 = \lambda_2 = \lambda_3 = \cdots = \lambda_n = \lambda$$

Note that here λ_j represents an error rate associated with the j^{th} fix (or somewhat equivalently, the j^{th} fault) as numbered in the 1^{st} replicate. Assuming that faults contribute independently to the error rate of a program, the error rate of a design stage is therefore the sum of the λ_js associated with the faults still in the program.

Assuming that the times to observation of an error due to fault j are exponentially distributed, the test statistic under the null hypothesis is

$$H = 2\left\{ \sum_{j=1}^{N} k_j \left| \ln \lambda_j \right| - K \ln \left(\frac{K}{T} \right) \right\}$$

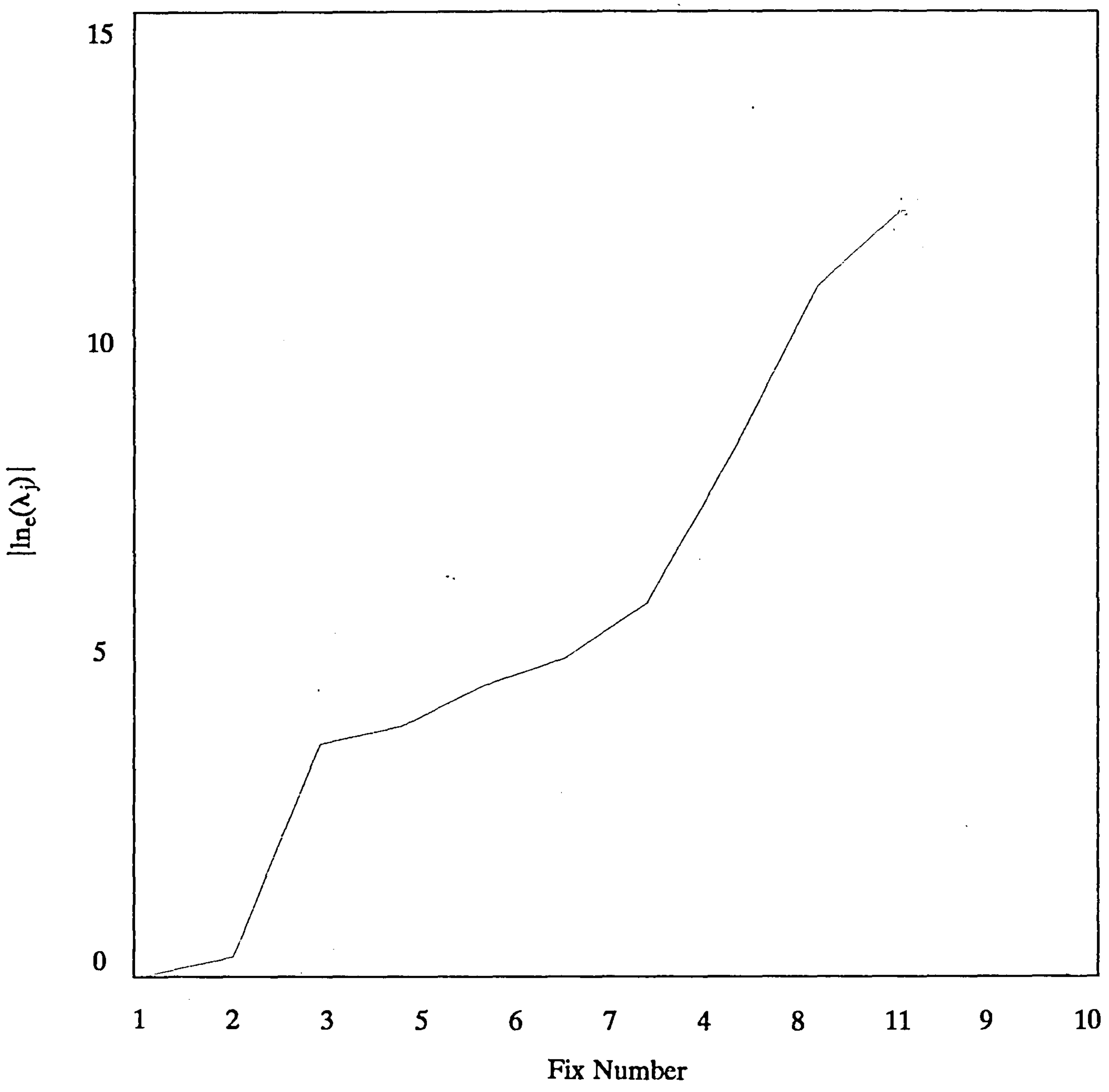
where N is the numbers of errors observed,
k_j is the number of observations of error j,
$K = \sum_{i=1}^{N} k_j$,
and T is the total time under test, in this case 250,000.

This test statistic asymptotically follows a chi-squared distribution with n-1 degrees of freedom. Table 8 gives the test statistics for AT1 and AT3 using partial and full data. Partial data omits the rates for the errors where less than ten observations occurred from the analysis. Full data analysis includes all error rates. The null hypotheses of equal error rates are rejected using full and partial data for AT1 and AT3 at the 95% significance level.

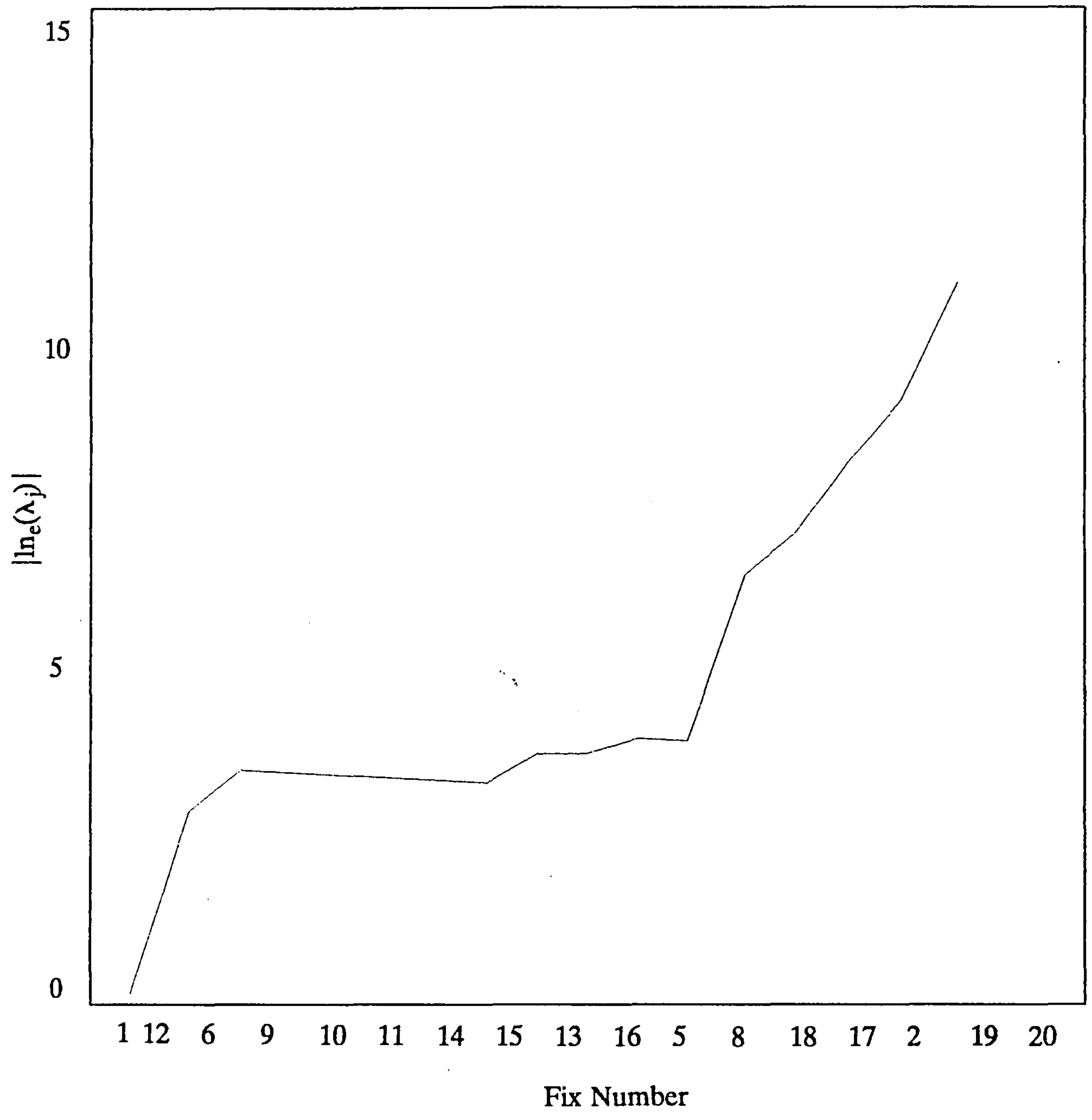
TABLE 7. RATES FOR AT1 and AT3 BY FIX APPLIED

Code	Fix	k_j	$\sum_{i=1}^{k_j} T_{i,j}$	λ_j	$\lvert \ln_e \lambda_j \rvert$	$k_j \lvert \ln_e \lambda_j \rvert$
AT1	1	25	26	.9615	.039	.975
	2	25	35	.7143	.336	8.400
	3	25	1038	.0241	3.726	93.150
	4	25	11589	.0022	6.119	152.975
	5	25	1435	.0174	4.051	101.275
	6	25	2833	.0088	4.733	118.325
	7	25	4527	.0055	5.203	130.075
	8	22	101873	.0002	8.517	187.374
	9	1	248539	.000004	12.429	12.429
	11	2	144357	.000014	11.176	22.352
PARTIAL	---	197	123356	-------	------	792.549
FULL	---	200	516072	-------	------	827.330
AT3	1	25	34	.7353	.307	7.675
	2	14	110335	.00013	8.948	125.272
	5	25	2421	.0103	4.576	114.400
	6	25	1093	.0229	3.777	94.425
	8	25	2421	.0103	4.576	114.400
	9	25	1093	.0229	3.777	94.425
	10	25	1093	.0229	3.777	94.425
	11	25	1093	.0229	3.777	94.425
	12	25	563	.0444	3.115	77.775
	13	25	1769	.0141	4.262	106.550
	14	25	1093	.0229	3.777	94.425
	15	25	1093	.0229	3.777	94.425
	16	25	1851	.0135	4.305	107.625
	17	24	68388	.0004	7.824	187.776
	18	25	29741	.0008	7.131	178.275
	19	1	124160	.000008	11.736	11.736
PARTIAL	---	363	224081	-------	------	1586.298
FULL	---	364	348241	-------	------	1598.034

where here $T_{i,j}$ is the time of observation of the error which gave rise to fix number j during the i^{th} replicate; $T_{i,j}$ is measured from the start of the replicate following the one in which fix number j last occurred, and λ_j is the error rate associated with fix number j.



Plot 4. Rates by Fix Number for AT1



Plot 5. Rates by Fix Number for AT3

TABLE 8. ERROR RATE TEST STATISTICS

CODE	H PARTIAL	RESULT	H FULL	RESULT
AT1	440	reject	4507	reject
AT3	7917	reject	7951	reject

6.2.4. Confidence Limits for Mean Time to Error

Since we have several failure times for each of the errors, it is useful to compute confidence limits. If we assume that x_{ij} the number of trials to failure of the jth error has a geometric distribution which is identical for each of the i replications,
then

$$\hat{m}_j = \sum_{i=1}^{k_j} \frac{X_j}{N}$$

is a sufficient estimator of the mean. Using the charts in Clems [19] Table 9 gives the ranges for x_j and the 98% confidence interval for m_j where $k_j \geq 20$ for each of the application tasks.

6.3. Limitations/Suggestions

It is important to note that the primary goal of this specific research task is to collect data on software failures. As a result, we restricted the amount of analysis performed and have forgone ascertaining the quality of predictions given by the existing software reliability models using this data. We have included the raw data in Section 5 of this report so that other researchers may use the data for modeling purposes. The following paragraphs highlight some of its limitations as well as some suggestions for further analysis.

Estimation with Few Data Points

It can be argued that the total number of errors observed in each of these programs is too small to use for obtaining predictions from some of the existing reliability models. This argument is problematic for two reasons. First, if we want to estimate the reliability of the system using failure data on the component modules, we may not be able to obtain enough observations of failures due to the module sizes. (Other studies [7] have reported a positive correlation with module size and number of errors observed). This type of estimation is important for reconfigurable systems constructed from program libraries of certified software and for flight-control systems which tend to have software components of small size. A second problem with the above argument is the time under test required to collect data for accurately estimating the desired reliability. Miller has indicated that to assure a very long MTTF requires at least an order of magnitude longer time on test. [20] We suggest that the existing models be extended or new models be developed based on small samples or that methods of accelerating error detection be explored.

TABLE 9. CONFIDENCE INTERVALS FOR MEAN NUMBER OF INPUT CASES

Code	Fix	Min	m_j	Max	98% C.I.	
					Lower	Upper
AT1	1	1	1.04	2	0	2
	2	1	1.40	3	0	3
	3	3	41.49	167	26	75
	4	4	454.55	1476	300	850
	5	1	57.47	373	36	110
	6	9	113.64	387	75	210
	7	7	181.82	669	115	325
	8	132	5000.00	13721	29994	105263
AT2	1	1	1.56	6	1	5
AT3	1	1	1.36	6	0	3
	5	4	97.09	324	60	175
	6	4	43.67	136	28	85
	8	4	97.09	324	61	180
	9	3	43.67	136	28	85
	10	3	43.67	136	28	85
	11	3	43.67	136	38	85
	12	1	22.52	42	13	42
	13	4	68.97	324	43	125
	14	3	43.67	136	28	85
	15	3	43.67	136	28	85
	16	3	71.94	253	45	130
	17	4	2500.00	6635	1500	5263
	18	76	1250.00	3971	800	2632

Proportional Hazards Model

We had two reasons for collecting descriptive information on the programmer activity during development. First, this information characterizes the experiment. Second, we intended to explore the use of this information as covariates in a proportional hazards model as did Nagel and Skrivan. [4] The proportional hazards model is of the form

$$\lambda(t;z) = \lambda_0(t)e^{-z\beta}$$

where $\lambda_0(t)$ is an unspecified baseline hazard function, and z is a vector of covariates or explanatory variables which act multiplicatively on the hazard function. [21] Using the proportional hazards model, however, is not tractable due to the small sample size. (Early in the experiment design, we considered having six programmers code one problem instead of three programmers coding two problems. Managing six programmers would have limited our ability to closely supervise the experiment. As a result using three programmers postpones the analysis of covariates until more experimental subjects are added).

7. CONCLUDING REMARKS

7.1. Results and Observations

The results obtained corroborate those found in the Boeing study. The program error rates exhibited a log-linear trend and the individual errors occurred with significantly different rates.

While the experiment can be viewed as a case study performed in a laboratory setting, the emulation of a realistic software development process increases confidence in the application of the conclusions resulting from the experiment in a broader context, i.e., real world projects. The following paragraphs describe some of the lessons which have been learned from this experiment.

The sequential development of small scale experiments, such as this one, appears more desirable than the one step investment in a large scale, large sample study. This lesson resulted from several problems surfacing during the conduct of the experiment. Cost is also a constraining factor affecting the number and character of the N-versions to be developed, the number and skill level of the programmers and data analyses to be generated.

One of the more vexing of the problems which arose was the difficulty of assuring that the data itself was collected without error, ambiguities, and inconsistencies. This problem was vexing because the execution control system had been developed to alleviate this problem. Fortunately, an interface between the system and the experimenters had also been developed and assisted in identifying and resolving data anomalies. This problem has identified the need for even greater automation of the system i.e. not just for efficiency of operation but also for raising confidence in the integrity of the raw data.

An additional problem involved the initial granularity in the definition of software errors (as related to program fixes and program faults) which contributed to some loss of information about the failure process. The initial definition was refined and resulted in additional insight into the failure process, in particular dependencies among faults.

7.2. Research Directions

A sequence of smaller scale experiments coordinated among a set of collaborating researchers appears to be the preferred mode of research. This preference is a result of the extensive cost of collecting software failure data for reliability modeling and the nature of unanticipated problems which occur during the conduct of this type of experiment. Experiment data integrity requires a much greater extent of automation than one would anticipate. Finally, while it would not be sensible to advocate more complex statistical techniques where simpler ones suffice, it appears that software reliability assessment demands computationally intensive techniques and that simpler ones will not suffice. Two possible modeling approaches are mentioned below.

Dependencies stemming from the order in which faults occurred and were fixed on different replications may be important to modeling the software failure process. The development of a competing risk model which used this information may be worthwhile pursuing. This type of model is based on viewing the data as multivariate failure time data and the problem to be solved as a competing risk problem. [21] This approach involves estimating the failure rates for certain faults given the removal of some or all other faults. Solving this problem necessitates the assumption that data under one set of study conditions in which n faults are operative is similar to a different set of study conditions in which only certain faults are operative. The failure rate function for a given fault is affected by removal of other faults. That is, computation of the risks associated with some failures must be conditioned on the occurrence and removal of a masking fault. Kalbfleish and Prentice categorize this problem as largely nonstatistical. They indicate:

> It is unrealistic to think that general statistical methods can be put forward to estimate failure rates under the removal of other causes. A good deal of knowledge of the physical or biological mechanisms giving rise to the failures as well as knowledge of the mechanism giving rise to the removal of certain failure types is necessary before reasonable methods can be proposed in any given setting. p.166.

This statement suggests that we should further investigate the underlying mechanisms which give rise to these software failures.

Another approach is modeling based on the data domain instead of the time domain. Development of a data domain model is computationally intensive, involving a formal characterization of the input space and the consideration of the usage distribution as a walk through that space. Although not straightforward, the functional transformation of data domain estimates into time-based reliability figures seems feasible.

8.0 BIBLIOGRAPHY

References

1. J. R. Dunham and J. C. Knight, eds., "Production of Reliable Flight Crucial Software: Validation Method Research for Fault-Tolerant Avionics and Control Systems Sub-Working-Group Meeting," NASA Conference Publication 2222 (1982).
2. Algirdas Avizienis, "Fault Tolerance: The Survival Attribute of Digital Systems," *Proceedings of the IEEE* **66**(10)(October 1978).
3. Thomas Anderson and P. A. Lee, *Fault Tolerance: Principles and Practice,* Prentice Hall, London (1981).
4. Phyllis M. Nagel and James A. Skrivan, "Software Reliability: Repetitive Run Experimentation and Modeling," NASA CR-165836 (February 1982).
5. G. E. Migneault, "Emulation Applied to Reliability Analysis of Reconfigurable, Highly Reliable, Fault-Tolerant Computing Systems," *AGARD Conference Proceeding,* (1980).
6. Frederick P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering,* Addison-Wesley Publishing Company, Reading, Massachusetts (1975).
7. Victor R. Basili and David M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," Computer Science Technical Report Series, TR-1235, University of Maryland, College Park, Maryland (December 1982).
8. B. A. Sheil, "The Psychological Study of Programming," *ACM Computing Surveys* **13**(1) pp. 101-120 (March 1981).
9. Thomas Moher and G. Michael Schneider, "Methods for Improving Controlled Experimentation in Software Engineering," Technical report 80-8, Computer Science Department, University of Minnesota (February 1980).
10. Pieter Eykhoff, *System Identification Parameter and State Estimation,* John Wiley and Sons, New York (1974).
11. J. R. Brown and H. N. Buchanan, "TRW SDP 1776," TRW Systems Group, Redondo Beach, California (1973).
12. W. Gellert, H. Kustner, M. Hellwich, and H. Kastner, eds., *The VNR Concise Encyclopedia of Mathematics,* Van Nosdrant Reinhold Company, New York (1977).
13. *Reference Manual,* IMSL Library, Edition 9, (June 1982).
14. G. S. Fishman and L. R. Moore, "Empirical Testing of Multiplicative Congruential Generators with Modulus (2**31) - 1.," Curriculum in Operations Research and Systems Analysis Technical Report 77-12, Curriculum in Operations Research and Systems Analysis University of North Carolian at Chapel Hill (1977).
15. William F. Ingogly, "NASA Software Engineering Experiment VAX/VMS Command Language Primer," RTI Research Document under NASA Contract NAS1-16489, Research Triangle Institute (June 1983).
16. R.K. Scott, J.N. Gault, D.F. McAllister, and J. Wiggs, "Experimental Validation of Six Fault-Tolerant Software Reliability Models," *Digest of Papers FTCS-14* **IEEE Computer society Press Order No. 540** pp. 102-107 (1984).

17. B. Littlewood, P.A. Keiller, D.R. Miller, and A. Sofer, *On the Quality of Software Reliability Prediction Electronic Systems Effectiveness and Life Cycle Costing* , Spring-Verlag, Berlin (1983).

18. D.R. Cox and P.A.W. Lewis, and J.K. Skwirzinski, e.d., V.R. Basili, and T.Y. Phillips, *Evaluating and Comparing Software Metrics in the Software Engineering Laboratory,* Springer-Verlag, New York (1983).

19. Kermig G. Clems, *Confidence Limits in the Case of the Geometric Distribution,* Biometrika (1959).

20. Douglas R. Miller, "Some Statistical Issues in Assurance of Very Highly Reliable Systems ," IEEE Computer Society Workshop on Laboratories for Reliable Systems Research Abstract (April 1983).

21. J. D. Kalbfleisch and R. L. Prentice, *The Statistical Analysis of Failure Time Data,* John Wiley and Sons, New York (1980).

Appendix A
Launch Interceptor Conditions (LIC)

1.0 LAUNCH INTERCEPTOR CONDITIONS (LIC)

Conditions were specified in such a way that the resulting program would be similar to a Site Defense program attempting to correlate radar tracking returns. Nineteen parameters were required as input to precisely specify these conditions. The Launch Interceptor Conditions (LIC) were defined as follows:

1) Any two consecutive data points are a distance greater than the length, ℓ, apart.

2) Any three consecutive data points cannot all be contained within or on a circle of radius r.

3) Any three consecutive data points form an angle, a, where a $< (\pi - \epsilon_2)$ or a $> (\pi + \epsilon_2)$. Being measured here is the angle a's deviation from 180 degrees. The second of the three consecutive points is always at the vertex of the angle.

4) Any three consecutive data points form a triangle with area greater than A. The three points are at the triangle's vertices.

5) Any M consecutive data points lie in more than Q quadrants. Where there is ambiguity as to which quadrant contains a given point, priority of decision will be by quadrant number, i.e., I, II, III, IV. For example, the data point (0,0) is in quadrant I. Also, the point (-1,0) is in quadrant II. The point (0,-1) is in quadrant III.

6) For any two consecutive data points, P_1 and P_2, the difference of their abscissas is negative, i.e., $(X_2 - X_1) < 0$.

7) At least one of any N consecutive data points lies a distance greater than ϵ_1 from the line joining the first and last of these points.

8) Any two data points (with n_1 consecutive intervening points) are a distance greater than the length, ℓ, apart.

9) Any three data points (with n_2 and m_2 consecutive intervening points, respectively) cannot be contained within or on a circle of radius r.

10) Any three data points (with n_3 and m_3 consecutive intervening points, respectively) form an angle, α, where $\alpha < (\pi - \epsilon_2)$ or $\alpha > (\pi + \epsilon_2)$. Being measured here is the angle α's deviation from 180 degrees. Of the above first mentioned three data points. the second is always at the vertex of the angle.

11) Any three data points (with n_4 and m_4 consecutive intervening points, respectively) form a triangle with area greater than A. The above first mentioned three data points are at the triangle's vertices.

12) For any two data points, P_1 and P_2 (with n_6 consecutive intervening points) the difference of their abscissas is negative, i.e., $(X_2 - X_1) < 0$.

13) Any two data points, with n_1 consecutive intervening points, are a distance greater than the length, ℓ, apart. Also, any two data points (which can be the same or different from the above first mentioned two data points), with n_1 consecutive intervening points, are a distance less than the length, L, apart.

14) Any three data points, with n_2 and m_2 consecutive intervening points, respectively, cannot be contained within or on a circle of radius r. Also, any three data points (which can be the same or different from the above first mentioned three data points), with n_2 and m_2 consecutive intervening points, respectively, can be contained in or on a circle of radius R.

15) Any three data points, with n_4 and m_4 consecutive intervening points, respectively, form a triangle with area greater than A. The above first mentioned three data points are at the triangle's vertices. Also, any three data points (which can be the same or different from the above first mentioned three data points), with n_4 and m_4 consecutive intervening points, respectively, form a triangle with area less than E. The above second mentioned three data points are at the (second) triangle's vertices.

2.0 PROBLEM LOGIC

1) Information was supplied indicating the logical connectors among all the LIC, as defined in Section 1.0. The format was a symmetrical square matrix where zero indicated NOT used, one indicated the OR connector between two conditions and two indicated the AND connector. The matrix was identified as the Logical Connector Matrix (LCM).

2) Part of the output data was a column matrix with resultant information as to whether or not the LIC were met, i.e., for each condition, zero meant the condition was not met and one meant it was met. The identification for this matrix was Conditions Met Matrix (CMM).

3) Preliminary unlocking information was generated. By definition, these were criteria which determined whether or not interceptors would be launched. These data were determined by the interaction of the LCM and CMM matrices to form the Preliminary Unlocking Matrix (PUM). Definitions of the matrix elements indicate how the two matrices interact to form PUM. The diagonal elements of PUM were input according to the desired or required unlocking sequence, i.e., a one indicated that the corresponding LIC was to be considered as a factor in signaling interceptor launch and a zero meant that it was not a factor. Non-diagonal elements were determined by the LCM operating as a Boolean operator, as defined in Section 2.0.1, on the operand CMM.

4) The Final Unlocking Matrix (FUM) was generated by having the PUM diagonal operate on non-diagonal elements. An element in the FUM was one (1) if:

The corresponding PUM diagonal element was zero (0), indicating no interest in the associated LIC; or

The corresponding PUM diagonal element was one (1) and all other elements in that diagonal element's row were one (1).

An element in the FUM represented the final conclusion with respect to its corresponding LIC.

5) In order to launch an interceptor, all elements in FUM had to be equal to one. In this case, the message "NOW" was generated and output to the printer, together with a lsting of all input data values. The information from all matrices was printed. The output was in matrix format for ease of interpretation.

3.0 DATA INFORMATION

1) Pairs of values for the rectangular coordinates (x, y) represented data points.

2) An input data set contained a maximum of 100 ordered data points.

3) P = number of data points in a data set.

4) The input data constants, as defined in Section 1.0 were specified for each input data set.

5) Restrictions on the input parameters were as follows:

$P > 2,\ \ell \geq 0,\ r \geq 0,\ 0 \leq \epsilon_2 < \pi,\quad A \geq 0,\ M \leq P,\ 1 \leq Q \leq 3,$

$\epsilon_1 \geq 0,\ L \geq 0,\ R \geq 0,\ E \geq 0.$

6) The Logical Connector Matrix (LCM) element values were given as input.

7) The Preliminary Unlocking Matrix (PUM) diagonal element values were given.

For the actual data values, see the example matrices in the following section.

4.0 EXAMPLE MATRICES

The following matrices are a model of the problem logic, as defined in Section 2.0.

Logical Connnector Matrix (LCM) (Input)

*	1	2	3	4	5 . . . 15
1	2	2	1	2	0 . . . 0
2	2	2	1	1	0 . . . 0
3	1	1	2	1	0 . . . 0
4	2	1	1	2	0 . . . 0
5	0	0	0	0	0 . . . 0
.	.	.	.	.	. . . .
.	.	.	.	.	. . . .
.	.	.	.	.	. . . .
15	0	0	0	0	0 . . . 0

*, Launch Interceptor Conditions (LIC)

Since we have zeros beyond the fourth LIC, the 5th through the 15th LIC are not to be considered in this example.

Definition - L_{ij} is the ij^{th} element in the LCM.

Conditions Met Matrix (CMM) (Output)

Condition	Value
1	0
2	1
3	1
4	0
5	0
.	.
.	.
.	.
15	0

Definition: C_i is the i^{th} element in the CMM.

The C_i are computed output, but in order to illustrate this example, we are arbitrarily setting these elements in the CMM.

Preliminary Unlocking Matrix (PUM)

(Output, non-diagonal elements)
(Input, diagonal elements)

LIC	1	2	3	4	5 . . .	15
1	1	0	1	0	1 . . .	1
2	0	0	1	1	1 . . .	1
3	1	1	1	1	1 . . .	1
4	0	1	1	0	1 . . .	1
5	1	1	1	1	0 . . .	1
.	.	.	.	.	. .	.
.	.	.	.	.	. .	.
.	.	.	.	.	. .	.
15	.	.	.	.	. .	0

Furthermore, defining the ij^th^ element in the PUM as P_{ij}, we have the following: $P_{11} = P_{33} = 1$ and all other P_{ii} (i.e., the diagonal elements) are zero. This means that only the first and third LIC are required in the unlocking sequence. Note that these are input values.

$P_{12} = 0$ since, $L_{12} = 2$, signifying the AND condition for C_1 and C_2 which are zero and one, respectively, i.e., 01 = 0.

$P_{13} = 1$ since, $L_{13} = 1$, signifyng the OR condition for C_1 and C_3 which are zero and one, respectively, i.e., 0 + 1 = 1.

$P_{14} = 0$ since, $L_{14} = 2$, signifying the AND condition for C_1 and C_4 which are both zero, i.e., 00 = 0.

$P_{15} = 1$ since $L_{15} = 0$, signifying the Not Used condition for C_1 and C_5. The above examples show how to generate the P_{ij} values.

Final Unlocking Matrix (FUM) (Output)

LIC	VALUE
1	0
2	1
3	1
4	1
5	1
.	.
.	.
.	.
15	1

Definition: F_i is the i^{th} element in the FUM.

$F_1 = 0$ since $P_{11} = 1$ and $P_{12} = P_{14} = 0$, i.e., the diagonal value is one and there is at least one zero element in the first row of PUM.

$F_2 = 1$, since $P_{22} = 0$.

$F_3 = 1$, since $P_{33} = 1$ and $P_{31} = P_{32} = P_{34} = P_{35} = \ldots = P_{3,15} = 1$.

$F_4 = 1$, since $P_{44} = 0$.

$F_5 = F_6 = \ldots = F_{15} = 1$, since, $P_{55} = P_{66} = \ldots = P_{15,15} = 0$, respectively.

Since there is a zero element in FUM, ($F_1 = 0$), the launch interceptor condition is not met.

5.0 SUPPLEMENTARY INFORMATION

1. The program will be written in FORTRAN on the BITS system.
2. No double precision or complex variables are required.
3. Your program will be a subroutine.
4. Assume the inputs are in labeled common, i.e., COMMON/INPUTS/ X(100), Y(100), EL, ... using the order in Section 6.0. You are free to use your own variable names, however.
5. Outputs will be in labeled common, i.e., COMMON/OUTPTS/CMM(15), ... using the order in Section 7.0. Again you are free to use your own variable names.
6. Use the IFOUT flag to control printing. Code the output statements, but branch around them if IFOUT = 1.

7. When the first and last points of N consecutive data points are identical, then the calculated distance to compare with ϵ (LIC #7), will be that distance from the coincident point to all others of the N consecutive points.

6.0 INPUTS

1. Data Points (x_i, y_i) $i=1, \ldots, P$ $2 \leq P \leq 100$

   accurate to one decimal place real

2. Nineteen Parameters

1.	l	real
2.	r	"
3.	ϵ_2	"
4.	A	"
5.	M	integer
6.	Q	"
7.	ϵ_1	real
8.	n_1	integer
9.	n_2	"
10.	m_2	"
11.	n_3	"
12.	m_3	"
13.	n_4	"
14.	m_4	"
15.	n_6	"
16.	L	real
17.	R	real
18.	E	real
19.	N	integer

3.	<u>LCM Array</u>	$LCM_{i,j}$	i=1,. . .,15
			j=1,. . .,15
				integer

4.	PUM array <u>Diagonal Terms</u>	$PUM_{i,i}$	i=1,. . .,15	integer
5.	P - number of data points			integer

6. IFOUT - Controlling output, i.e., = 0 Program prints output
= 1 Program prints <u>no</u> output
integer

7.0 OUTPUTS

1.	<u>Conditions Met Matrix</u>	CMM_i,	i=1,-,15	integer
2.	<u>Final Unlocking Matrix</u>	FUM_i,	i=1,-,15	integer

3. <u>"LAUNCH" or "NO LAUNCH"</u>

(Use a logical variable which is true for launch and false for no launch).

4. Preliminary unlocking matrix PUM_i. i=1, -, 15 integer.

Appendix B
Input Generation Scheme for the Launch Interruptor Condition Problem

(X,Y) Coordinates

The (X,Y) Coordinates describe a radar track. Ninety-five percent of these (X,Y) coordinates are uniformly distributed within the region bounded by the lines connecting the points (5,0), (0,5), (-5,0), and (0,-5). Five percent of the (X,Y) coordates generated are uniformly distributed within the combined regions bounded by the lines connecting the points (20,20), (40,20), (40,40), and (20,40) in each of the four quadrants.

LCM Elements

The 225 elements of the LCM were generated according to the following specification:

Let the LCM be defined as a symmetric matrix $\{ L_{ij} \}$

where

i is the row index

j is the column index such that $i \leq j$

then

$L_{ij} \leq 2$ for all i

$j = 1,2,3,4,5,6,7,8,9,10,11$ and

$i \leq j$, and

$\Pr \{ L_{ij} = 0 \} = .78$

$\Pr \{ L_{ij} = 1 \} = .20$ and

$\Pr \{ L_{ij} = 2 \} = .02$.

PUM Elements

The fifteen diagonal elements of the PUM were generated according to the following specification:

Let the PUM be defined as the matrix $\{ P_{ij} \}$

where

i is the row index and

j is the column index,

then

$\Pr \{ P_{ij} = 0 \} = 0.5$, and

$\Pr \{ P_{ij} = 1 \} = 0.5$

for all $i,j = 1,2,\ldots,15$ and $i=j$.

LIC Parameters

All of the Launch Interceptor Conditions had parameters associated with them. The values of these parameters are:

l=14.5
r=7.1
$\epsilon_2 = 0.5$
A=50
M=4
Q=3
N=4
$\epsilon_1 = 15.0$
L=10
R=5
E=25
$n_1 = n_2 = n_3 = n_4 = 1$
$n_6 = 5$
$m_2 = m_3 = m_4 = 0$

<u>Input Data Modification</u>

1. P was generated according to a UNIFORM (2,100) distribution.
2. The parameter M was changed so Launch Interceptor Condition 5 which specifies that "M consecutive data points lie in more than Q quadrants" could be met with Q=3. If M <= Q, this condition would never be met.
3. The parameter L was changed so Launch Interceptor Condition 13 which specifies "any two data points ... are a distance less than the length L apart" can met with non-identical points.
4. The parameter R was changed so that Launch Interceptor Condition 14 which specifies that "any three data points ... can be contained in or on a circle of radius R" can be met with non-identical points.
5. The parameter E was changed so Launch Interceptor Condition 15 which specifies "Also, any three data points ... form a triangle with area less than E" could be met with non-identical points.
6. The parameter ϵ_2 was changed so that Launch Interceptor Condition 3 which specifies "Any three data points form an angle, a, where $a < (\pi - \epsilon_2)$ or $a > (\pi + \epsilon_2)$. " would not always be met.
7. The parameter m_1 was discarded since it is not used by any of the conditions.

Appendix C
Seeds for the Pseudo-Random Number Generator

REPLICATION	SEED
1.0	1436099972
2.0	0744513290
2.01	0744513290
2.1	0744513290
2.2	0744513290
3.0	1751976185
3.1	1751976185
3.2	1751976185
3.3	1751976185
4.0	0031162681
5.0	1914684009
6.0	0957636397
7.0	1523183247
7.1	1523183247
7.2	1523183247
8.0	0255979618
9.0	0412542844
10.0	0366320340
11.0	1776255921
12.0	1863079069
13.0	1949058887
14.0	0095717758
15.0	0084721889
16.0	1670222942
16.1	1670222942
16.2	1670222942
17.0	1282766561
17.1	1282766561
17.2	1282766561
18.0	1937632417
19.0	0246022436
20.0	1927677362

Appendix D
Failure Data

TABLE 10

INPUT CASES TO FAILURE FOR APPLICATION TASK 1

	Fix Number										
Rep	1	2	3	4	5	6	7	8	9	10	11
1	1	1	43	1476	151	101	204	2986	---	---	---
2	1	1	33	228	24	161	177	893	---	---	---
3	1	1	69	540	78	280	371	4969	---	---	---
4	1	1	19	277	48	48	25	3036	---	---	---
5	1	1	116	514	9	34	364	1376	---	---	---
6	1	3	84	56	46	60	7	132	---	---	---
7	2	1	38	413	4	239	257	---	---	---	---
8	1	1	14	617	193	15	175	1933	---	---	---
9	1	1	27	149	7	219	51	---	---	---	---
10	1	2	7	1102	14	143	338	344	---	---	---
11	1	2	3	589	64	273	469	3466	---	---	---
12	1	1	7	188	87	142	106	3905	---	---	---
13	1	2	21	319	5	387	42	6606	---	---	4160
14	1	1	3	1078	61	111	150	1900	---	---	---
15	1	2	17	4	14	24	317	5235	---	---	---
16	1	2	32	13	4	117	8	4165	---	---	197
17	1	1	30	454	373	50	20	4813	---	---	---
18	1	1	167	980	81	113	16	---	---	---	---
19	1	1	58	25	37	24	25	3721	---	---	---
20	1	1	44	105	15	15	200	8177	---	---	---
21	1	1	5	57	22	9	67	1326	---	---	---
22	1	1	16	679	73	126	99	2879	---	---	---
23	1	2	97	1132	1	52	133	4600	---	---	---
24	1	1	80	483	21	53	217	936	---	---	---
25	1	3	8	111	3	37	689	4475	8359	---	---

Key

Table shows input cases to failure from start of replication for designated fix.

--- Indicates that error did not occur and fix was not applied during the replication.

TABLE 11

INPUT CASES TO FAILURE FOR APPLICATION TASK 2

Rep	Fix Number	Rep	Fix Number
1	1	14	1
2	1	15	1
3	3	16	6
4	1	17	1
5	1	18	1
6	1	19	1
7	1	20	1
8	1	21	1
9	1	22	1
10	1	23	2
11	1	24	1
12	4	25	3
13	2		

Key

Table shows input cases to failure from start of replication for designated fix.

--- Indicates that error did not occur and fix was not applied during the replication.

TABLE 12

INPUT CASES TO FAILURE FOR APPLICATION TASK 3

	Fix Number																			
Rep	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	---	*	*	91	5	*	91	5	5	5	5	5	5	5	1	2642	76	---	---
2	1	91	*	*	151	136	*	151	136	136	136	33	151	136	136	10	4202	2352	---	---
3	3	22	*	*	142	22	*	142	22	22	22	22	22	22	22	142	2890	569	---	---
4	1	---	*	*	21	21	*	21	21	21	21	7	7	21	21	24	1132	324	---	---
5	1	---	*	*	49	20	*	49	20	20	20	20	20	20	20	253	228	4553	---	---
6	1	---	*	*	11	11	*	11	11	11	11	11	11	11	11	71	383	287	---	---
7	1	1	*	*	50	50	*	50	50	50	50	10	50	50	50	74	642	2533	---	---
8	1	47	*	*	301	53	*	301	53	53	53	31	301	53	53	56	1542	587	---	---
9	1	21	*	*	149	51	*	149	51	51	51	51	149	51	51	40	1757	1432	---	---
10	1	---	*	*	59	16	*	59	16	16	16	16	16	16	16	79	4488	751	---	---
11	1	---	*	*	49	17	*	49	17	17	17	17	17	17	17	57	2158	243	---	---
12	1	---	*	*	83	41	*	83	41	41	41	41	41	41	41	3	4803	625	---	---
13	1	---	*	*	58	58	*	58	58	58	58	42	42	58	58	54	4160	3971	4160	---
14	1	---	*	*	134	118	*	134	118	118	118	39	39	118	118	111	---	238	---	---
15	1	---	*	*	4	4	*	4	4	4	4	4	4	4	4	39	508	508	---	---
16	6	---	*	*	183	13	*	183	13	13	13	8	8	13	13	222	2138	1323	---	---
17	1	11	*	*	36	36	*	36	36	36	36	35	36	36	36	79	6635	837	---	---
18	1	15	*	*	324	134	*	324	134	134	134	15	324	134	134	44	4402	1143	---	---
19	1	15	*	*	112	15	*	112	15	15	15	15	112	15	15	17	343	1244	---	---
20	1	40	*	*	92	92	*	92	92	92	92	40	92	92	92	20	4	2582	---	---
21	1	28	*	*	57	57	*	57	57	57	57	28	57	57	57	126	227	227	---	---
22	1	8	*	*	101	57	*	101	57	57	57	37	101	57	57	62	3421	1264	---	---
23	1	3	*	*	100	3	*	100	3	3	3	3	100	3	3	109	1136	594	---	---
24	1	32	*	*	43	42	*	43	42	42	42	32	43	42	42	54	2993	499	---	---
25	3	1	*	*	21	21	*	21	21	21	21	1	21	21	21	104	5554	979	---	---

Key

Table shows input cases to failure from start of replication for designated fix.

--- Indicates that error did not occur and fix was not applied during the replication.

*** Fix numbers 3, 4, and 7 should be ignored. There are no corresponding fixes and faults. These fix numbers were mistakenly added to the data. They have been kept in the Table merely to keep the fix numbering consistent with the raw data files.

1. Report No. NASA CR-172553	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle AN EXPERIMENT IN SOFTWARE RELIABILITY		5. Report Date March 1985
		6. Performing Organization Code
7. Author(s) Janet R. Dunham and John L. Pierce		8. Performing Organization Report No. 412U-2094-12 & 14
9. Performing Organization Name and Address Research Triangle Institute P. O. Box 12194 Research Triangle Park, NC 27709		10. Work Unit No.
		11. Contract or Grant No. NAS1-16489
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: G. Earle Migneault
Final Report

16. Abstract

The report documents the results of a software reliability experiment conducted in a controlled laboratory setting. The experiment was undertaken to gather data on software failures and is one in a series of experiments being pursued by the Fault Tolerant Systems Branch of NASA - Langley Research Center to find a means of credibly performing reliability evaluations of flight control software.

The experiment tests a small sample of implementations of radar tracking software having ultra-reliability requirements and uses n-version programming for error detection, and repetitive run modeling for failure and fault rate estimation. The experiment results agree with those of Nagel and Skrivan in that the program error rates suggest an approximate log-linear pattern and the individual faults occurred with significantly different error rates.

Additional analysis of the experimental data raises new questions concerning the phenomenon of interacting faults. This phenomenon may provide one explanation for software reliability decay.

17. Key Words (Suggested by Author(s)) software reliability fault-tolerant software software engineering	18. Distribution Statement Unclassified—Unlimited

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 60	22. Price

For sale by the National Technical Information Service, Springfield, Virginia 22161