NASA CR-172,378

NASA Contractor Report 172378

Software Reliability: Additional Investigations Into Modeling with Replicated Experiments

P. M. Nagel, F. W. Scholz and J. A. Skrivan

Boeing Computer Services
Space and Military Applications Division
Seattle, Washington 98124

Contract No. NAS1-16481
June 1984

NASA
National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23665

SOFTWARE RELIABILITY: ADDITIONAL INVESTIGATIONS INTO MODELING WITH REPLICATED EXPERIMENTS

Prepared Under Contract NASI-16481

By

P. M. Nagel
F. W. Scholz
J. A. Skrivan

Boeing Computer Services
Systems Division
Seattle, Washington 98124

For

National Aeronautics and Space Administration

June 1984

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

1.0 SUMMARY AND INTRODUCTION

1.1 SUMMARY

This report documents the second of two studies performed by Boeing Computer Services on modeling the process of software error detection from the results of experiments specifically designed to complement this activity. The experiments consist of simulations conducted on code prepared under controlled conditions and executed with randomly selected inputs. Six codes were developed in the first study and this study continues the experiment with six more. The code is initialized to an original state and flexed with independently generated random inputs. Errors are corrected as they are encountered until a stopping rule is satisfied. Replication is introduced by repeating the entire process from initialization.

This experiment has confirmed several of the conclusions of the first study with regard to the probabilistic diversity of error structure, linearity of the log stage failure rate, and the nearly exponential character of the stage distribution.

The previous study explored the effects of programmer and problem as experimental design factors on the error structure. The current study enlarges this set of factors by varying the experience level of the programmer and the relative frequency or usage of the program inputs. The use of FORTRAN is contrasted with the use of a micro-based assembler language as another design factor. All of these factors, not surprisingly, affected performance and some very tentative relational hypotheses are suggested.

Although it can be demonstrated that stages are neither independent nor exponentially distributed, empirical estimates show that the exponential assumption is nearly valid for all but the extreme tails of the distribution. Empirical studies of the dependence of a stage on its past indicate that some of the estimated correlations are high and exhibit a curious periodicity. Except for the degree of dependence in the stage probabilities, it still appears that Cox's proportional hazards model, introduced in reference [1], approximates to a degree what is being observed.

An analytic framework for replicated and non-replicated (i.e., traditional) software experiments is initiated in this study in order to present the results in a meaningful context. A method of obtaining an upper bound on the error rate of the next error is proposed. The method was validated empirically by comparing forecasts based on the method with actual data. In all 14 cases the bound exceeded the observed parameter, albeit somewhat conservatively. Two other forecasting methods are proposed. One based on a crude approximation to the proportional hazards model proved to be in the neighborhood of the estimated parameter, as measured by a 95% confidence interval, 53% of the time. The other subtracted the observed error probability and the program's success rate from one to estimate the remaining error rate. This method generally underestimated the observed parameter but was within the interval 67% of the time.

1.2 INTRODUCTION

Software reliability forecasting continues to be an elusive problem. Many models have been introduced in the literature, most of them without a clear statement of the mathematical and statistical foundations that motivated the model and most of them without data validation. Many statements have stressed the need for realism in data collection and modeling. Unfortunately by enforcing realism, particularly in data collection, control is sacrificed, i.e., so many variables are influencing quality that it is difficult to identify the drivers causing change. Risk assessment has been confused with software management goals and testing has been confused with debugging.

This experiment and the previous companion study documented in reference [1] represent an attempt to wed experimental results and a modeling framework. The study has four primary goals:

a. To produce quality data on software error detection with controlled experiments,

b. To model the exact nature of the experiment as carefully as possible, bowing to mathematical expediency only when necessary and after as much empirical evaluation as possible,

c. To experimentally verify some of the more popular assumptions regarding the process of error detection, and

d. To provide measures of the impact of various environmental factors on software quality by varying them as design factors in the experiment.

The previous study defines an experimental framework with all of the controls thought necessary for quality data. This experiment does not try to investigate all of the issues that influence the software error structure but concentrates only on those that are related to problems with fixed specifications and with code prepared by a single programmer. An error is carefully defined in terms of a detector or criteria establishing correctness, and the probability of detection is governed completely by the set of inputs producing incorrectness and the probability distribution controlling the usage of the input set.

Little software reliability growth modeling has been based on feedback gained from controlled experiments. This is particularly relevant in forecasting the future behavior of a program that necessarily depends on a model of the error removal process. Since programming is a human endeavor, without a behavior model only controlled experiments can provide this knowledge. The empirical studies reported on in this document suggest that there is a commonality in the methods employed by programmers to achieve their professional goals and that the accuracy of the performance is related to some of the characteristics of the human producer. Overall these studies also suggest that the error structure of software is a forecastable process.

The remainder of the report is in six sections. Section 2.0 briefly reviews the features of the overall experiment and describes the experimental factors explored in the current study. Section 3.0 provides some operational detail of conducting the experiment with a description of the programmers and problems used and the data collected. In Section 4.0, descriptive statistics of the data observed in this study are presented and compared with the previous work. Analytic results are discussed in Section 5.0, together with an empirical study on forecasts. The conclusions of the study are summarized in Section 6.0.

2.0 EXPERIMENT

Details of the replicated experiments that form the core of the experimental results obtained in this study and its predecessor are contained in reference [1] and therefore this section only describes the broad framework of the experiment. The experiments are simulations performed on software written to support a given problem specification by a programmer with designated qualifications. Both the qualifications of the programmer and the specification of the problems are selected with a predetermined experimental set of factors in mind.

2.1 TEST FRAMEWORK

The simulations are initiated by generating random input according to a distribution called the usage distribution defined as part of the original problem specification. The problems for this and the earlier study were selected from problems that had been previously programmed and had been in use for some time. The output of this seasoned program, executed with identical inputs to the program on test, is used as a comparator to determine the correctness of the new program. Complementing this error detection mechanism is the error detection capability of the operating system itself used as a companion detector.

Once correctness according to the detectors used has been established, new inputs are generated independently and the process repeated. If for some execution an error is indicated, the error is recorded together with the number of executions since the last error, and then corrected.

The simulation begins with the program in its initial state. This state is reached when the program successfully complies and correctly executes a number of predetermined test cases. These tests are defined as the static tests for a given specification and the program must pass these static tests as well as successfully execute the input causing failure before simulation can be reinitiated. Once reinitiated, the process is repeated error by error until a predetermined stopping rule is satisfied. Termination in this study occurs when an error is detected that is too costly to fix or an upper bound on the number of samples is reached, whichever comes first.

Traditional tests on software force the experimenter to forecast the future from a single manifestation of the error process, that is, a single series of program executions, with the errors corrected in a single order. In order to provide statistical replication of this process, in this experiment once the stopping rule is satisfied for a particular program run the program is reinitiated to its original state and the process, i.e., a new run is repeated. The experimental flow for each run is exactly the same except for the consequences of using different random numbers. Each run has the opportunity then, of generating different random errors in different orders with different life lengths between errors. Fifty runs are simulated for each experimental treatment examined.

Two concepts associated with this sampling are defined: program stage and program state. Since errors can be recognized from run to run, they are identified as encountered with an error number. This number is recorded as part of the data base when the error shows up in the sampling. A program stage at a particular point in the sampling refers to the number of errors that have been corrected since sampling began for that run. A program state is a listing of errors by number that have been detected and corrected since the run was initiated.

Figure 2.1-1 illustrates the overall flow of this experiment for a given code.

2.2 DESIGN FACTORS

The test framework introduced in the previous section was used in the first study to explore several combinations of problem specification and programmer. This study continues this investigation by exploring new features. These features are combined with old features and new data is combined with old data in order to investigate many new issues.

The experiments conducted are summarized in Figure 2.2-1. A factorial design of two programmers (A and B) each programming from three separate problem specifications (1, 2 and 3) formed the nucleus of the first study. To this have been added three programmers (C, D and E) and one new problem specification (4). From the resulting 5 x 4 array only those combinations indicated were selected for coding.

Programmers C and D are highly experienced programmers each with a strong background in technical programming extending over several years as opposed to the relative inexperience of programmers A and B. Programmer E has little more work experience than B, but is considered a senior programmer in terms of job performance and the quality of the experience. In coding problem #4, program design skills are emphasized more than the combined analysis/design skills called for in coding the other three problems. Programs A3 and B3 have been rerun with a different usage distribution. Programmers C and D reprogrammed problem #3 in the Z8000 assembler language of the ONYX micro computer. FORTRAN is the language common to all of the other programs written for this study. Programmers D and E each programmed problem #1 and the new problem in FORTRAN.

It was the intent of the original design, that programmer C would assume the role subsequently played by programmer E. Unfortunately, though not discovered until nearly the end of the coding of C3, this programmer was undergoing a difficult period of personal stress. As this condition was influencing the quality of the work being performed, a substitution was necessary for the remaining work in the design. The data for C3, though it is included in this report for consistency, should be considered with some caution, particularly when compared across treatments in the design.

2.3 DATA RECORD

For each test condition, i.e., programmer-problem combination, and for each run, the number of executions until failure for each stage is recorded together with the error number of the error causing failure. In some cases, when a clock with sufficient sensitivity was available for the machine in use at the time, the total time between failures of the program was also recorded. The total data base recorded for this experiment is presented in Appendices D through K.

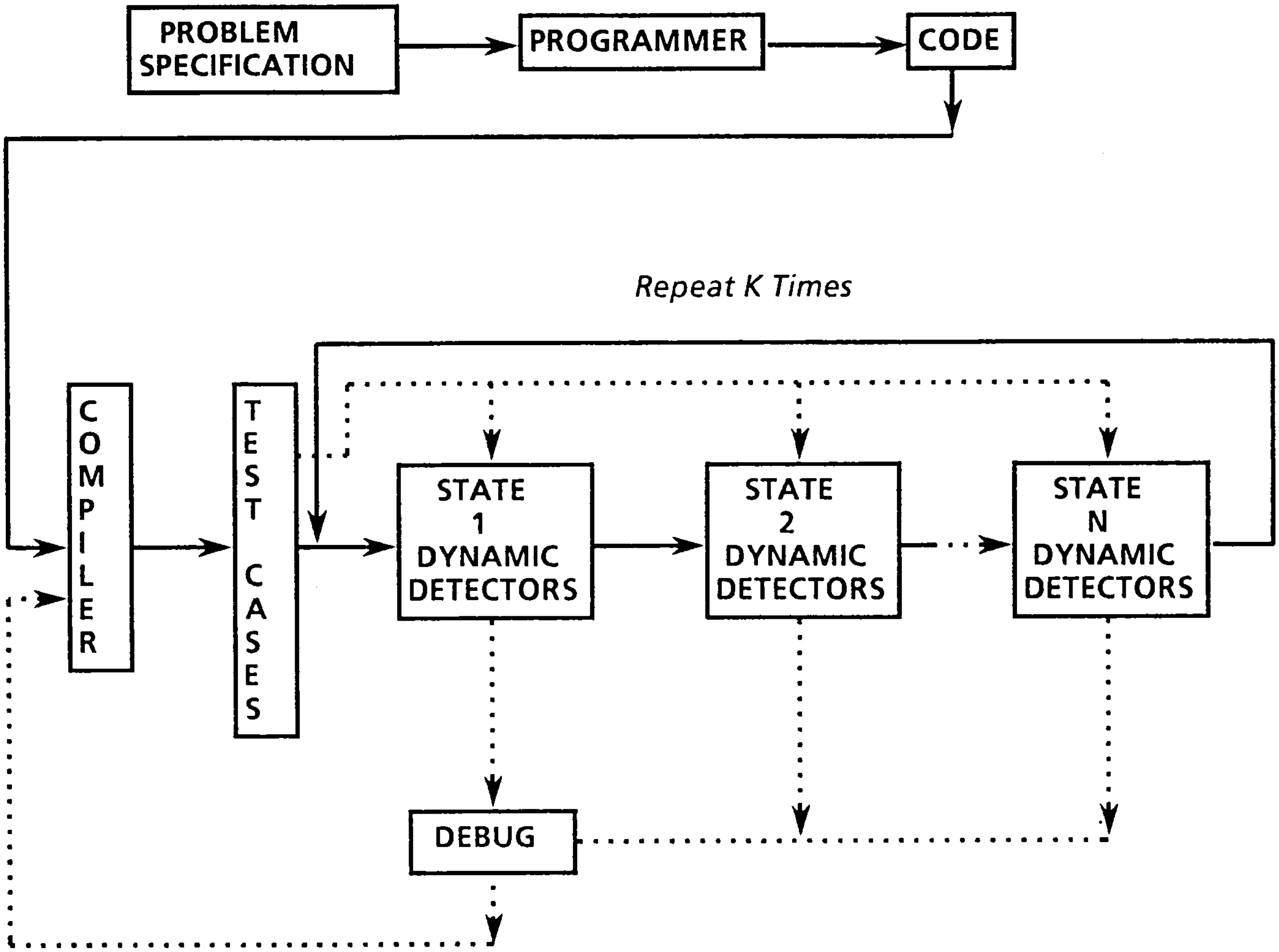

PROBLEM SPECIFICATION
PROGRAMMER
CODE
Repeat K Times
COMPILER
TEST CASES
STATE 1 DYNAMIC DETECTORS
STATE 2 DYNAMIC DETECTORS
STATE N DYNAMIC DETECTORS
DEBUG

Figure 2.1-1. Experiment Flow Diagram.

Program

Programmer	1	2	3	4
A	A1/F	A2/F	A3/F A3*/F	
B	B1/F	B2/F	B3/F B3*/F	
C			C3/A	
D	D1/F		D3/A	D4/F
E	E1/F			E4/F

Study #1

A1, B1
A2, B2
A3, B3

Study #2

A3*, B3*
C3, D3
D1, E1
D4, E4

Design Conditions

F = FORTRAN
A = Z8000 Assembler
* = Different Usage Distribution

Figure 2.2-1. Experimental Design Matrix.

3.0 EXPERIMENT DATA COLLECTION

3.1 INTRODUCTION

The gathering of statistics on the failure detection/error correction process involves embedding the subject programs in a software-test environment. The overhead programs of the experiment as well as operational details are fully described in reference [1].

In the current study, the DEC VAX/VMS System was extensively used for 3 of the 4 problems. The remaining problem was run on the ONYX Microcomputer, a Z8000-based system with the UNIX^{TM} operating system. Z8000 Assembly Language was used in this latter problem and FORTRAN was used in the other problems.

This chapter and the referenced appendices present the data-collection results of the experiment. Also included is a brief description of the programmers' backgrounds.

There are four parallel sections (3.3, 3.4, 3.5, and 3.6) corresponding to the four problems of the experiment. Each of these sections together with corresponding appendices and material in reference [1] where appropriate, gives background information on the problem specifications and correct version, descriptions of test cases and the usage distribution of the experiment runs.

A tabulation of software errors is given for each subject program. The identified errors are categorized using the categories in reference [2]. These categories are given in Appendix A.

UNIX is a trademark of the Bell Laboratories.

3.2 PROGRAMMER DESCRIPTIONS

Programs from five programmers (labeled A, B, C, D, and E) have been used in the current study. Programmers A and B were also involved in the previous work [1].

Programmer A received a B.S. degree in Computer Science in 1979, and joined BCS in June 1979 as a programmer. The principal job of this programmer has been to support and enhance a geometry package used to design wing and body configurations. Past experience in the field of computer science has emphasized structured software design, development and languages, including FORTRAN, Pascal, ALGOL, SNOBOL and COBOL.

Programmer B received a B.S. degree in Computer Science in 1975, and joined BCS in January 1976 as a programmer. B has worked on nuclear-waste engineering and radiation-monitoring problems, using FORTRAN on a variety of machines. Later assignments have involved integration testing on the AWACS program using JOVIAL language, and conversion of a missile-simulation program and graphics package from and IBM machine to the VAX/VMS system.

Programmer C received a B.S. in Physics in 1967 and a Ph.D in Astronomy in 1979 and has had extensive experience in both scientific and systems work. Scientific applications have included submarine avionics and navigation algorithms. The experience in systems work has involved executive systems, I/0 interfaces and real-time data acquisition. C is familiar with CDC, DEC and IBM mainframes plus FORTRAN, CMS-2 and a variety of assembly languages.

Programmer D graduated in 1962 with a B.S. in Mathematics and in 1964 with an M.S. in Math and currently has over 20 years experience in systems and software development. This experience has included participation in large-scale applications scientific software development and extensive research in software validation and verification technology. D is familiar with a variety of host mainframes, FORTRAN, PASCAL and assembly languages.

Programmer E received a B.S. degree in Computer Science in 1979 and joined BCS shortly thereafter as a software engineer. Assignments have included design and development of a variety of software tools, including graphics software, Data Manipulation Language precompiler, PL/1 dynamic analyzer, HAL/S static and dynamic analyzer, and a state-of-the-art symbolic execution tool for HAL/S. Language proficiencies include Pascal, FORTRAN and Basic, and a working knowledge of HAL/S, Lisp, SNOBOL, Simula and APL. E has worked on CDC and DEC mainframes.

E was chosen to participate in the study as a senior programmer, based not on years of experience alone, but more importantly on versatility; E has had much more experience with a variety of languages compared to programmers A and B. In addition, E's applications have also been more varied than those of A and B, ranging from scientific programming to compiler construction. However, maturity based on years of experience is also a part of one's seniority, which E is still increasing. Consequently, the study findings with regard to the effects of seniority may be clouded by the difficult question of what constitutes seniority in computer programming.

3.3 PROBLEM #1

3.3.1 Background

The problem used in Problem #3 of reference [1] was used in the present study to consider the effects of a different usage distribution. That program involved earth-satellite calculations for which programmers A and B designed, coded and tested their individual programs, A3 and B3, respectively. Those same programs were used in the present study but with another usage distribution of the input data. They are labelled A3* and B3*, respectively, to differentiate findings. See reference [1] for program specifications, test cases, and a description of the correct version.

3.3.2 Usage Distribution

Three latitude-longitude coordinates on the Earth are required inputs, as well as an angle between 0^o and 180^o. In the previous study, the distribution for the latitude-longitude coordinates was uniform over the sphere, but rounded to the nearest 5^o in both latitude and longitude. The distribution for the angle was uniform between 0^o and 180^o with no rounding.

The distribution for the three latitude-longitude coordinates has been changed to allow more latitudes at or near the equator. Figure 3.3.2-1 illustrates the triangular distribution centered at the equator (0^{o} latitude) used for latitudes. Longitude coordinates are still uniform over the sphere, and both latitude and longitude coordinates are rounded to the nearest 5^{o}. The continuous curve in Figure 3.3.2-1 approximates the discrete curve resulting from latitude rounding. The distribution for the angle remains uniform between 0^{o} and 180^{o} with no rounding.

3.3.3 Error Descriptions

The same errors occurring for B3* and all errors but No. 5 in A3* in reference [1] were also detected and corrected in the present study. Subject program A3* had ten software failures, for which six errors were corrected. Subject program B3* had seven different software failures for which five software errors were corrected. (Note that in sections 4.5.6 and 4.5.7 of reference [1], the descriptions of A3 and B3 are transposed.)

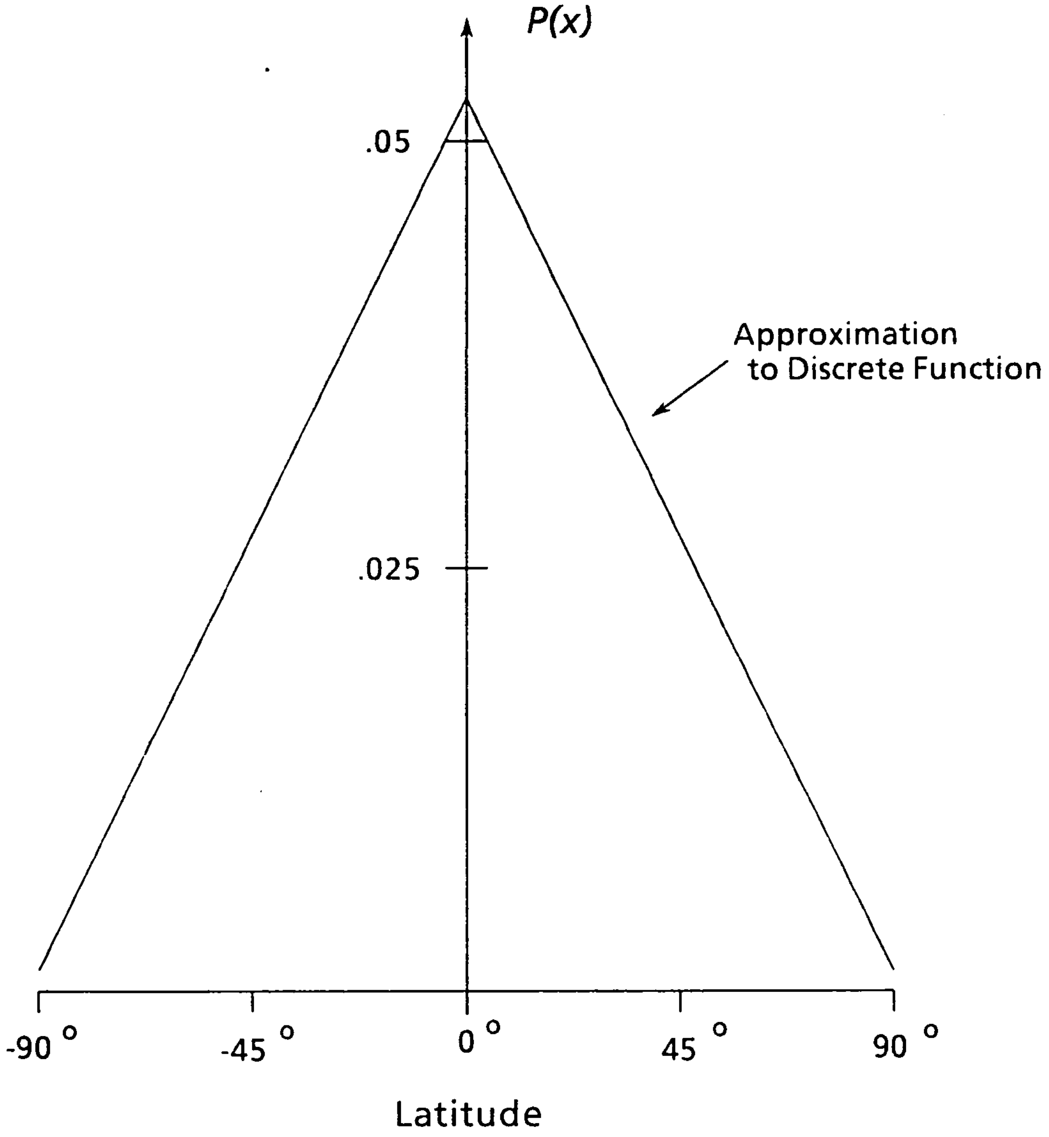


Figure 3.3.2-1. Usage Distribution for Latitude Coordinates.

3.3.3.1 Subject Program A3*

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A800	The determination of the sign of the azimuth was not done.
2	A600	The algorithm to determine the order of the two intersection points was incorrect.
3	A900	The argument for arccos was greater than 1.0 or less than -1.0.
4	A600	The algorithm to determine intersections failed to find a correct intersection point.
5	A900	The argument for arcsin and/or arccos was greater than 1.0 or less than -1.0.
6	A600	The sign of the calculated azimuth was incorrect, when the magnitude of the azimuth is pi.
7	A600	Determination of colinearity of two coordinates and the center of the earth was incorrect.
8	A600	There was an accuracy failure in some output item (relative error >1%).
9	A600	There was division by zero when determining intersections.
10	A600	The azimuth was incorrectly calculated as 0, when the correct value was pi.

3.3.3.2 Subject Program B3*

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A600	The determination of the sign of the azimuth was incorrect.
2	A800	There were uninitialized variables when cross-product calculations were bypassed under certain conditions.
3	A900	The argument for arccos was greater than 1.0 or less than -1.0.
4	A600	The algorithm to determine intersections calculated the wrong point of intersection.
5	A600	The azimuth was incorrectly calculated when the path went through either the north or south pole.
6	A600	The algorithm for calculating intersections failed to determine the correct number of intersections.
7	A600	There was an accuracy failure in some output item (relative error > 1%).

3.3.4 Run Results

3.3.4.1 Subject Program A3*

Figure 3.3.4.1-1 presents the results of the experiment for subject program A3*. This figure, as well as those for the remaining programs (3.3.4.2-1, 3.4.3.1-1, 3.4.3.2-1, 3.5.3.1-1, 3.5.3.2-1, and 3.6.7-1), traces the 50 runs for the particular program.

The figure is composed of levels, or stages, of program states, where each stage is defined by the number of errors detected. Beginning with state 0, the occurring program states and their frequencies are shown for the 50 runs. The encircled number(s) represent a program state, in particular, the error numbers of the corrected errors. For example, 12 is a given subject program with errors #1 and #2 corrected. A subject program at state 0 is identified with 0 following the program name and dash, e.g., A3*-0. The directed line segments represent the random walk of the subject program going from one state to another, i.e., having one or more errors corrected. The number to the left of this line segment is the number of runs experiencing that particular change in state.

For example, using Figure 3.3.4.1-1 and beginning with state 0 (A3*-0), 47 of the runs experienced error #1 first and errors #2, #7 and #9 were initially experienced by a single run each. As shown in the table to the right of the figure, these 50 runs required a total of 187 input cases for the first error(s) to occur.

From these states the runs continue to another stage. Note that not all 50 runs continue through all stages, because some errors detected are not corrected. In general, the number of input cases per run required to detect errors increases as the stage increases.

Because of the complexity of the figure, multiple failures on the same case are not indicated in the figure, (as was included in similar figures in reference [1]), but instead the trace follows the error numbers of the multiple failure in sequence.

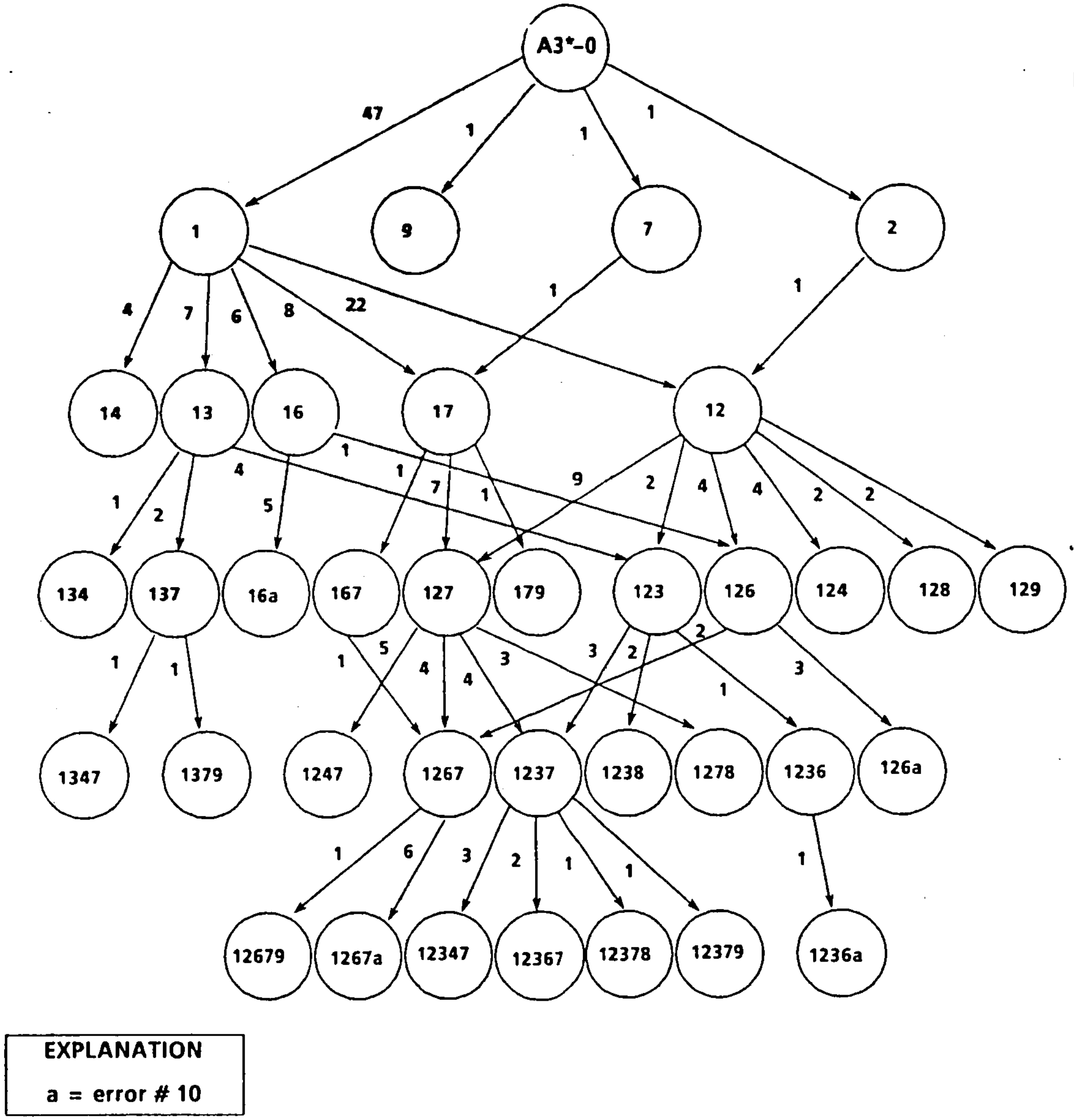
3.3.4.2 Subject Program B3*

Figure 3.3.4.2-1 illustrates the result of the software failure/error correction process for B3*. Similiarly to Figure 3.3.4.1-1, multiple failures on the same case are not shown in Figure 3.3.4.2-1.

3.4 PROBLEM #2

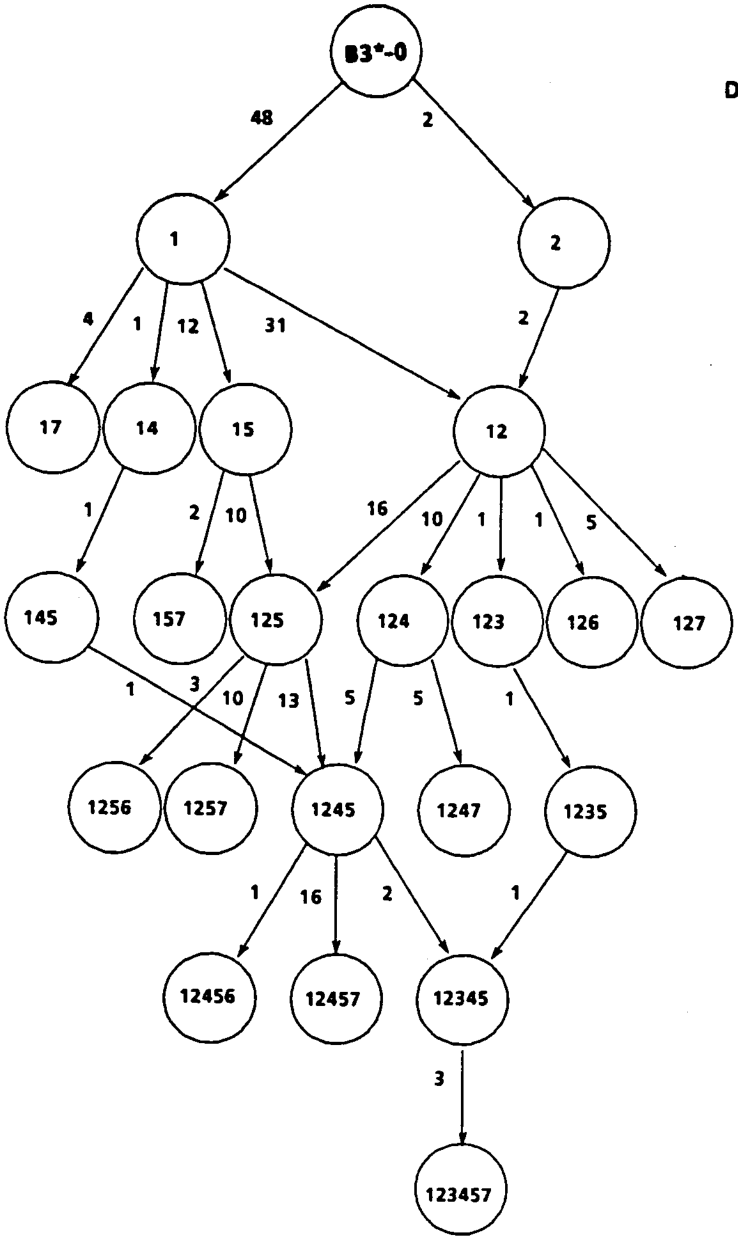
3.4.1 Background

The program for Problem #3 of reference [1] served as the basis for Problem #2 of this study to consider the effects of computer language. Programmers C and D designed and coded their respective programs in Z8000 Assembly Language based on the program specifications from the first study. The subject programs are labelled C3 and D3, respectively. The same test cases and usage distribution from that study were used again. See reference [1] for program specifications, test cases, usage distribution, and a description of the correct version.



ERRORS DETECTED	RUNS	TOTAL CASES
1	50	187
2	49	1521
3	45	1667
4	30	2086
5	15	964

Figure 3.3.4.1-1. Trace of Runs for Subject Program A3*.



ERRORS DETECTED	RUNS	TOTAL CASES
1	50	127
2	50	2459
3	46	4852
4	38	7137
5	20	6703
6	3	1373

Figure 3.3.4.2-1. Trace of Runs for Subject Program B3*.

3.4.2 Error Descriptions

During the experiment for Problem #2, subject program C3 had 13 software, all of which were corrected. Subject program D3 had 15 failures with 13 corresponding corrected software errors.

3.4.2.1 Subject Program C3

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A800	There were spurious or missing intercepts.
2	A600	The computed longitude of the intercept was off by a factor of 2 pi.
3	A600	The computed latitude of the intercept was off by a factor of pi.
4	A600	The algorithm to determine the order of the two intercepts was incorrect.
5	A600	The sign of the calculated azimuth was incorrect when the magnitude of the azimuth was pi.
6	A600	An accuracy failure caused a refinement of error No. 2.
7	A600	The azimuth was incorrectly calculated when the path went through either the North or South Pole.
8	A800	Intercepts were incorrectly calculated for coincident or antipodal points.
9	A600	A refinement of error No. 7 was necessary to correctly calculate intercepts for North Pole cases.
10	A800	Intercept cases were incorrect when all three points lay on a great circle.
11	A600	A refinement of error No. 2 was necessary.
12	A600	The computed longitude was close to 2 pi instead of 0.
13	A600	A further refinement of error No. 2 was necessary.

3.4.2.2 Subject Program D3

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A600	The determination of the sign of the azimuth was incorrect.
2	A800	The calculated intercepts were incorrect due to inadequate checking of quadratic solutions.
3	A600	The azimuth was incorrect by a multiple of pi.
4	A600	Calculated intercepts were incorrect due to wrong testing of point-solution angles.
5	A600	The algorithm to determine the order of the two intersection points was incorrect.
6	A600	There was a division by zero when determining intercepts.
7	A600	The azimuth was incorrectly calculated when the path went through either the North or South Pole.
8	A600	A refinement of error No. 6 was necessary.
9	A600	An azimuth of -pi was calculated when the correct value was +pi.
10	A600	A refinement of error No. 8 was necessary.
11	A600	There was an accuracy failure in some output item (relative error > 1%).
12	A800	A "tangent point" intercept was not detected.
13	A600	A refinement of error No. 7 was necessary.
14	A600	There was an incorrect intercept of (0, pi) when 3 longitude coordinates were identical.
15	A600	There was an incorrect intercept in which a calculated longitude was 2 pi instead of pi.

3.4.3 Run Results

3.4.3.1 Subject Program C3

Figure 3.4.3.1-1 shows the results of the software failure/error correction process for C3. Because of the complexity of the figure, multiple failures on the same case are not shown, but instead the trace follows the error numbers of the multiple failure in sequence.

3.4.3.2 Subject Program D3

Figure 3.4.3.2-1 shows the corresponding results of the experiment for program D3. As was the case in Figure 3.4.3.1-1, multiple failures on the same case are not shown.

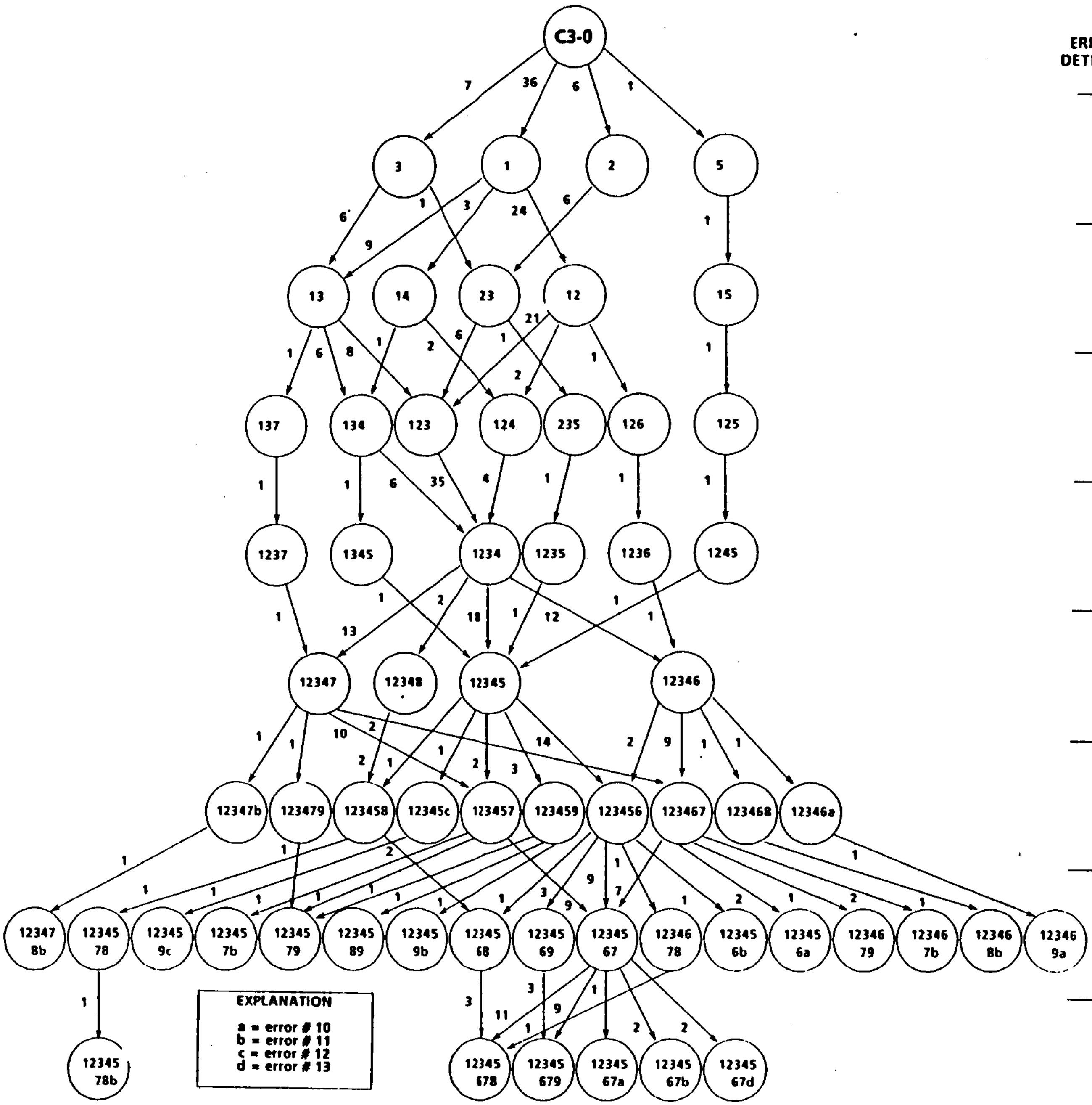
3.5 PROBLEM #3

3.5.1 Background

The program in Problem #1 of reference [1] served as the basis for Problem #3 of the current study. The original program is a missile-tracking simulation described in reference [2]. Two senior-level FORTRAN programmers, D and E, were given the specifications of the program. They then designed, coded and tested their own versions. Their programs are designated D1 and E1 with reference to Problem #1 of the original study. See reference [1] for program specifications, test cases, usage distribution and a description of the correct version.

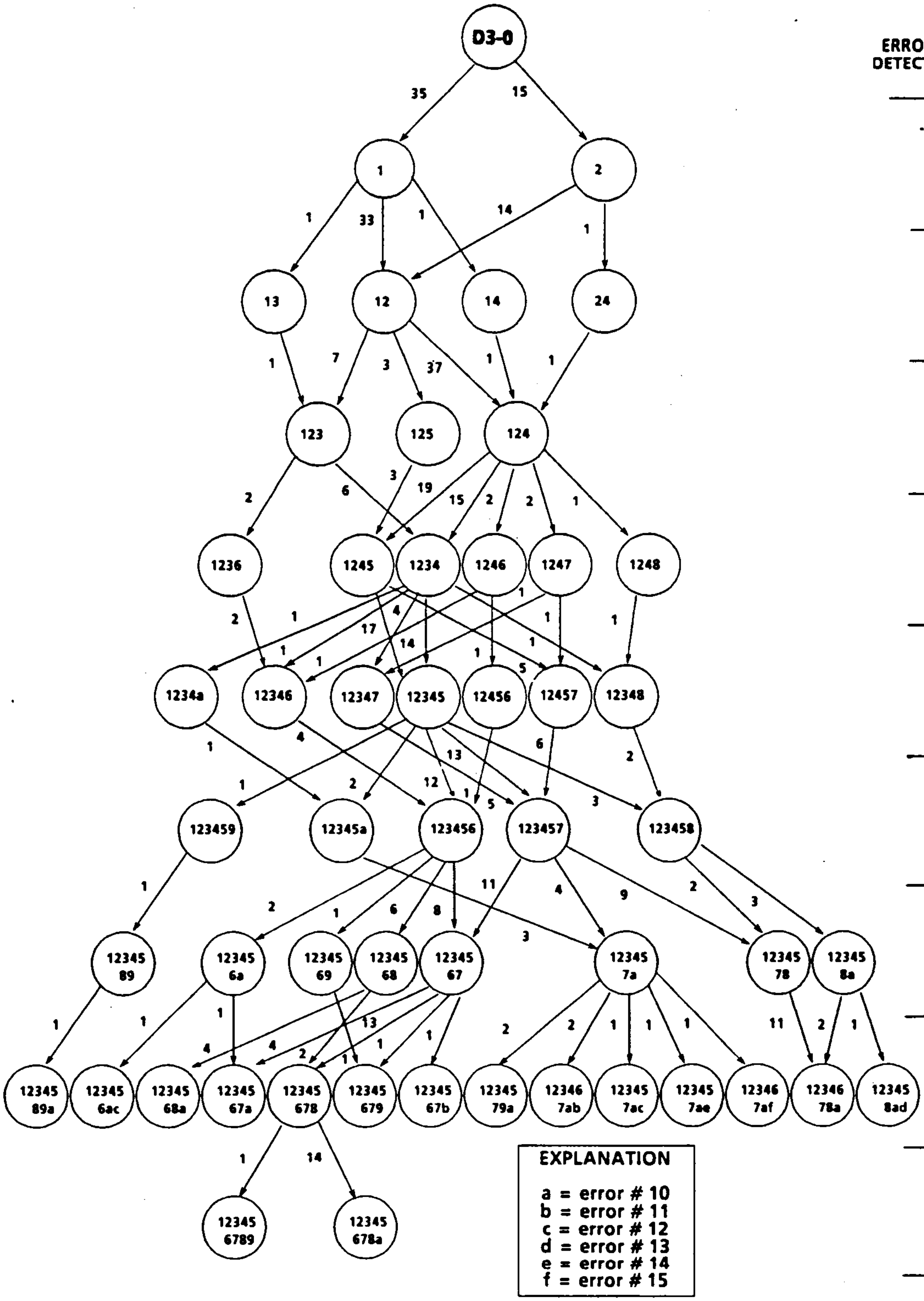
3.5.2 Error Descriptions

In the course of the experiment for Problem #3, subject program D1 had 4 detected software failures and corresponding corrected errors. Subject program E1 had 6 detected failures, for which there were 6 corresponding errors corrected.



ERRORS DETECTED	RUNS	TOTAL CASES
1	50	112
2	50	86
3	50	89
4	50	198
5	50	4720
6	50	6763
7	50	12,459
8	33	24,404

Figure 3.4.3.1-1. Trace of Runs for Subject Program C3.



ERRORS DETECTED	RUNS	TOTAL CASES
1	50	71
2	50	106
3	50	1257
4	50	1742
5	50	5010
6	50	23,024
7	50	45,542
8	50	153,045
9	15	19,838

Figure 3.4.3.2-1. Trace of Runs for Subject Program D3.

3.5.2.1 Subject Program D1

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A900	The argument for square root was a very small negative number.
2	A900	The argument for arccos was greater than 1.0 or less than -1.0.
3	A900	The argument for arccos was greater than 1.0 or less than -1.0.
4	A900	The argument for square root was a very small number.

3.5.2.2 Subject Program E1

ERROR NUMBER	CLASSIFICATION CODE	DESCRIPTION
1	A600	An incorrect equation was used to calculate the area of a triangle.
2	A600	An incorrect algorithm was used to calculate the angles of a triangle.
3	A600	An incorrect equation was used to calculate the area of a triangle.
4	A600	Wrong subscripts were used in the algorithm to determine point coverage by a circle.
5	A900	The argument for arccos was greater than 1.0 or less than -1.0.
6	A600	There was a division by zero.

3.5.3 Run Results

3.5.3.1 Subject Program D1

Figure 3.5.3.1-1 presents the results of the software failure/error correction process for D1.

3.5.3.2 Subject Program E1

Figure 3.5.3.2-1 illustrates the results of the software failure/error correction process for E1.

3.6 PROBLEM #4

3.6.1 Background

Algorithms of the Association for Computing Machinery (ACM) are a collection of published and extensively tested routines used primarily for scientific applications. Algorithm 479, A Minimal Spanning Tree Clustering Method [3], was chosen as the correct version for Problem #4. This FORTRAN subroutine determines cluster-membership of two-dimensional coordinates based on several cluster-description parameters. Programmers D and E designed and coded programs D4 and E4, respectively, to test against this correct version.

3.6.2 Specifications

Cluster-membership determination is based on first constructing a minimal spanning tree connecting all of the input two-dimensional coordinates. Dijkstra [4] presents an efficient algorithm for this construction. Zahn [5] then uses this spanning tree to determine cluster membership. Specifications were provided to the programmers, based on these latter two references. The complete specifications for Problem #4 are presented in Appendix B.

3.6.3 Test Cases

Two test cases were used to bring both subject programs D4 and E4 to state 0. The test cases and corresponding correct output are presented in Appendix C.

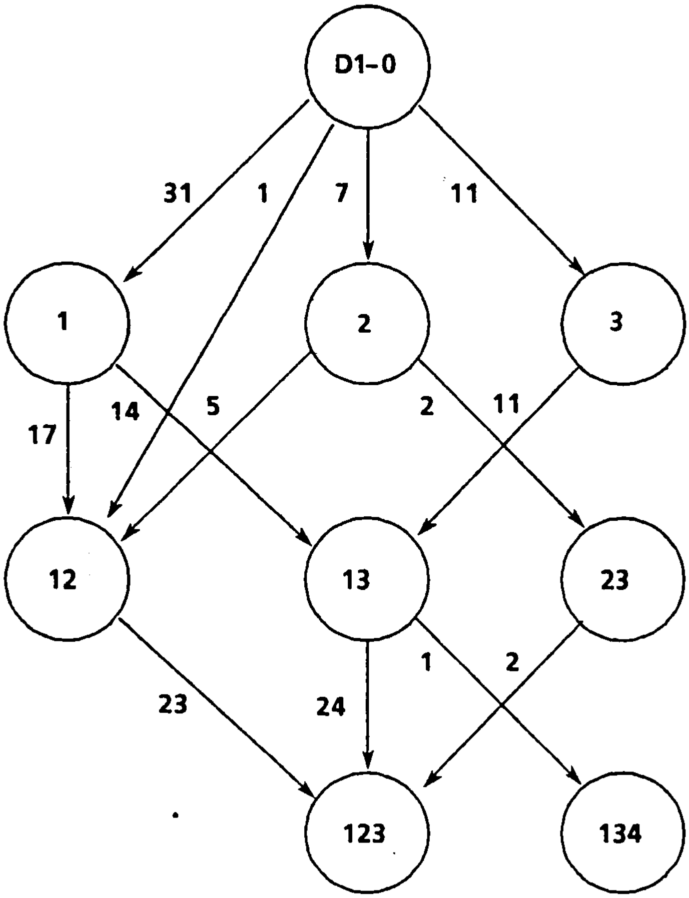
3.6.4 Usage Distribution

The usage distribution was designed to simulate randomly occurring clusters ("shotgun shots") each with a random number of points ("pellets").

The coordinates of the points are generated for each input case in the following manner:

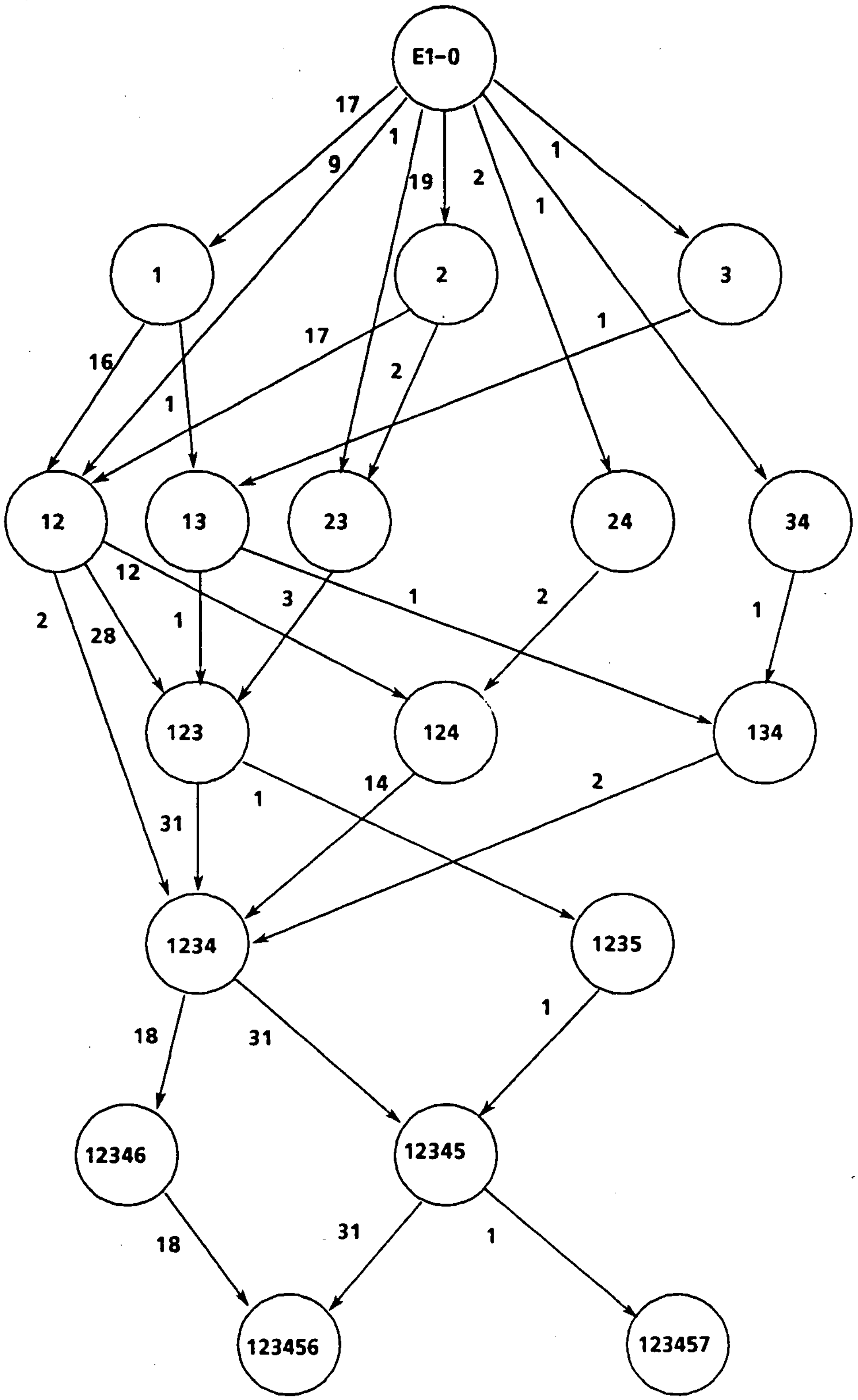
1. Pick n, the number of clusters, uniformly in [1, 2, 3, 4, 5].

2. For each cluster, choose a center from the distribution shown in Figure 3.6.4-1.



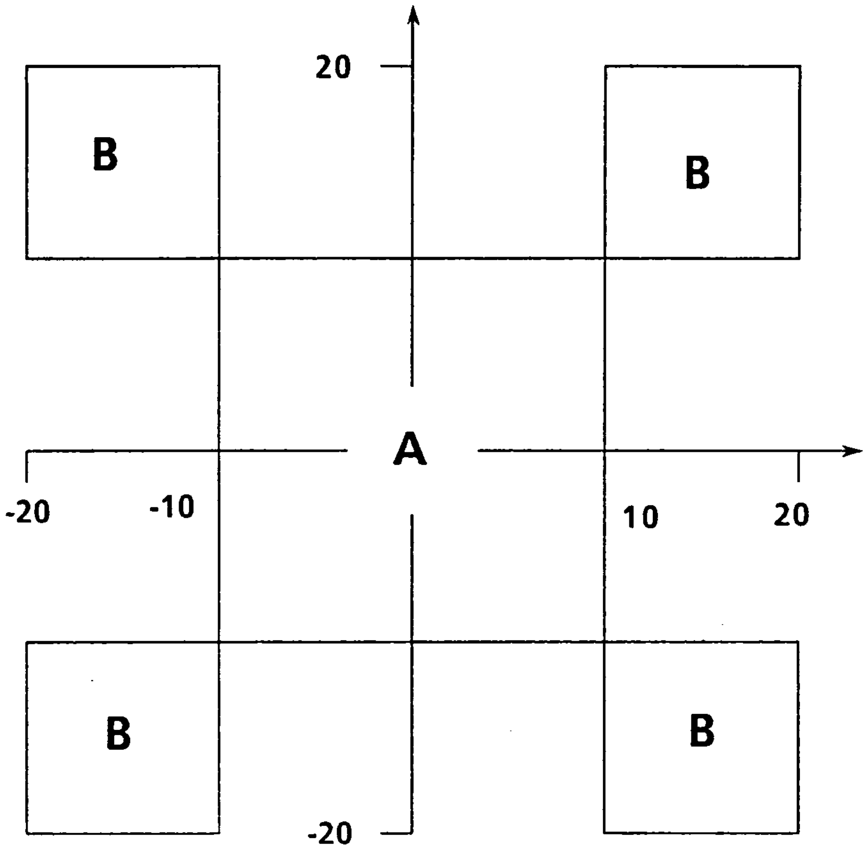
ERRORS DECECTED	RUNS	TOTAL CASES
1	50	38,126
2	50	88,243
3	50	147,941

Figure 3.5.3.1-1. Trace of Runs for Subject Program D1.



ERRORS DECECTED	RUNS	TOTAL CASES
1	50	65
2	50	72
3	50	497
4	50	794
5	50	65,507
6	50	109,512

Figure 3.5.3.2-1. Trace of Runs for Subject Program E1.



P(A) = 75%, uniformly distributed within A

P(B) = 25%, uniformly distributed within B

Figure 3.6.4-1. Usage Distribution for Cluster Centers.

3. For each center, choose a radius R, uniformly in [1,5].

4. Pick m, the number of points for each cluster, uniformly in [1,2,3,4,5].

5. Calculate the spherical coordinates (r,θ) for each point in one cluster, for r uniform in [0,R] and θ uniform in $[0,2\pi]$.

6. Calculate rectangular coordinates for all points from the spherical coordinates.

In this way, each input case has from 1 to 25 2-dimensional coordinates representing from 1-5 clusters.

The same cluster parameters (F,S and D) used in the test cases were part of the input cases described herein.

3.6.5 Correct Version

Correctness of the output was determined by composing the subject program's output with that from ACM Algorithm 479 [3].

3.6.6 Error Descriptions

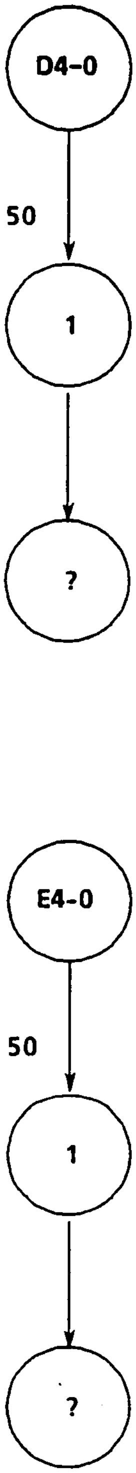
In the course of the experiment for Problem #4, both subject programs D4 and E4 had only one detected software failure, both of which were corrected.

The software error (Classification Code A800) in program D4 was the lack of provision to handle the special case of only one point in the input data.

Program E4 software error (Classification Code A600) was the use of incorrect subscripts in calculating the standard deviation of nearby branches.

3.6.7 Run Results

The results of the software failure detection/error correction process for both subject programs D4 and E4, are shown in Figure 3.6.7-1.



ERRORS DECECTED	RUNS	TOTAL CASES
1	50	1519
2	0	25,000

ERRORS DECECTED	RUNS	TOTAL CASES
1	50	285
2	0	22,000

Figure 3.6.7-1. Trace of Runs for Subject Programs D4 and E4.

4.0 DATA ANALYSIS

The data obtained in this test has been used to explore a number of possible relationships. Some of them are motivated by the need to measure and evaluate a few of the more popular assumptions regarding the probabilistic failure structure of software. Some of them are motivated by new theoretical results. Some of them are motivated by an interest in determining if programmers display any similarity in the errors made or, more significantly, in the rate at which their errors are detected.

The first study demonstrated that errors are distributed with widely varying failure probabilities and that the logarithms of these probabilities show a nearly linear decrease when plotted as a function of the number of errors corrected. These results are re-explored in the current study as well as several other issues associated with the design of this experiment.

4.1 ERROR PROBABILITIES

One of the major concerns of the original project at the time it was first proposed was a concern that the models of the day made the assumption that errors embedded in a particular code are identically distributed regardless of the mechanism guiding their identification and withdrawal. In reference [1], Table 5.2-1, experimental evidence was offered that seriously negates this assumption by every one of the six codes developed for that project.

The same is true of the current project. Estimates of the error probability for each of the errors discovered during simulation for a given code are summarized in Table 4.1-1. The estimates are computed for the i'th error by

$$\hat{p}_i = \frac{r_i}{TTT_i}$$

where r_i is the actual number of occurrences of the error in the 50 runs and TTT_i is the total time on test function. TTT_i is the sum of all the life lengths of the particular error measured from time 0, with the simulation in its initial state, to the time of occurrence of the error if it did so, or to the time the simulation was terminated for the run if it did not.

The table indicates a behavior very similar to that observed in the first study. In every situation a wide range of probabilities are observed. In one case, the range extends from 5×10^{-1} to 6×10^{-6}.

4.2 STAGE PROBABILITIES

The probability for the i'th stage has several interpretations depending on the conditional information contained in the previous i-1 stages available to the researcher. The ramifications of this remark will be discussed more completely in Section 5.0, but in this section the stage probability of interest will be considered to be conditional on the number of errors corrected prior to the initiation of the stage. Table 4.2-1 gives estimates for the stage probabilities for the experiments conducted

TABLE 4.1-1

SPECIFIC ERROR PROBABILITIES - RANKED ESTIMATES

Prog.	Error No.	No. of Failures	Error Prob. Per Execution	Prog.	Error No.	No. of Failures	Error Prob. Per Execution
A3*	1	49	$2.50x10^{-1}$	B3*	1	50	$3.85x10^{-1}$
	2	36	$1.63x10^{-2}$		2	44	$1.26x10^{-2}$
	7	25	$7.05x10^{-3}$		5	35	$4.74x10^{-3}$
	6	18	$3.23x10^{-3}$		7	45	$1.99x10^{-3}$
	4	18	$2.80x10^{-3}$		4	25	$1.79x10^{-3}$
	10	16	$2.49x10^{-3}$		6	5	$2.21x10^{-4}$
	3	13	$2.31x10^{-3}$		3	3	$1.42x10^{-4}$
	8	8	$1.25x10^{-3}$				
	9	7	$1.09x10^{-3}$				
	5	Did Not Occur					
C3	1	50	$3.29x10^{-1}$	D3	1	50	$4.67x10^{-1}$
	3	50	$2.25x10^{-1}$		2	50	$3.38x10^{-1}$
	2	50	$1.77x10^{-1}$		4	50	$2.75x10^{-2}$
	4	50	$1.14x10^{-1}$		3	50	$7.70x10^{-3}$
	5	44	$4.04x10^{-3}$		5	50	$7.04x10^{-3}$
	7	42	$2.79x10^{-3}$		7	43	$6.54x10^{-4}$
	6	40	$1.94x10^{-3}$		10	46	$2.85x10^{-4}$
	9	30	$4.76x10^{-4}$		8	34	$1.74x10^{-4}$
	8	19	$4.02x10^{-4}$		6	28	$1.69x10^{-4}$
	11	15	$2.27x10^{-4}$		9	6	$2.44x10^{-5}$
	10	5	$7.57x10^{-5}$		11	3	$1.20x10^{-5}$
	13	2	$3.03x10^{-5}$		12	2	$8.01x10^{-6}$
	12	1	$1.52x10^{-5}$		13	1	$4.01x10^{-6}$
					14	1	$4.01x10^{-6}$
					15	1	4.01x10-6

TABLE 4.1-1 (Continued)
SPECIFIC ERROR PROBABILITIES - RANKED ESTIMATES

Prog.	Error No.	No. of Failures	Error Prob. Per Execution	Prog.	Error No.	No. of Failures	Error Prob. Per Execution
D1	1	50	9.37×10^{-4}	E1	2	50	4.59×10^{-1}
	2	49	2.71×10^{-4}		1	50	4.42×10^{-1}
	3	50	2.67×10^{-4}		3	50	6.41×10^{-2}
	4	1	3.44×10^{-6}		4	50	3.30×10^{-2}
					5	50	5.04×10^{-4}
					6	49	3.51×10^{-4}
					7	1	5.81×10^{-6}
D4	1	50	3.29×10^{-2}	E4	1	50	1.75×10^{-1}
	2**	0	$< 3.77 \times 10^{-5}$		2**	0	$< 4.48 \times 10^{-5}$

** No second error encountered in 25,000(D4) and 22,000(E4) additional runs.

TABLE 4.2-1

STAGE PROBABILITIES AS A FUNCTION OF THE NUMBER OF CORRECTED ERRORS

Prog.	No. of Corrected Errors	Stage Prob. Per Execution	|ln P|	Prog.	No of Corrected Errors	Stage Prob Per Execution	|ln P|
A3*	0	$2.67\text{x}10^{-1}$	1.32	B3*	0	$3.94\text{x}10^{-1}$	0.93
	1	$3.20\text{x}10^{-2}$	3.44		1	$2.03\text{x}10^{-2}$	3.90
	2	$2.45\text{x}10^{-2}$	3.71		2	$9.31\text{x}10^{-3}$	4.68
	3	$1.41\text{x}10^{-2}$	4.27		3	$4.79\text{x}10^{-3}$	5.34
	4	$1.16\text{x}10^{-2}$	4.45		4	$2.98\text{x}10^{-3}$	5.81
	5	$2.08\text{x}10^{-2}$	3.87		5	$2.18\text{x}10^{-3}$	6.13
C3	0	$4.46\text{x}10^{-1}$	0.81	D3	0	$7.04\text{x}10^{-1}$	0.35
	1	$3.76\text{x}10^{-1}$	0.98		1	$4.07\text{x}10^{-1}$	0.90
	2	$2.84\text{x}10^{-1}$	1.26		2	$3.97\text{x}10^{-2}$	3.23
	3	$2.02\text{x}10^{-1}$	1.60		3	$2.45\text{x}10^{-2}$	3.71
	4	$1.06\text{x}10^{-2}$	4.55		4	$9.78\text{x}10^{-3}$	4.63
	5	$6.12\text{x}10^{-3}$	5.10		5	$2.17\text{x}10^{-3}$	6.13
	6	$3.54\text{x}10^{-3}$	5.64		6	$1.10\text{x}10^{-3}$	6.81
	7	$1.32\text{x}10^{-3}$	6.63		7	$3.27\text{x}10^{-4}$	8.03
	8	$8.72\text{x}10^{-3}$	7.05		8	$7.56\text{x}10^{-4}$	7.19
D1	0	$1.31\text{x}10^{-3}$	6.64	E1	0	$7.69\text{x}10^{-1}$	0.26
	1	$8.95\text{x}10^{-4}$	7.02		1	$5.62\text{x}10^{-1}$	0.58
	2	$2.53\text{x}10^{-4}$	8.28		2	$1.01\text{x}10^{-1}$	2.30
					3	$6.14\text{x}10^{-2}$	2.79
					4	$7.63\text{x}10^{-4}$	7.18
					5	$4.75\text{x}10^{-4}$	7.65
D4	0	$3.29\text{x}10^{-2}$	3.41	E4	0	$1.75\text{x}10^{-1}$	1.74
	1**	$<4.00\text{x}10^{-5}$			1**	$<4.54\text{x}10^{-5}$	

** No second error encountered.

for this study and is comparable to Table 5.3-1 of reference [1]. The entry for line i of this table estimates the conditional probability that a random execution of the program indicated will result in an error given that i-1 errors have been corrected. This probability has averaged out all of the effects due to state, that is, all of the effects due to the particular set of errors corrected and their order of correction. Since these effects do exert a random influence on this conditional probability, the actual probability is a random variable and the estimates in this table represent the rate associated with the mean number of executions to failure for the i'th stage. The first stage is based on the program in its initial state as defined by the static detectors, with all of the program's remaining errors intact.

Figure 4.2-1 is a graph of these probabilities as a function of the number of errors corrected for the programs of this study. For comparison, Figure 4.2-2 is a graph of the results obtained in the first study. More detailed comparisons will be made in later sections, but it is notable that for all their diversity the graphs of this experiment are strikingly similar to those of the previous study and exhibit a similar degree of linearity.

One of the more difficult problems in modeling from these graphs involves the definition of the origin on the horizontal axis. The reason for this difficulty is that highly probable errors indicate programs that are not well checked out. Although minimum codes are screened by the initial static program tests, the actual state of the program at this time is not controlled and is, among other variables, a function of the time spent in the programming activity or the rate at which programming is conducted. The most interesting feature of these graphs is their slope, particularly at maturity, and too many of the early, highly probable errors remaining in the codes, can exert undue influence.

Reference [1] discusses this problem in some detail and introduces an ad hoc definition of the initial error state as $\min|\ln P_i| \geq 1.0$ where P_i is the i'th stage probability. A more interesting definition might involve sliding the graphs horizontally, so that the pairwise vertical distances between observed points is optimal in some sense. Since this seemed beyond the intent of the study at this time, the data of Figure 4.2-1 was replotted in Figure 4.2-3 using the definition of reference [1].

The advantages of using this definition of origin in reference [1] seemed minimal at best and these results seem to confirm it.

4.3 EFFECT OF USAGE DISTRIBUTION ON ERROR RATE

One of the issues of interest in this experiment was a comparison of the effect of using two quite different usage distributions on error detection. The hypothesis had been proposed that the more uniformly inputs are selected the more linear the graph of the log rate parameter. Of the specifications described in reference [1], problem 3 had the most uniform requirement on input usage and therefore was selected as a candidate for exploring this hypothesis.

Two new experiments were run: one based on the code labelled A3 of reference [1] and the other on code B3. The new non-uniformly weighted usage distribution is described in Section 3. Tables 4.3-1 and 4.3-2 compare the results of the four

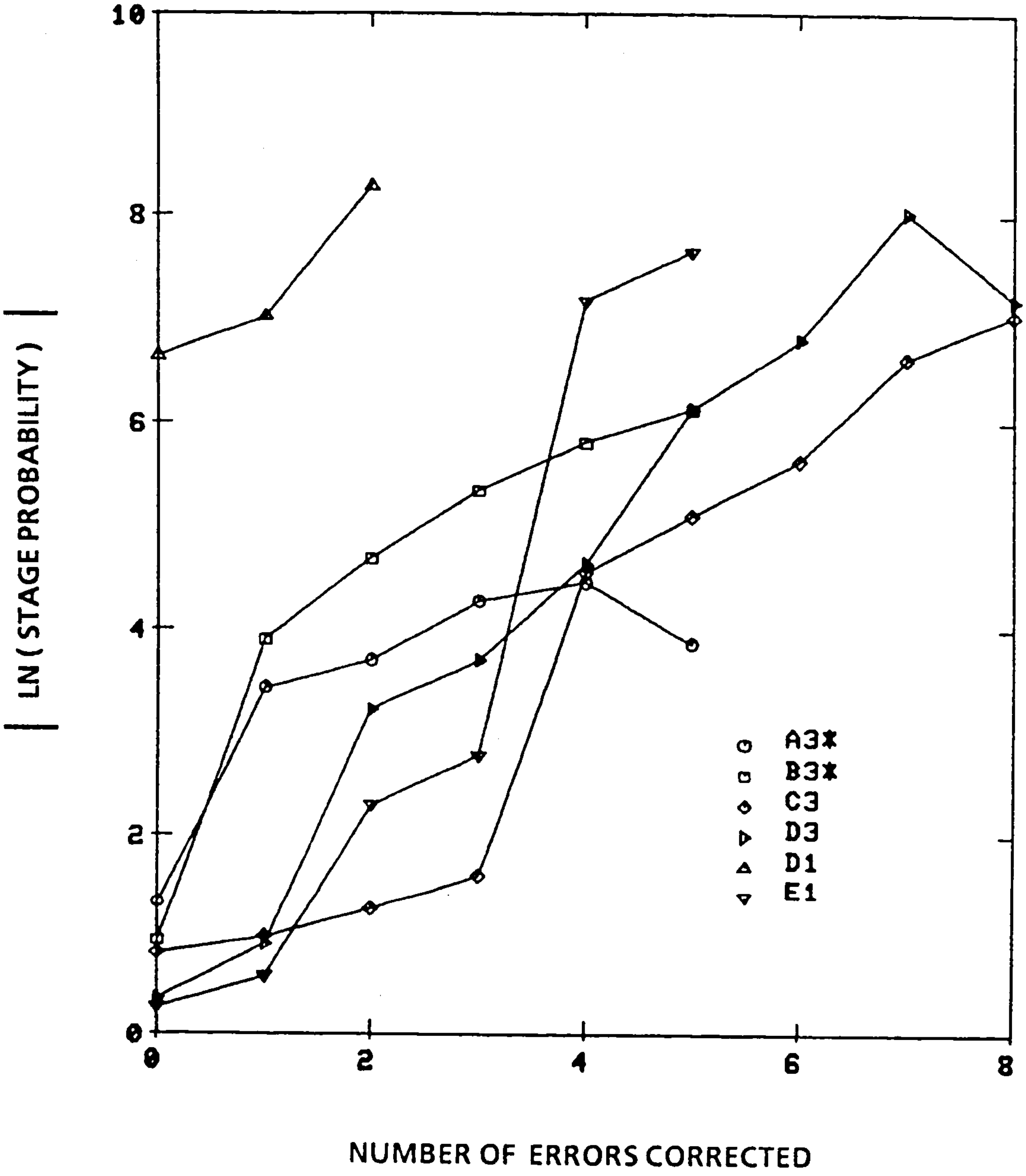


Figure 4.2-1. Estimated Error Rate as a Function of the Number of Errors Corrected -- Original Data Current Study.

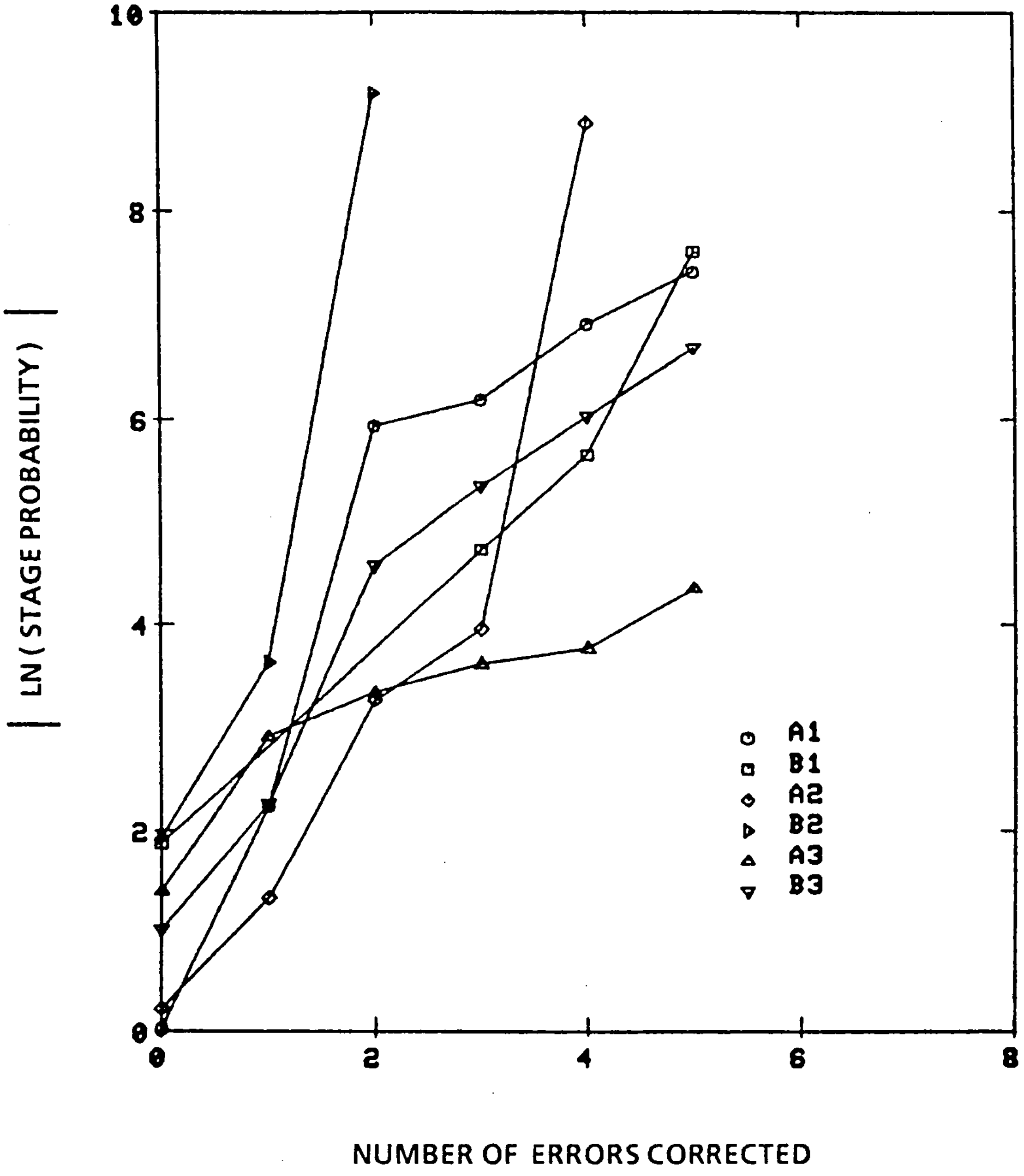


Figure 4.2-2. Estimated Error Rate as a Function of the Number of Errors Corrected -- Original Data Study No. 1.

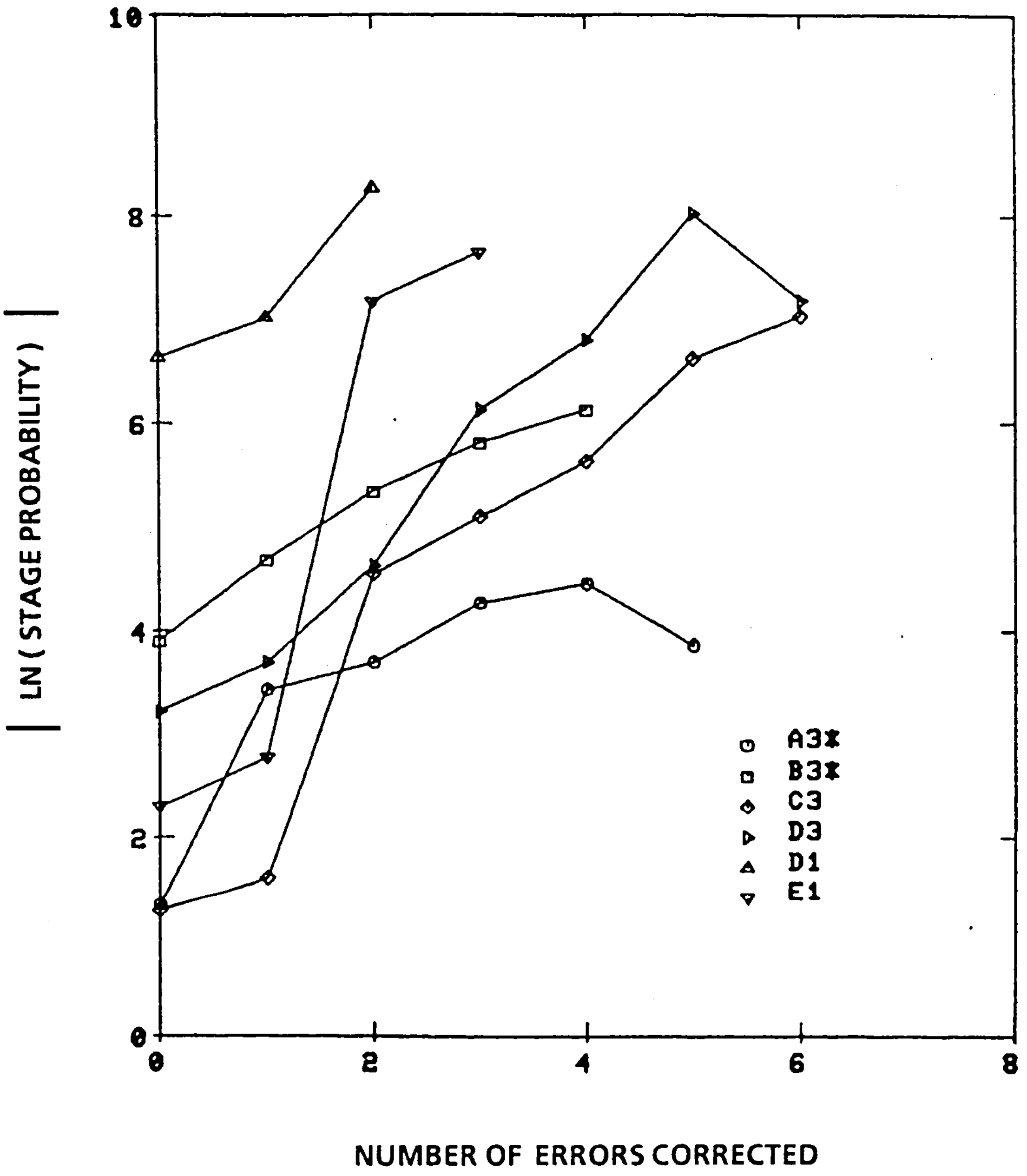


Figure 4.2-3. Estimated Error Rate as a Function of the Number of Errors Corrected -- Modified Origin.

TABLE 4.3-1

COMPARING THE EFFECT OF UNIFORM AND NON-UNIFORM USAGE DISTRIBUTION ON ERROR PROBABILITIES

Uniform (Study #1)				Non Uniform (Current Study)			
Program	Error	Error Prob. Per Execution	|ln P|	Program	Error	Error Prob. per Execution	|ln P|
A3	1	2.37x10^{-1}	1.44	A3*	1	2.50x10^{-1}	1.39
	2	1.78x10^{-2}	4.03		2	1.63x10^{-2}	4.12
	3	2.51x10^{-3}	5.99		3	2.31x10^{-3}	6.07
	4	2.37x10^{-3}	6.04		4	2.80x10^{-3}	5.88
	5	1.03x10^{-2}	4.58		5**	1.56x10^{-4}	8.77
	6	1.16x10^{-2}	4.46		6	3.23x10^{-3}	5.73
	7	4.08x10^{-3}	5.50		7	7.05x10^{-3}	4.95
	8	1.38x10^{-3}	6.59		8	1.25x10^{-3}	6.69
	9	1.97x10^{-4}	8.53		9	1.09x10^{-3}	6.82
	10	3.15x10^{-3}	5.76		10	2.49x10^{-3}	5.99
						** DID NOT OCCUR	
B3	1	3.29x10^{-1}	1.11	B3*	1	3.85x10^{-1}	0.95
	2	8.13x10^{-2}	2.51		2	1.26x10^{-2}	4.37
	3	3.08x10^{-4}	8.04		3	1.42x10^{-4}	8.86
	4	1.53x10^{-3}	6.48		4	1.79x10^{-3}	6.32
	5	4.55x10^{-3}	5.39		5	4.74x10^{-3}	5.35
	6	3.84x10^{-4}	7.86		6	2.21x10^{-4}	8.42
	7	1.75x10^{-3}	6.35		7	1.99x10^{-3}	6.22

TABLE 4.3-2

COMPARING THE EFFECT OF UNIFORM AND NON-UNIFORM USAGE DISTRIBUTION ON STAGE PROBABILITIES

	Uniform (Study #1)				Non-Uniform (Current Study)		
Program	No. of Corrected Errors	Stage Prob. Per Execution	$\lvert \ln P \rvert$	Program	No. of Corrected Errors	Stage Prob Per Execution	$\lvert \ln P \rvert$
A3	0	2.49x10^{-1}	1.39	A3*	0	2.67x10^{-1}	1.32
	1	5.38x10^{-2}	2.92		1	3.20x10^{-2}	3.44
	2	3.52x10^{-2}	3.35		2	2.45x10^{-2}	3.71
	3	2.69x10^{-2}	3.62		3	1.41x10^{-2}	4.27
	4	2.30x10^{-2}	3.77		4	1.16x10^{-2}	4.45
	5	1.29x10^{-2}	4.35		5	2.08x10^{-2}	3.87
B3	0	3.65x10^{-1}	1.01	B3*	0	3.94x10^{-1}	0.93
	1	1.04x10^{-1}	2.27		1	2.03x10^{-2}	3.90
	2	1.02x10^{-1}	4.58		2	9.31x10^{-3}	4.68
	3	4.68x10^{-3}	5.36		3	4.79x10^{-3}	5.34
	4	2.41x10^{-3}	6.03		4	2.98x10^{-3}	5.81
	5	1.24x10^{-3}	6.69		5	2.18x10^{-3}	6.13

experiments on the error probabilities and on the stage probabilities respectively and Figure 4.3-1 plots the stage probabilities for all four as a function of the number of errors corrected. Figures 4.3-2 and 4.3-3 pair the results of the two experiments by error number and by stage.

Figure 4.3-1 indicates that a drastic change in usage distribution seems to alter the detection process but in ways that maintain the basic patterns of detection. Figure 4.3-2 indicates that some of the error probabilities are severely altered by a change in usage, but that these differences average into the stage probabilities and have a less pronounced impact. There does not seem to be a great deal of support for the hypothesis referred to earlier but it is extremely premature to make any kind of judgment at this time.

4.4 EFFECT OF EXPERIENCE ON ERROR RATE

Experience was a deliberate design factor of this experiment, and the four codes written for problem #1 can be compared in order to partially understand the effect of this variable. Programmer D is a very mature programmer with over 20 years of experience. B and E are professional programmers, each with about five years of experience, but programmer E has had a richer, more varied exposure to computer science which manifested itself in a more sophisticated, more structured coding style for problem #1. Programmer A is the most junior of the group.

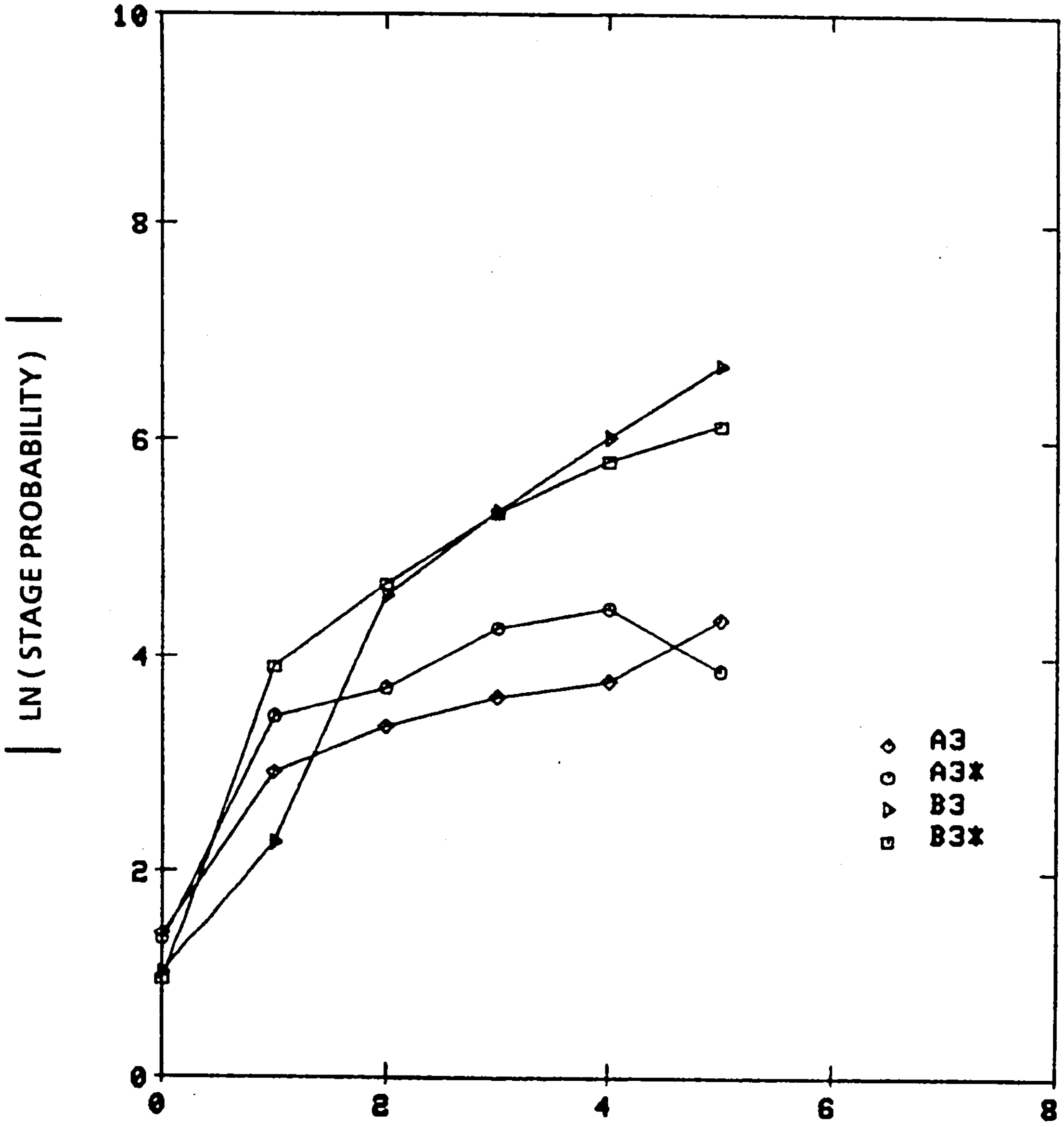
Figure 4.4-1 compares the four programs A1, B1, D1 and E1 using stage probability as a function of the number of errors corrected. As there were some errors in the published table for A1, and B1 has been recalculated according to the suggestion of reference [1], page 53, the recomputed values of both the individual error probabilities and the stage probabilities for these programs are given in Table 4.4-1.

The graphs are somewhat surprising in that except for the position of the crossing at zero, the slopes of the linear tendency in all cases are fairly similar and show no particular difference as a function of experience. Since D is by far the most experienced of the four programmers, it would seem that this experience greatly affects the initial zero crossing but affects the remaining structure only slightly.

4.5 EFFECT OF LANGUAGE ON ERROR DETECTION

Since reasonably efficient assembler-level programmers are usually found only in experienced professionals, these two factors are confounded in the next comparison. It is expected, however, that the effect due to language will far outweigh the effect of experience particularly in light of the results of the previous section. It will be assumed therefore that the observed differences are due to language alone and leave the subtleties of interaction to other researchers.

Figure 4.5-1 makes this comparison as before in terms of the stage probabilities. All of the four programmers are coding problem 3 with the junior level programmers A and B writing in FORTRAN and the senior programmers C and D writing in the assembler language for the ONYX Microcomputer. Figure 4.5-2 compares the same assembler language program of programmer D to another program by D written in FORTRAN for another problem.



NUMBER OF ERRORS CORRECTED

Figure 4.3-1. Estimated Stage Error Rate as a Function of the No. of Errors Corrected.

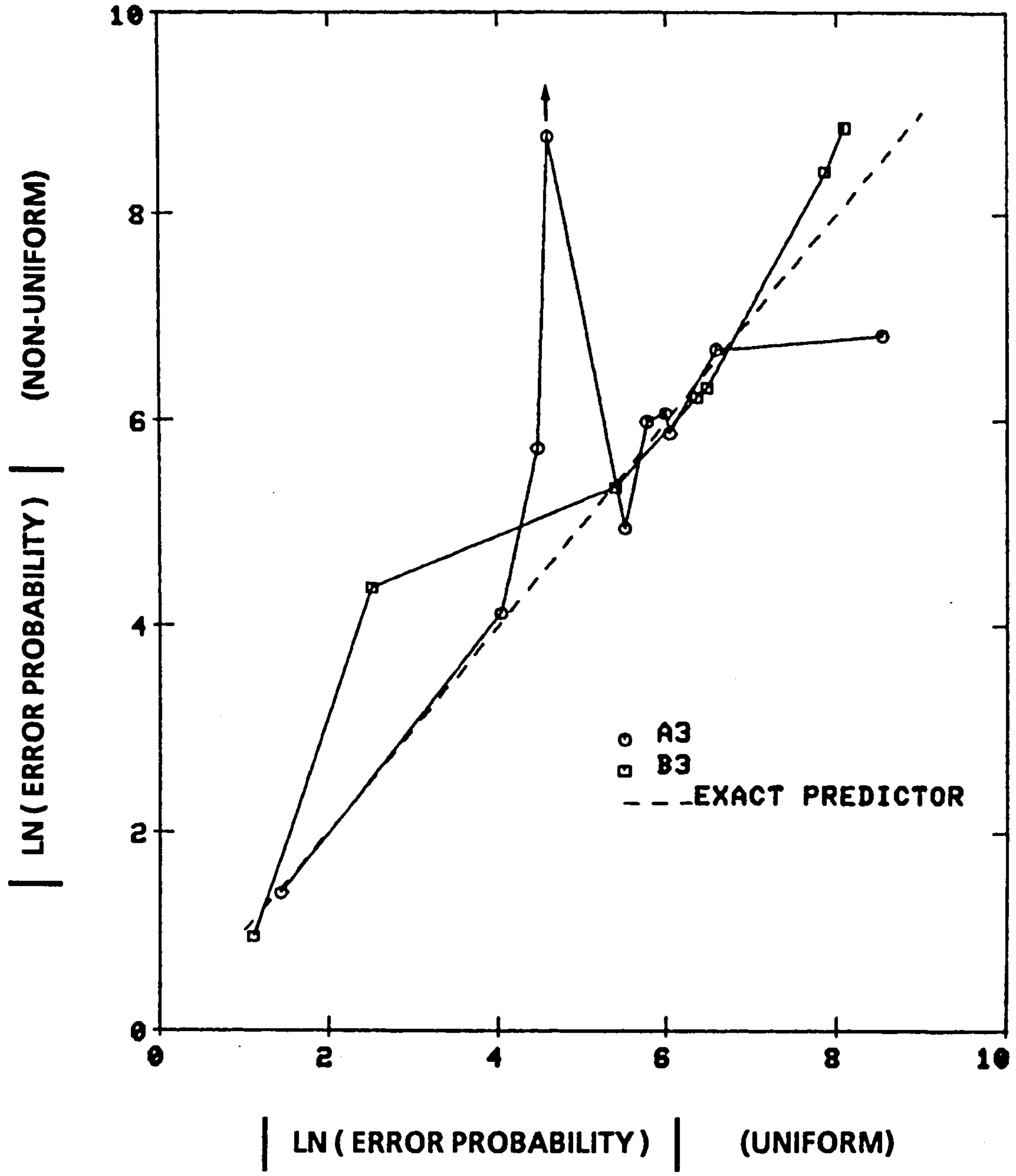


Figure 4.3-2. Comparing the Effect of Uniform and Non-uniform Usage Distributions on Error Probabilities.

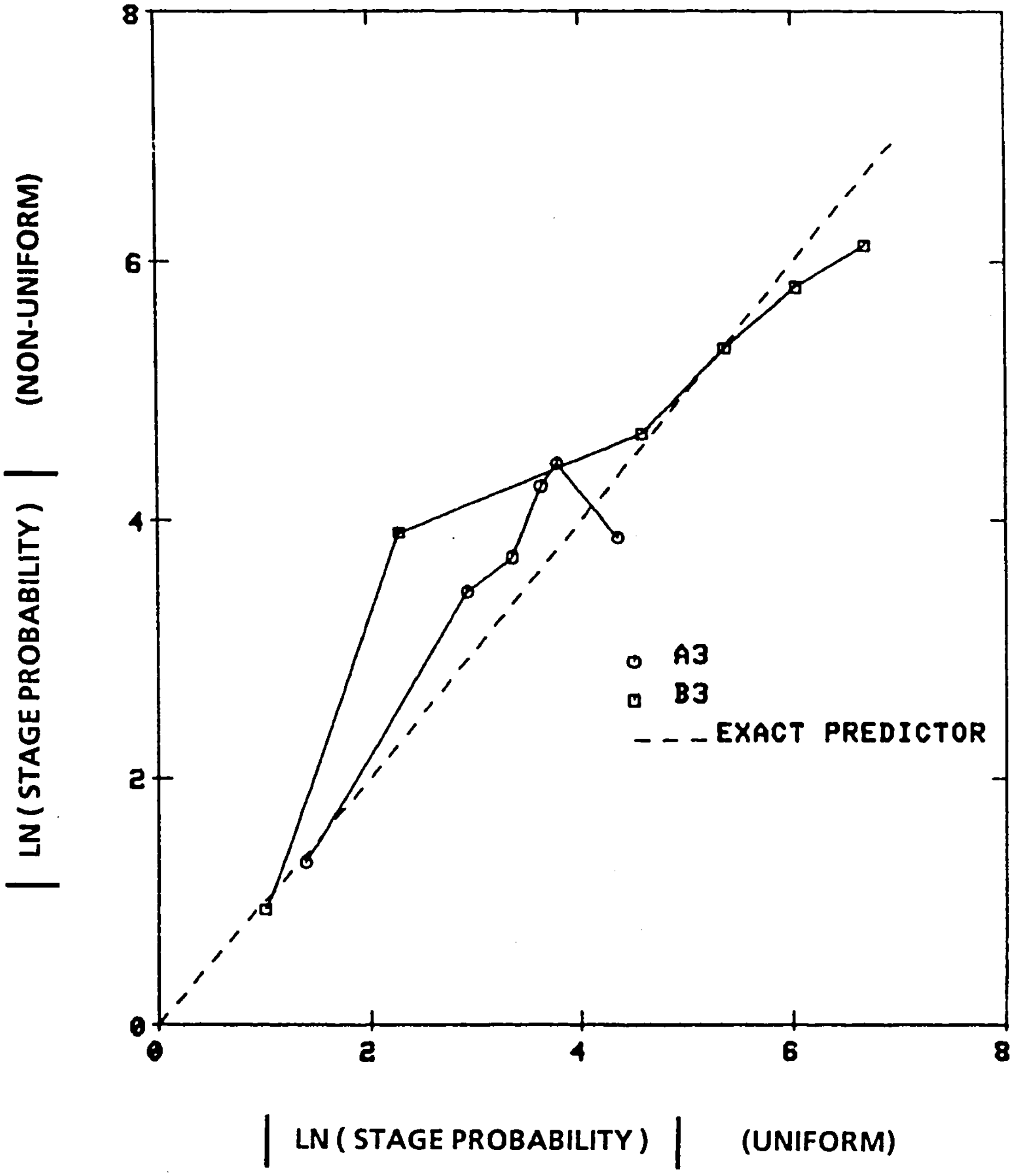


Figure 4.3-3. Comparing the Effect of Uniform and Non-uniform Usage Distributions on Stage Probabilities.

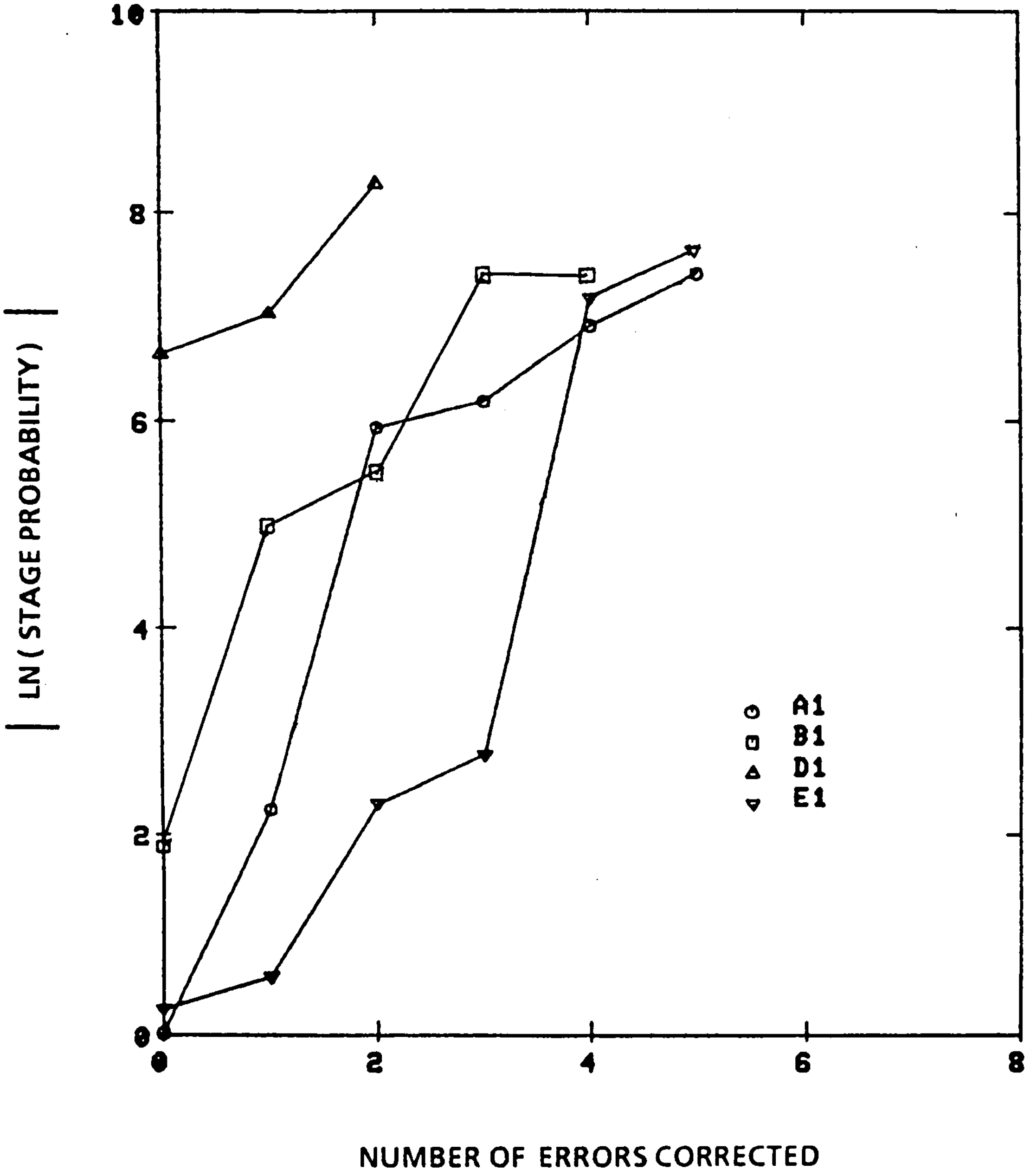


Figure 4.4-1. Effect of Experience on Log Failure Rate as a Functon of the No. of Errors Corrected.

Table 4.4-1

RECOMPUTED ERROR PROBABILITIES FOR STUDY NO. 1 - RANKED ESTIMATES

Prog.	Error No.	No. of Failures	Error Prob. Per Execution	Prog.	Error No.	No. of Failures	Error Prob. per Execution
A1	1	50	$8.20x10^{-1}$	B1	1	50	$1.52x10^{-1}$
	2	50	$9.84x10^{-2}$		4	50	$6.89x10^{-3}$
	4	30	$1.22x10^{-3}$		5	46	$2.38x10^{-3}$
	3	26	$8.04x10^{-4}$		6	25	$2.23x10^{-4}$
	5	23	$4.16x10^{-4}$		8	11	$1.07x10^{-4}$
	6	22	$3.81x10^{-4}$		7	12	$9.84x10^{-5}$
	8	14	$2.42x10^{-4}$		9	3	$2.68x10^{-5}$
	10	5	$8.66x10^{-5}$				
	7	1	$1.73x10^{-5}$				
	9	1	$1.73x10^{-5}$				

RECOMPUTED STAGE PROBABILITIES STUDY #1

Prog.	No. of Corrected Errors	Stage Prob. Per Execution	|ln P|	Prog.	No. of Corrected Errors	Stage Prob Per Execution	|ln P|
A1	0	$9.80x10^{-1}$	0.02	B1	0	$1.52x10^{-1}$	1.88
	1	$1.07x10^{-1}$	2.24		1	$8.80x10^{-3}$	4.73
	2	$2,65x10^{-3}$	5.93		2	$3.48x10^{-3}$	5.66
	3	$2.04x10^{-3}$	6.19		3	$4.99x10^{-4}$	7.60
	4	$9.96x10^{-4}$	6.91		4	$5.21x10^{-4}$	7.56
	5	$6.05x10^{-4}$	7.41				

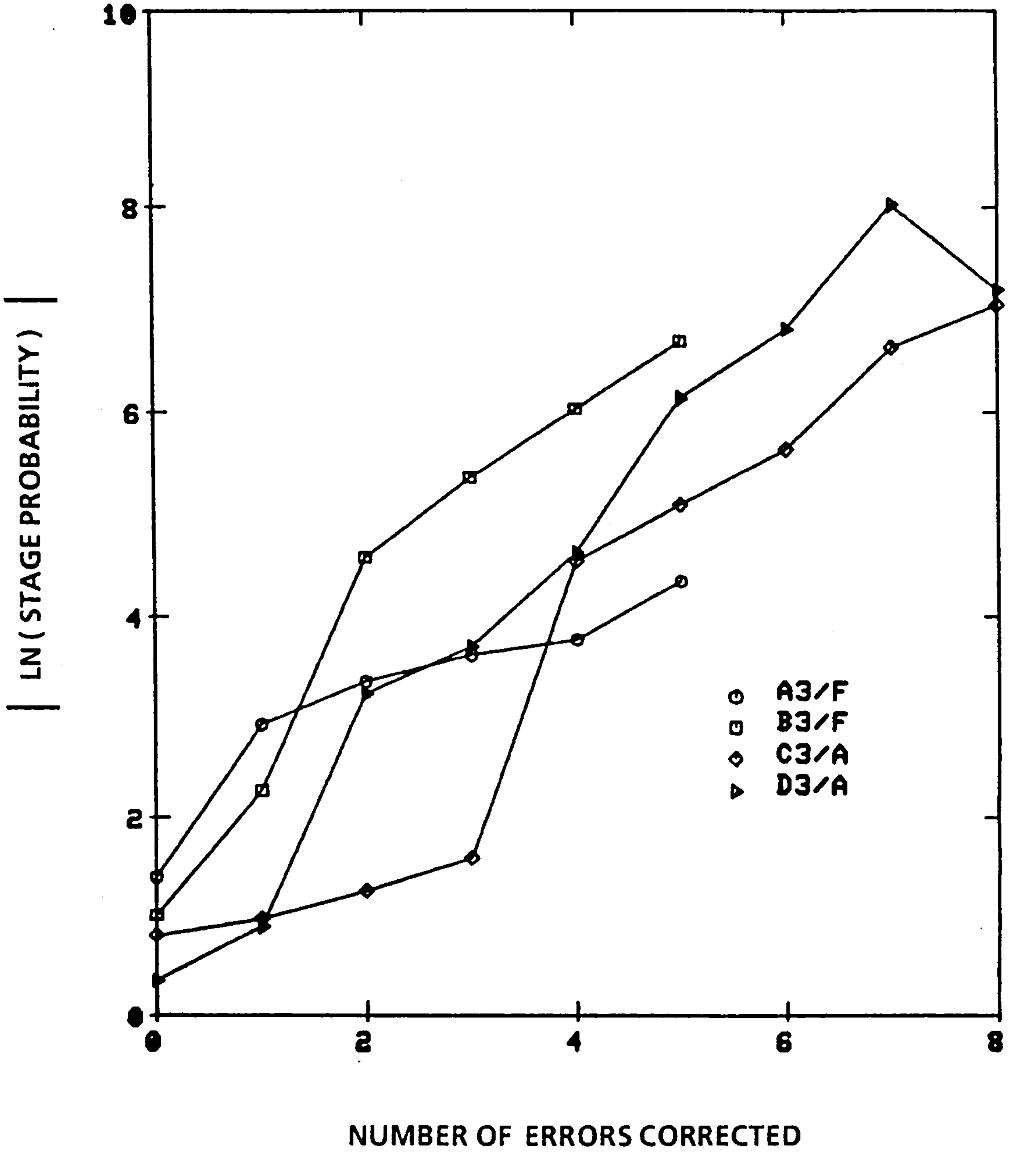


Figure 4.5-1. Effect of Language / Experience on Log Failure Rate as a Function of the No. of Errors Corrected.

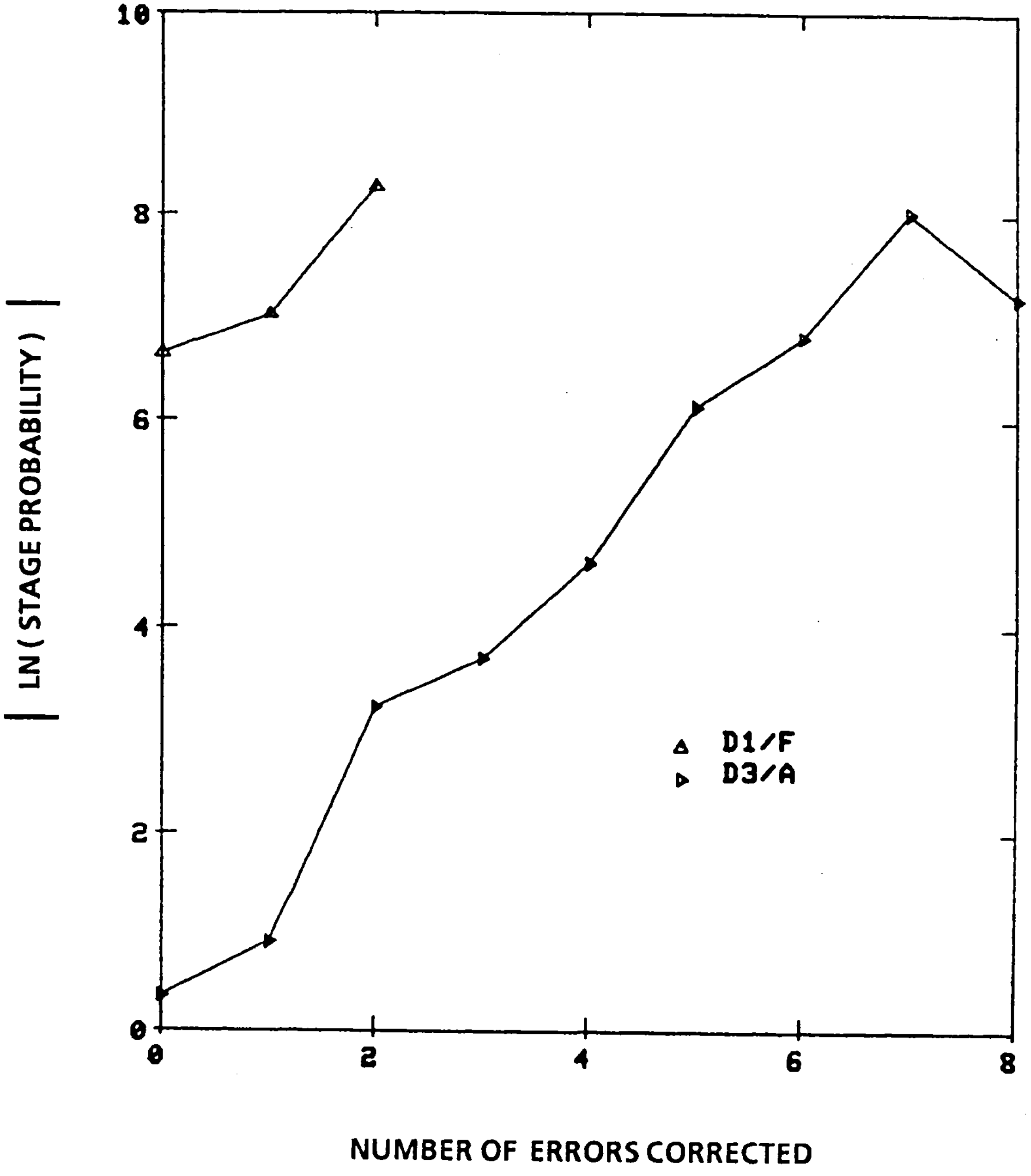


Figure 4.5-2. Effect of Language on the Performance of a Single Programmer.

The graphs in Figure 4.5-2 suggest that assembler language, as expected, has a strong effect on the failure structure of software. Those in Figure 4.5-1 suggest that it is equivalent to several years of experience. That is, the performance of an experienced programmer writing in a low-level language is little different in terms of error structure from a novice programmer writing in FORTRAN. When comparing the performance of a single programmer using high- and low-level languages, the differences are even greater but primarily one of zero crossing rather than slope. The surprising result in all of these charts is that slope varies so little across widely varying conditions.

4.6 EFFECT OF PROBLEM TYPE ON ERROR STRUCTURE

Program 4 was selected to emphasize a still different process in program development. Programs 1, 2 and 3 were selected, somewhat naively, as problems offering enough challenge to the programmer to cause errors, without being so difficult that programming would take weeks of effort. In thinking about the task of scientific programming, however, it seems that there are at least two broad categories depending on the nature of the problem, one primarily involving analysis and the other primarily involving program design, and that all programs represent some mix of these two. Problems 1, 2 and 3 place most of their emphasis on analysis and little on design so a 4th problem was selected with design in mind. Unfortunately, the problem selected offered little challenge to experienced programmers and only a single error was detected for each programmer in the more than 47,000 executions conducted to date. See Table 4.2-1 for a summary of this data.

4.7 EXPONENTIAL ASSUMPTION AND TIME BETWEEN SOFTWARE FAILURES

In Section 5.0 of this document, analytical results are presented that show that the unconditional times between failures under a very general model of software failure are not exponentially distributed nor are they mutually independent. This has particular impact on the analysis of the data of this experiment since the consequence of this result means that the distribution of time to next failure, that is the distribution of stage failure, is not exponential and is in fact dependent on the history observed to that stage.

The unsuitability of the exponential assumption was known at the time of the first study and a result was presented in reference [1] that showed for a special case that the distribution of the time to next failure was an exponential mixture with decreasing failure rate. Since an exponential mixture distribution is still monotone decreasing everywhere, an exponential approximation to the mixture might be reasonable. This has been explored for several stages and programs in terms of the cumulative distribution of the observed data plotted as if it were exponentially distributed. That is, if the data set is exponential the log of the survivor function plotted against the ordered observations should approximate a straight line. The survivor function in the case of noncensored data is defined as

$$R(t_{(i)}) = 1 - i/n$$

Log survivor functions for several program-stage combinations have been plotted in Figures 4.7-1 through 4.7-4.

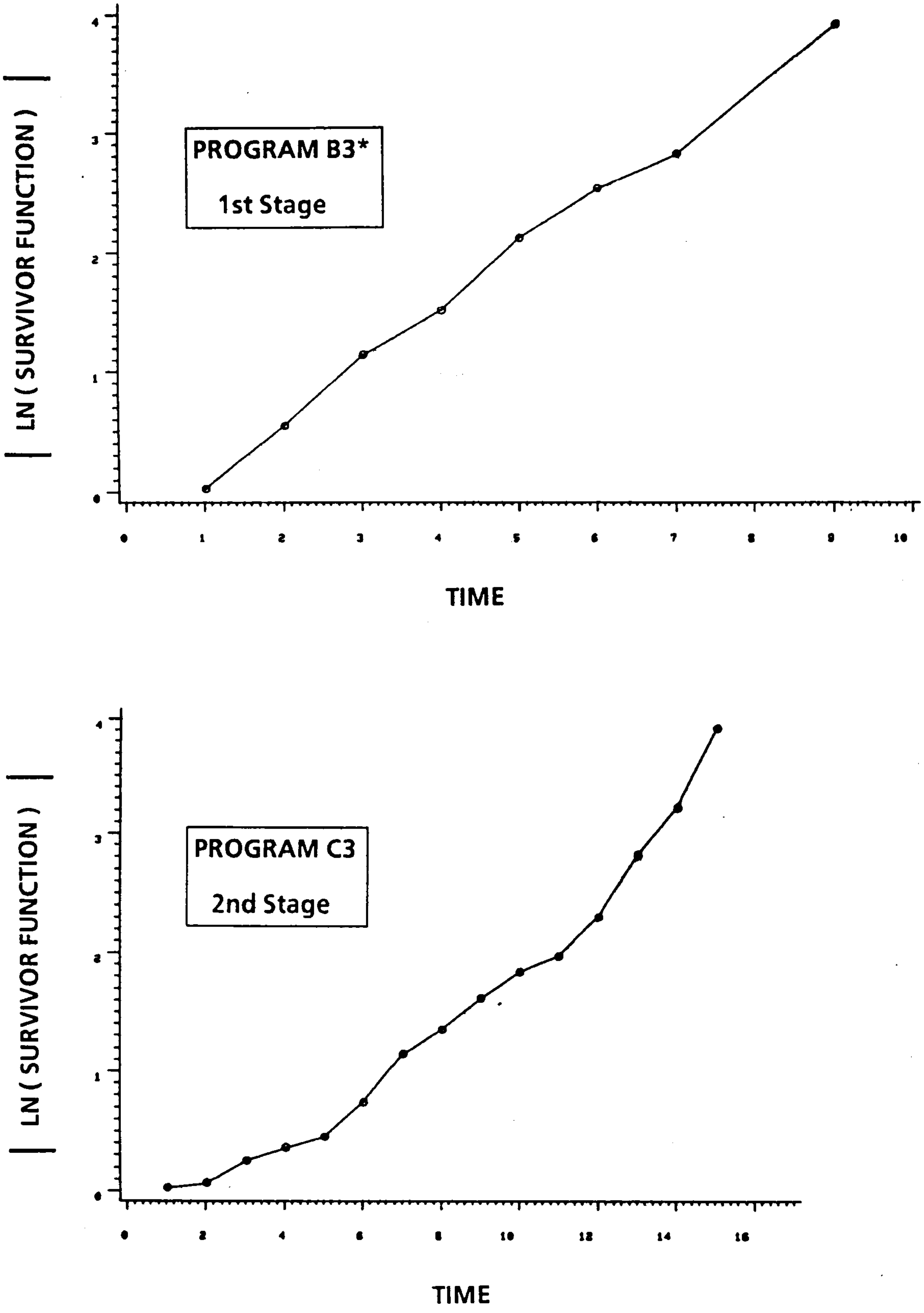


Figure 4.7-1. Minus Log of Observed Survivor Function (Selected Points) for Program B3*, 1st Stage and Program C3, 2nd Stage.

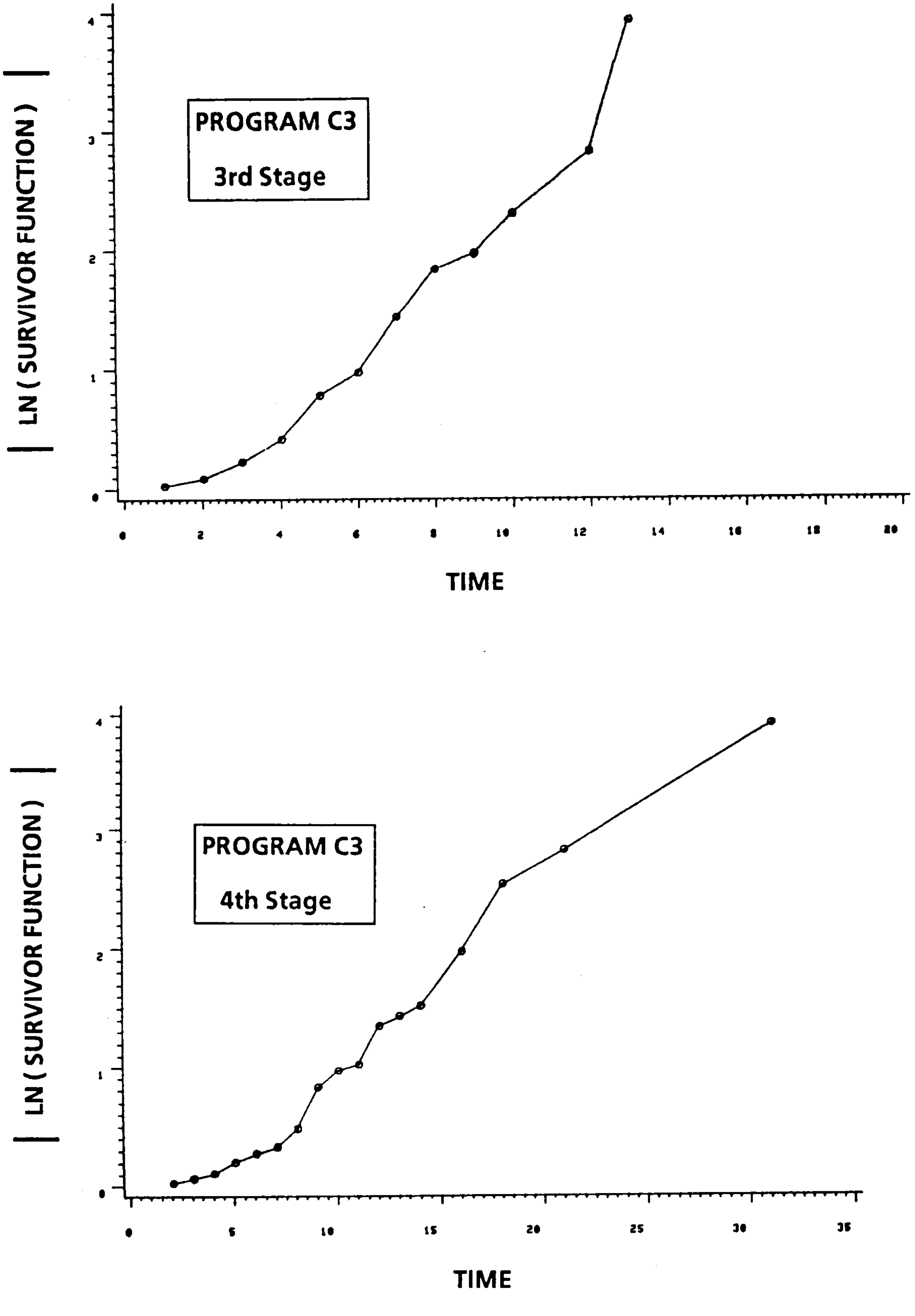


Figure 4.7-2. Minus Log of Observed Survivor Function (Selected Points) for Program C3, 3rd and 4th Stages.

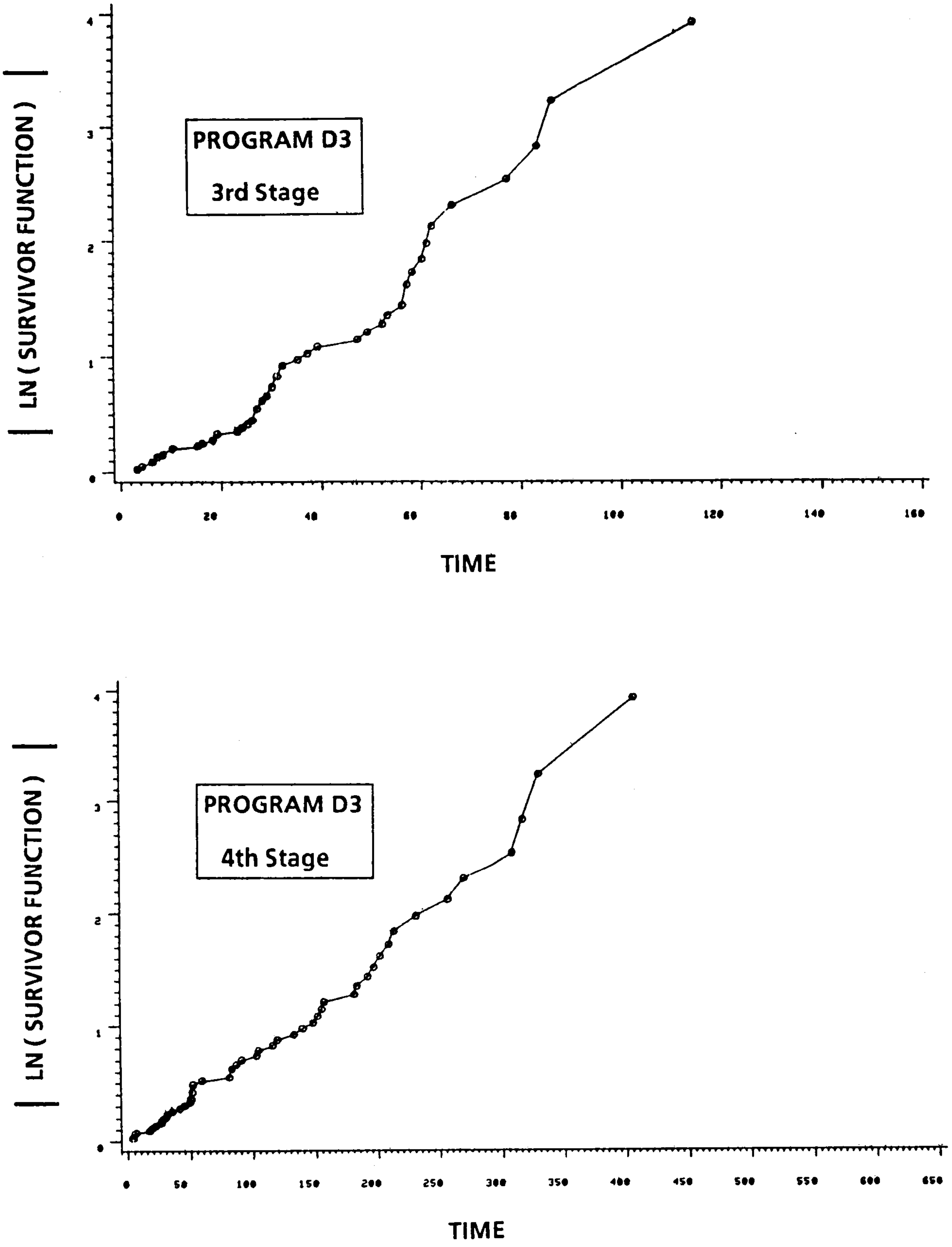


Figure 4.7-3. Minus Log of Observed Survivor Function (Selected Points) for Program D3, 3rd and 4th Stages.

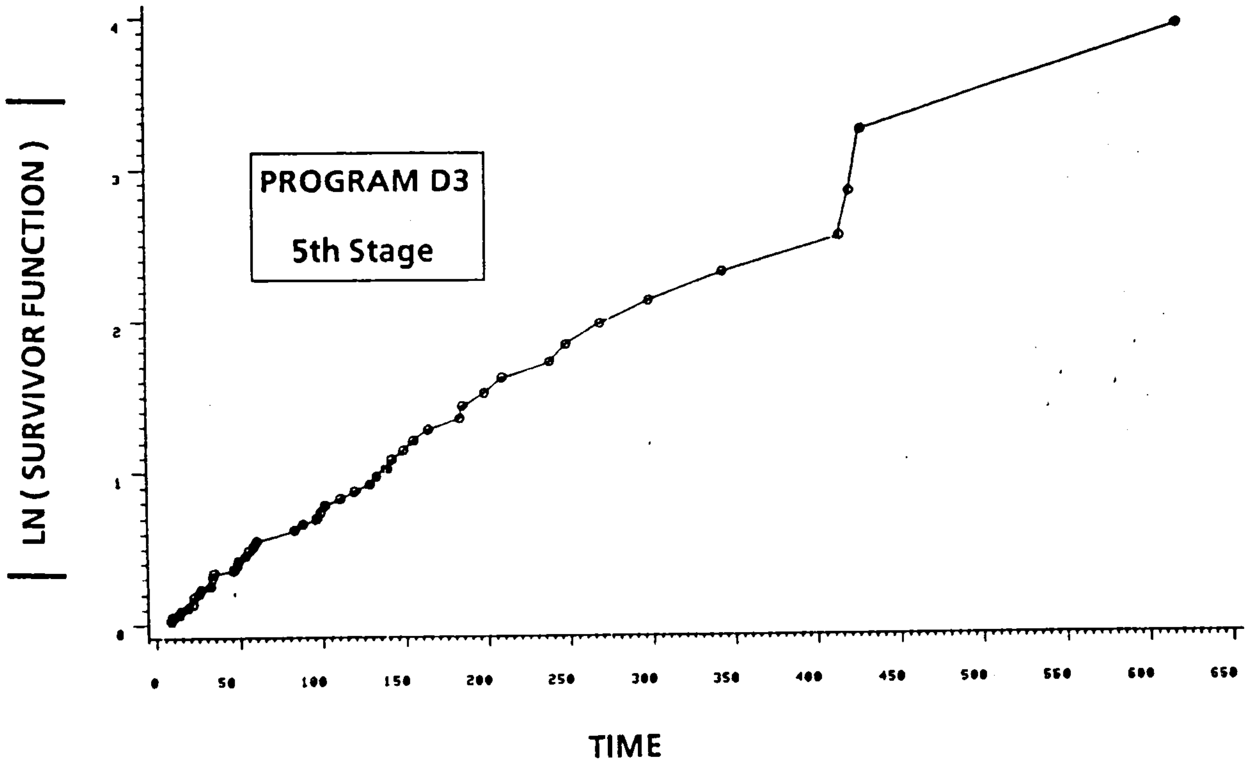


Figure 4.7-4. Minus Log of Observed Survivor Function (Selected Points) for Program D3, 5th Stage.

The thing to note in these graphs and similar ones given in reference [1] is that although a straight line appears to be a reasonable approximation to the log survivor function, in every case, a "best fitting" straight line using some reasonable optimization criteria would underrun the data in the upper tail. That is, in every case, for some large t_o and for some optimum λ

$$- \ln (1-F(t)) > \lambda t \quad \text{for } t \geq t_o$$

where F(t) is the cumulative distribution function of the program's life in a given stage. This implies that

$$e^{-\lambda t} > 1-F(t) \text{ for } t \geq t_o$$

which suggests that F has a smaller right-hand tail than the "best" exponential. Thus the exponential defined in this way does not provide a conservative approximation to the mixture distribution since it weights too heavily the longer times between errors, for a given stage.

4.8 DEPENDENCE OF STAGE LIFE LENGTH ON TOTAL LIFE AND ADJACENT LIFE

In order to measure the degree of the dependence between stages to determine if this is an important factor, it was felt that the nature of the effect would be strengthened if it were considered on a cumulative life length basis rather than between stages. That is, the total life of the program up to the beginning of i'th stage should govern the life of the i'th stage more than the life of any of the preceding stages. The expectation was that the correlation between the total life up to the i'th stage would be negatively correlated with the life of the i'th stage but only mildly. Long lives before i would tend to support short lives after i and vice versa but because of the mixing of the order of withdrawal, the correlation would not be strong. Table 4.8-1 gives the observed correlation between the total time to the i'th failure and the time to the next failure i+1 where time is measured in numbers of executions. For this computation multiple errors are treated as single errors as the emphasis here is on the observed spacings and their relative size, not the number of errors.

There is some tendency for these values to cycle: starting low, increasing, decreasing and increasing again with some variation in cycle length, but the expected negative correlation is nowhere substantiated. Not only are the correlations generally positive but are, in some cases, large as well as positive. The reasons for this are not at all clear but may have something to do with the probability distribution of the original errors and the relative likelihoods of the order in which they are removed.

Cox's proportional hazards model, introduced in reference [1] assumes that the time between shocks is independent. Theoretical results presented in Section 5.0 demonstrate that under very general conditions the spacings for software are dependent. It was therefore hoped that the nature of the dependence would be negligible which does not appear to be the case.

Correlations of adjacent spacings are given in Table 4.8-2. Generally the values are small and positive but some are significantly greater than .5. It is not known if this effect lessens as the length of the spacing increases as more errors are corrected. The tendency to pulse is also in this data although not as pronounced as in Table 4.8-1. Correlations between the j'th and (j+2)nd spacing have also been computed. They are generally smaller and again predominately positive. These facts generally imply that the size of the dependence between the spacings is not a negligible factor.

Table 4.8-1

CORRELATION BETWEEN THE TOTAL TIME TO THE ith FAILURE AND THE TIME TO THE NEXT FAILURE i + 1

Current Study

Program	i	S.S.	Corr	Program	i	S.S.	Corr
A3*	1	49	.12	B3*	1	50	.26
	2	41	.21		2	45	.53
	3	26	- .26		3	37	-.027
	4	9	.65		4	17	-.065
C3	1	50	.009	D3	1	50	.21
	2	50	.50		2	50	.48
	3	48	.30		3	50	.45
	4	47	.40		4	50	.035
	5	35	.01		5	50	.015
	6	18	.49		6	49	.59
	7	9	.61		7	36	.012
					8	8	-.27
D1	1	50	.015	E1	1	50	.024
	2	49	- .042		2	50	.19
					3	50	.030
					4	50	-.15
					5	35	.047

Table 4.8-1 (Continued)

CORRELATION BETWEEN THE TOTAL TIME TO THE ith FAILURE AND THE TIME TO THE NEXT FAILURE i + 1

Study #1

Program	i	S.S.	Corr	Program	i	S.S.	Corr
A1	1	50	-.035	B1	1	50	-.009
	2	49	-.054		2	50	.05
	3	34	-.24		3	46	.43
	4	17	-.16				
A3	1		-.012	B3	1	50	-.062
	2		.25		2	50	.22
	3		.28		3	36	-.14
	4		.33		4	17	-.14

Table 4.8-2

AUTOCORRELATION BETWEEN ADJACENT STAGES
i AND i+1

Current Study

Program	i	S.S.	Corr	Program	i	S.S.	Corr
A3*	1	49	.12	B3*	1	50	.26
	2	41	.19		2	45	.53
	3	26	-.22		3	37	-.022
	4	9	.67		4	17	.15
C3	1	50	.0087	D3	1	50	.21
	2	50	.50		2	50	.47
	3	48	.24		3	50	.46
	4	47	.28		4	50	.0022
	5	35	-.040		5	50	-.040
	6	18	.35		6	49	.49
	7	9	.73		7	36	.059
					8	8	-.21
D1	1	50	.015	E1	1	50	.024
	2	49	-.073		2	50	.214
					3	50	-.055
					4	50	-.15
					5	35	.046

Table 4.8-2 (Continued)
AUTOCORRELATION BETWEEN ADJACENT STAGES
(i AND (i+1))

Study #1

Program	i	S.S.	Corr	Program	i	S.S.	Corr
A1	1	50	-.035	B1	1	50	-.0086
	2	49	-.054		2	50	.042
	3	34	-.239		3	44	.23
	4	17	.109				
A3	1	49	.073	B3	1	50	-.062
	2	45	.21		2	50	-.011
	3	36	.164		3	34	-.18
	4	25	.159		4	14	-.087
					5	4	.44

5.0 ANALYTIC CONSIDERATIONS

At the time the first study was conceived, the fact that replication would have its own statistical framework that would differ from those which had become popular for the traditional problem, was not completely considered. As the requirement for a deeper understanding of the process grew, so did the need for a consistent theoretical foundation. This section introduces the modeling advances achieved to date. They are by no means complete and many important problems remain to be solved.

The software reliability problem is equivalent to the problem of a not quite perfect hardware design released before all the bugs are removed to let experience perfect the design. It is assumed that there are some number of bugs in the design and that each bug has a certain probability or average rate with which it is detected. Its time to detection, measured in some convenient unit, is a random variable. The bugs, then, compete for detection and are removed as discovered. The ordered detection times form the basic observations. The following paragraphs formalize this process.

The section begins by discussing the general problem from two points of view: discrete and continuous, and a theorem is proved that gives a sufficient condition under which they can be treated as equivalent processes. These results are then particularized to the problem of forecasting the future from a given state. The results in this section have a direct bearing on the replicated experiment.

5.1 MULTINOMIAL MODEL

In the following, a probabilistic model or framework is presented within which the process of software debugging may be analyzed and understood. The model is by no means a perfect reflection of reality but it is a step towards capturing some of the very basic aspects of a software debugging experience. In particular, it can provide a general model within which the appropriateness of certain modeling assumptions made by other researchers in a more or less ad hoc fashion can be evaluated.

It is assumed that the generic piece of software manipulates a certain set of inputs, taken from a space A of possible inputs, and either successfully computes the desired output or not. In the latter case, the used inputs have resulted in the detection of a bug. Thus, the input space A is viewed as partitioned into mutually disjoint sets

$$A = A_0 \cup A_1 \cup \ldots \cup A_N \quad (A_i \cap A_j = 0 \quad i \neq j),$$

where input from A_0 results in the successful execution of the software program and input from A_i (i=1, ..., N) results in the detection of bug i in the program.

At this point, the possibility that multiple bugs may be found for certain inputs is excluded. This model deficiency may be dealt with at a later stage after the ramifications of the current model have been fully understood.

Each execution of the program constitutes a multinomial trial, the outcome of which can be characterized by the "cell" A_i (i=0, 1, ..., N) in which the input was chosen.

A probability structure is imposed by considering a usage probability distribution over the input space A. Let $p_o, p_1, ..., p_N$ be the respective probabilities that the input will be chosen in $A_o, A_1, ..., A_N$, with $p_o + p_1 + ... + p_N = 1$. The probabilities p_i confound the "size" and location of the sets A_i with the usage frequency distribution over A.

The program debugging sequence consists of repeatedly exercising the program with various inputs and observing whether the program executes properly or whether a bug has been uncovered. In the latter case, the program is repaired with respect to this particular bug and the debugging sequence is continued. The repair of the program with respect to bug i amounts to joining the partition set A_i to the set A_o.

At this point, a further strong model assumption is made, namely that the individual multinomial trials are independent. In some contexts this assumption may be quite realistic whereas in other contexts successive usages of the program may use inputs which are highly correlated. Relaxation of this independence assumption may be pursued after the independence model is well understood.

Some useful notation for describing the debugging experience is now introduced.

$$\text{Let } X_{it} = 1 \quad \text{if the } t^{th} \text{ trial results in input choice from cell } A_i, \; i = 0, ..., N.$$

$$= 0 \quad \text{otherwise}$$

$$X_{ot} + X_{1t} + ... + X_{Nt} = 1$$

$$P(X_{it} = 1) = 1 - P(X_{it} = 0) = p_i \qquad t = 1, 2, 3, ... \,.$$

$$\text{Let } Y_i = \min \{t: X_{it} = 1\} \qquad i = 1, 2, ..., N,$$

i.e., Y_i represents the waiting time (counted in number of trials) to the first detection of bug i.

In the process of debugging, $Y_1, Y_2, ...$ are not actually observed since the "model label" of the bug is not known when it is found. All that is known is that a new bug has been detected (since all previously detected bugs have been corrected) and its occurrence time.

Thus, what is really observed after having found the k^{th} bug are the first k order statistics $Y_{(1)} \; ... \; Y_{(k)}$ of $Y_1, ..., Y_N$. Since the joint distribution of $(Y_{(1)}, ..., Y_{(k)})$ derives from that of $(Y_{(1)}, ..., Y_{(N)})$ the latter will be studied first.

Some more notation will be useful. Let $\pi = (\pi_1, ..., \pi_N)$ denote a permutation of (1, ..., N) and let $\underline{P}_N$ denote the set of all N! such permutations. Further let

$$\Lambda_i(\pi) = \sum_{j=i}^{N} p_{\pi_j}, \; i = 1, ..., N$$

and note that $\Lambda_1(\pi) = 1 - p_o$ for all $\pi \in \underline{P}_N$. Let Q be the random vector in $\underline{P}_N$ consisting of the bug labels in the order in which the bugs are uncovered, i.e., if $Q = \pi$ then π_1 is the model label of the first bug uncovered, etc. Further, let $D_i = Y(i) - Y(i-1)$, $i = 1, \ldots, N$, $(Y(o) = 0)$ denote the spacings between the bugs.

Then for integers $d_i \geq 1$ and $\pi \in \underline{P}_N$,

$$P(Q = \pi, D_i = d_i, i=1, \ldots, N) = P(Y_{\pi_i} = \sum_{j=1}^{i} d_j \ , i = 1, \ldots, N)$$

$$= \prod_{i=1}^{N} p_i \prod_{i=1}^{N} (1 - \Lambda_i(\pi))^{d_i - 1} \qquad (1)$$

$$= w(\pi) \prod_{i=1}^{N} \Lambda_i(\pi)(1 - \Lambda_i(\pi))^{d_i - 1}$$

where $w(\pi) = \prod_{i=1}^{N} (p_i / \Lambda_i(\pi))$.

Summing (1) over $d_i \geq 1$, $i = 1, \ldots, N$ gives $P(Q = \pi) = w(\pi)$ and thus

$$P(D_i = d_i, i = 1, \ldots, N / Q = \pi) = \prod_{i=1}^{N} \Lambda_i(\pi)(1 - \Lambda_i(\pi))^{d_i - 1}. \qquad (2)$$

The marginal distribution of $D_1, \ldots, D_N$ (and hence of $Y(1) < \ldots < Y(N)$) is given by

$$P(D_i = d_i, i = 1, \ldots, N) = \sum_{\pi \in \underline{P}_N} w(\pi) \prod_{i=1}^{N} \Lambda_i(\pi)(1 - \Lambda_i(\pi))^{d_i - 1} \qquad (3)$$

$$= \sum_{\pi \in \underline{P}_N} P(Q = \pi) \, P(D_i = d_i, i = 1, \ldots, N / Q = \pi).$$

As seen from (2) the spacings $D_1, \ldots, D_N$ are conditionally (given $Q = \pi$) independent geometric random variables with decreasing "success probabilities" $\Lambda_1(\pi) > \Lambda_2(\pi) > \ldots > \Lambda_N(\pi)$. Unconditionally, as seen by (3), the spacings $D_2, \ldots, D_N$ are neither independent nor geometrically distributed. However, D_1 is independent of $(D_2, \ldots, D_N)$, since $\Lambda_1(\pi) = 1 - p_o$ is independent of π. Also, D_1 has unconditionally a geometric distribution with "success probability" $\Lambda_1(\pi) = 1 - p_o$.

Ultimately, only the first k ($k \leq N$) bugs in the debugging sequence are observed which is equivalent to knowing the first k spacings $D_1, \ldots, D_k$. Their joint density is obtainable from (3) as

$$P(D_i = d_i, i = 1, \ldots, k) = \sum_{\pi \in \underline{P}_N} w(\pi) \prod_{i=1}^{k} \Lambda_i(\pi)(1 - \Lambda_i(\pi))^{d_i - 1} \tag{4}$$

$$= \sum_{\pi \in \underline{P}_N} P(Q = \pi)\, P(D_i = d_i, i = 1, \ldots, k/Q = \pi),$$

where

$$P(D_i = d_i, i = 1, \ldots, k/Q = \pi) = \prod_{i=1}^{k} \Lambda_i(\pi)(1 - \Lambda_i(\pi))^{d_i - 1}. \tag{5}$$

5.2 EXPONENTIAL MODEL

The discrete nature of program executions and the nature of the error withdrawal process (except for intersecting errors) makes the multinomial model a natural one for formulating the software reliability problem. Many models in use, however, model the process, for convenience, with independent exponential random variables with decreasing failure rate. In this section, results similar to those of the previous section are obtained.

5.2.1 General Model

Let $Z_1, \ldots, Z_N$ be independent exponential random variables with respective failure rates $\lambda_i = p_i$, $i = 1, \ldots, N$, i.e., the joint density of $Z = (Z_1, \ldots, Z_N)$ is

$$f_N(Z_1, \ldots, Z_N) = \prod_{i=1}^{N} p_i e^{-p_i Z_i}, \quad Z_i > 0, i = 1, \ldots, N$$

Here Z_i is interpreted as the waiting time to the first occurrence of bug i. Note that here the waiting times for the individual bugs are assumed to be independent whereas in the discrete model the waiting times are by assumption dependent since they are based on a sequence of multinomial trials.

Let $Z_{(1)}, \ldots, Z_{(N)}$ denote the order statistics of $Z_1, \ldots, Z_N$ and by $\tilde{D}_i = Z_{(i)} - Z_{(i-1)}$, $(Z_{(o)} = 0)$ denote the spacings between these order statistics. Further, let $\tilde{Q}$ denote the random N-vector of bug labels in the order in which they are uncovered, i.e.,

$$\tilde{Q} = \pi \in P_N \text{ iff } Z_{(i)} = Z_{\pi_i} \; i = 1, \ldots, N.$$

The joint density of $\tilde{D} = (\tilde{D}_1, \ldots, \tilde{D}_N)$ and $\tilde{Q}$ is given by

$$g_N(d, \pi) = \prod_{i=1}^{N} p_{\pi_i} \exp(- p_{\pi_i} \sum_{j=1}^{i} d_j)$$

$$= w(\pi) \prod_{i=1}^{N} \Lambda_i(\pi) \exp(- \Lambda_i(\pi) d_i) \quad , d_i > 0 \; i=1, \ldots, N$$

where $\Lambda_i(\pi) = \sum_{j=i}^{N} p_{\pi_j}$ and $w(\pi) = \prod_{i=1}^{N} (p_i / \Lambda_i(\pi))$ as before.

Again, it can be seen that $P(\tilde{Q}=\pi) = w(\pi)$ and that the joint density of $\tilde{D}$ is:

$$h_N(d) = \sum_{\pi \in \underline{P}_N} w(\pi) \prod_{i=1}^{N} \Lambda_i(\pi) \exp(- \Lambda_i(\pi) d_i) \tag{6}$$

whereas the conditional density of $\tilde{D}$ given $\tilde{Q} = \pi$ is:

$$g_N(d/\pi) = \prod_{i=1}^{N} \Lambda_i(\pi) \exp(- \Lambda_i(\pi) d_i), \tag{7}$$

i.e., conditionally, given $\tilde{Q} = \pi$, the spacings are independent exponentially distributed with decreasing failure rates $\Lambda_1(\pi) > \ldots > \Lambda_N(\pi)$. However, unconditionally only $\tilde{D}_1$ and $(\tilde{D}_2, \ldots, \tilde{D}_N)$ are independent and only $\tilde{D}_1$ is exponentially distributed. $\tilde{D}_2, \ldots, \tilde{D}_N$ are dependent and not exponentially distributed. By integrating out the remaining d variables from (6), the unconditional distribution of d_j is obtained as

$$h'_{j,N}(d_j) = \sum_{\pi \in \underline{P}_N} w(\pi)\ \Lambda_j(\pi) \exp(-\Lambda_j(\pi) d_j) \tag{8}$$

i.e., an exponential mixture. This distribution is the general case of the marginal distribution of the second spacing for N = 3 given in reference (1).

5.2.2 Specific Results

The results of Section 5.2.1 have been particularized for the cases N=2 and N=3. This section gives the joint distribution for the spacings for these N's, and the marginal distributions of the spacings as a function of the number of errors corrected. Also included are the results of a Bayesian inquiry into the nature of the distribution of the parameter for these N's given the number of errors corrected. The results are applicable to a general prior but are confined to N=2, 3. The extension to a general N has been completed but is not included in this report.

5.2.2.1 N=2

Let λ_1 and λ_2 be the failure rate of the two errors and let t_1 and t_2 be the times of occurrence for each error respectively. Then the two possible orderings are $\pi = (1,2)$ and $\pi = (2,1)$ and from $w(\pi)$ in (1) one obtains

$$P(\pi = (1,2)) = P(t_1 \leq t_2) = \lambda_1/(\lambda_1 + \lambda_2)$$

and

$$P(\pi = (2,1)) = \lambda_2/(\lambda_1 + \lambda_2)\ .$$

Thus using (6) the joint density of d_1 and d_2 is

$$h_2(d_1,d_2) = \frac{\lambda_1}{\lambda_1 + \lambda_2}(\lambda_1 + \lambda_2)e^{-(\lambda_1 + \lambda_2)d_1}\lambda_2\, e^{-\lambda_2 d_2}$$

$$= \frac{\lambda_2}{\lambda_1 + \lambda_2}(\lambda_1 + \lambda_2)e^{-(\lambda_1 + \lambda_2)d_1}\lambda_1\, e^{-\lambda_1 d_2}$$

$$= \lambda_1 \lambda_2 e^{-(\lambda_1 + \lambda_2) d_1} \left[e^{-\lambda_1 d_2} + e^{-\lambda_2 d_2} \right].$$

Note that d_1 is independent of d_2. By integrating over d_1, the marginal distribution of the second spacing is

$$h'_{2,2}(d_2) = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} \left[e^{-\lambda_1 d_2} + e^{-\lambda_2 d_2} \right]$$

which is the exponential mixture distribution characteristic of the marginal spacings.

5.2.2.2 N=3

For this case define $\lambda = (\lambda_1, \lambda_2, \lambda_3)$, $\underline{t} = (t_1, t_2, t_3)$ and note that since there are six permutations of three numbers, there are six π's. From $w(\pi)$ in (1) one again obtains

$$P(\pi = (\pi_1, \pi_2, \pi_3)) = P(t_{\pi_1} \leq t_{\pi_2} \leq t_{\pi_3})$$

$$= \lambda_{\pi_1} \lambda_{\pi_2} / ((\lambda_{\pi_2} + \lambda_{\pi_3})(\lambda_{\pi_1} + \lambda_{\pi_2} + \lambda_{\pi_3})).$$

Therefore, again by (6), the joint distribution is the exponential product mixture:

$$h_3(d_1, d_2, d_3) = \lambda_1 \lambda_2 \lambda_3 e^{-\sum_{i=1}^{3} \lambda_i d_1} \Big[e^{-(\lambda_2 + \lambda_3)d_2} \Big(e^{-\lambda_3 d_3} + e^{-\lambda_2 d_3} \Big)$$

$$+ e^{-(\lambda_1 + \lambda_3)d_2} \Big(e^{-\lambda_1 d_3} + e^{-\lambda_3 d_3} \Big)$$

$$+ e^{-(\lambda_1 + \lambda_2)d_2} \Big(e^{-\lambda_1 d_3} + e^{-\lambda_2 d_3} \Big) \Big]$$

As before by integrating over d_1 and d_3 the marginal distribution of the second spacing is obtained as:

$$h'_{2,3}(d_2) = \Big[\lambda_1 (\lambda_2 + \lambda_3) e^{-(\lambda_2 + \lambda_3)d_2} + \lambda_2 (\lambda_1 + \lambda_3) e^{-(\lambda_1 + \lambda_3)d_2}$$

$$+ \lambda_3 (\lambda_1 + \lambda_2) e^{-(\lambda_1 + \lambda_2)d_2} \Big] / (\lambda_1 + \lambda_2 + \lambda_3)$$

which agrees with the exponential mixture distribution given in reference (1).

The distribution of the third spacing can be found in similar fashion:

$$h'_{3,3}(d_3) = \frac{\lambda_1\,\lambda_2(A+B)\,\lambda_3}{ABD}\,e^{-\lambda_3 d_3} + \frac{\lambda_1\,\lambda_3(C+B)\,\lambda_2}{CBD}\,e^{-\lambda_2 d_3}$$

$$+ \frac{\lambda_2\,\lambda_3(A+C)\,\lambda_1}{ACD}\,e^{-\lambda_1 d_3}$$

where $A = \lambda_1 + \lambda_3$

$B = \lambda_2 + \lambda_3$

$C = \lambda_1 + \lambda_2$

$D = \lambda_1 + \lambda_2 + \lambda_3$.

As expected, $h'_{3,3}$ is also an exponential mixture.

5.2.2.3 Distribution of the Rate Parameter Given the Number of Errors Corrected

Littlewood [6] gives a Bayesian analysis of the prediction problem for the traditional software experiment, i.e., where forecasting is based on a single series of observed spacings. In that reference, he derives the posterior distribution of the parameter of the time to next error detection conditioning, not only on the number of errors corrected, but also on the total life, τ , of the program observed to that time. The gamma prior on the failure rate of the individual bugs transforms into a gamma posterior for the failure rate of the time to next detection. The parameters of the posterior gamma reflect not only the fewer numbers of remaining bugs in the program but also the change in the scale parameter of the gamma by an amount equal to the observed life. This has a reciprocal effect on the standard deviation of the parameter. Thus the distribution of the posterior is shifting closer to zero for two reasons: one, there are numerically fewer parameters, and two, by living, the program tells the observer to bet on those that are smaller and smaller. That is with age comes rigidity. There is still a third effect causing the shift toward zero in that by observing the order statistics of the process the errors are being removed roughly in the order of decreasing failure rate. This section quantifies this effect for some special cases and suggests a general conjecture.

In the context of the replicated experiment, it is interesting to know what is happening to the distribution of the parameter of the next spacing as a function of the number of errors corrected keeping in mind that the distribution of the spacing is not exponential with a single parameter, but an exponential mixture with several parameter possibilities. The cases for N=2 and N=3 have been examined and density functions for a general prior have been obtained.

The general conjecture on the effect of the "ordered" removal of failure rates is as follows:

For the j'th spacing if G_{N-j+1} is the probability distribution function for the sum of N-j+1 independent, identically distributed parameters with Bayesian prior density g, then the posterior distribution $G_{j,N}$ of the actual failure rate of the j'th spacing satisfies

$$G_{j,N}(x) > G_{N-j+1}(x) \quad \text{for all } x$$

Proof: (For N=2 second spacing and N=3, second and third spacings).

In the following assume that the failure rates $\lambda_1,\ldots, \lambda_N$ represent a random sample from some distribution with density g(x). Given λ_i the conditional distribution of the time to failure Z_i for bug i is exponential with failure rate λ_i. The $Z_1,\ldots, Z_n$ are assumed to be independent as in Section 5.2.1. The following analysis concentrates on N=2 and N=3, the general case being a conjecture at this point.

For the second spacing for N=2, let λ be the parameter remaining after the first error has been corrected, i.e.

$$\begin{aligned} \lambda &= \lambda_1 && \text{if } Z_2 < Z_1 \\ &= \lambda_2 && \text{if } Z_1 < Z_2 \end{aligned}$$

Then

$$\begin{aligned} P(\lambda \le x) &= P(\lambda_1 \le x, Z_2 < Z_1) + P(\lambda_2 \le x, Z_1 < Z_2) \\ &= 2\,P(\lambda_2 \le x, Z_1 < Z_2), \end{aligned}$$

where the second equality follows from symmetry considerations.

Further

$$\begin{aligned} P(\lambda \le x) &= 2\,E(I_{[\lambda_2 \le x]}\,P(Z_1 < Z_2 \mid \lambda_1, \lambda_2)) \\ &= 2\int_0^x\int_0^\infty \frac{\lambda_1}{\lambda_1 + \lambda_2}\; g(\lambda_1)\,g(\lambda_2)\,d\lambda_1\,d\lambda_2 \\ &= G_{2,2}(x). \end{aligned}$$

The marginal density of the remaining parameter λ after one error has been corrected, can then be obtained by differentiating:

$$g_{2,2}(x) = g(x)\left[2\int_{o}^{\infty}\frac{\lambda_1}{\lambda_1 + x} \quad g(\lambda_1)\, d\lambda_1\right].$$

The bracketed expression is the modifier of the original prior g due to the removal of an error. The major question then is, "What is its effect"?

The intuitively appealing answer is that the distribution $G_{2,2}$ of λ is stochastically smaller than G, the cumulative distribution of the original prior density g, i.e., the distribution $G_{2,2}$ is shifted toward zero or

$$G_{2,2}(x) > G(x) \quad \text{for all} \quad x > o.$$

To demonstrate this note that $g_{2,2}(x)/g(x)$ is strictly decreasing in x, i.e., the monotone likelihood ratio property is satisfied. The stated result is thus a direct consequence of Corollary 1, p. 67 or Theorem 2, p. 68 of Lehmann [7].

For the second spacing for N=3 let

$$\begin{aligned}\lambda &= \lambda_2 + \lambda_3 \quad \text{if } Z_1 < Z_2 \text{ and } Z_1 < Z_3\\ &= \lambda_1 + \lambda_3 \quad \text{if } Z_2 < Z_1 \text{ and } Z_2 < Z_3\\ &= \lambda_1 + \lambda_2 \quad \text{if } Z_3 < Z_1 \text{ and } Z_3 < Z_2\end{aligned}$$

Again

$$\begin{aligned}P(\lambda \le x) &= 3\,P(\lambda_2 + \lambda_3 \le x, Z_1 < Z_2, Z_1 < Z_3)\\ &= 3\int_o^x\int_o^\infty \frac{\lambda_1}{\lambda_1 + \lambda_2} \quad g(\lambda_1)\, g_2(Z)\, d\lambda_1\, dz = G_{2,3}(x).\end{aligned}$$

Here g_2 is the convolution density of $\lambda_1 + \lambda_2$.

As before the density of $G_{2,3}$ is

$$g_{2,3}(x) = g_2(x) \cdot \left[3 \int_0^\infty \frac{\lambda_1}{\lambda_1 + X} \, g(\lambda_1) \, d\lambda_1 \right]$$

and

$$G_{2,3}(x) > G_2(x) \quad \text{for all } x > o,$$

where G_2 is the cumulative distribution function of g_2.

For the third spacing for N=3 let

$$\begin{aligned} \lambda &= \lambda_1 \qquad \text{if } Z_2 < Z_1 \text{ and } Z_3 < Z_1 \\ &= \lambda_2 \qquad \text{if } Z_1 < Z_2 \text{ and } Z_3 < Z_2 \\ &= \lambda_3 \qquad \text{if } Z_1 < Z_3 \text{ and } Z_2 < Z_3. \end{aligned}$$

As before

$$P(\lambda \leq x) = 3\,P(\lambda_3 \leq x, Z_1 < Z_3, Z_2 < Z_3)$$

$$= 3 \int_0^x \int_0^\infty \int_0^\infty \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3} \left(\frac{1}{\lambda_2 + \lambda_3} + \frac{1}{\lambda_1 + \lambda_3} \right) g(\lambda_1)\, g(\lambda_2)\, g(\lambda_3)\, d\lambda_1\, d\lambda_2\, d\lambda_3$$

$$= G_{3,3}(x)$$

with density

$$g_{3,3}(x) = g(x) \left[3 \cdot \int_0^\infty \int_0^\infty \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2 + x} \left(\frac{1}{\lambda_2 + x} + \frac{1}{\lambda_1 + x} \right) g(\lambda_1)\, g(\lambda_2)\, d\lambda_1\, d\lambda_2 \right]$$

Again one concludes that

$$G_{3,3}(x) > G(x) \quad \text{for all } x > o.$$

Although the above stochastic ordering pattern generalizes to any N and any spacing, the proof was determined too late for inclusion in this document. Results of similar spirit, but in a different setting, were obtained by Pledger and Proschan [8].

5.3 THE TRADITIONAL EXPERIMENT: MODELS AND METHODS OF PREDICTION

For the traditional software debugging experiment a single partial replicate is observed, i.e., one series of spacings ending with the k'th observed bug. Forecasting the future behavior of the program based on this information is then of major interest. Based on the theory developed in Sections 5.1 and 5.2, multinomial and exponential models are developed for this case and methods of forecasting are suggested. The section ends with a discussion of the similarities of the two methods of modeling and gives a sufficient condition under which the discrete case can be approximated by its continuous analog.

5.3.1 Multinomial Case

Using the notation of Section 5.1 where it was assumed that the first k bugs are observed, let $Q = (Q_1, \ldots, Q_N)$ be some permutation of the N integers 1 through N and let Q^* denote the (k-1)-vector consisting of the first k-1 components of Q. With the usual convention that $\sum_a^b = 0$ when $b < a$, note that

$$\Lambda_i(\pi) = \sum_{j=i}^{N} p_{\pi_j} = 1 - p_o \sum_{j=1}^{i-1} p_{\pi_j} \qquad i=1, 2, \ldots, k$$

remain fixed as long as only $\pi_k, \ldots, \pi_N$ vary while holding the projection $\pi^* = h(\pi) = (\pi_1, \ldots, \pi_{k-1})$ fixed. Thus the notational convention $\Lambda_i(\pi^*) = \Lambda_i(\pi)$ for $i \leq k$ poses no problems. Further let

$$\underline{P}_N^* = \{\pi^* : \pi^* = h(\pi),\ \pi \in \underline{P}_N\}$$

and

$$w^*(\pi^*) = \sum_{\substack{\pi \in \underline{P}_N \\ h(\pi) = \pi^*}} w(\pi) = P(Q^* = \pi^*) \tag{9}$$

Thus (4) and (5) may be rewritten as follows:

$$P(D_i = d_i,\ i=1, \ldots, k) = \sum_{\pi^* \in \underline{P}^*_N} w^*(\pi^*) \prod_{i=1}^{k} \Lambda_i(\pi^*)(1-\Lambda_i(\pi^*))^{d_i-1} \tag{10}$$

$$= \sum_{\pi^* \in \underline{P}^*_N} P(Q^*=\pi^*)P(D_i=d_i,\ i=1, \ldots, k/Q^*=\pi^*)$$

where

$$P(D_i = d_i,\ i=1, \ldots, k \,|\, Q^*=\pi^*) = \prod_{i=1}^{k} \Lambda_i(\pi^*)(1-\Lambda_i(\pi^*))^{d_i-1}. \tag{11}$$

At this point, it is not yet clear whether formula (10) or (11) represents the most appropriate basis for inference concerning the reliability of the software that is being debugged. To this end the various aspects to this problem are reconsidered.

i) N, $p_1, \ldots, p_N$ and p_0 represent unknown parameters which (in conjunction with the independence assumption of the multinomial trials) describe the probabilistic structure of the debugging process.

ii) Q^* is an unobservable random label vector whose distribution is determined by the quantities N, $p_1, \ldots, p_N$, cf. (1) and (9).

iii) $D_1, \ldots, D_k$ are the spacings between the first k detected bugs.

Since only one (although unknown) realization of $Q^* \in \underline{P}^*_N$ pertains, just as only one (unknown) set of parameter values N, $p_1, \ldots, p_N$ pertains it would seem reasonable to treat the unknown value π^* of Q^* as a parameter just like $p_1, \ldots, p_N$ and N.

This then leads to

$$P(D_1 = d_i, \ldots, D_k = d_k/N, p_1, \ldots, p_N, Q^* = \pi^*)$$

$$= \prod_{i=1}^{k} \Lambda_i(\pi^*)(1-\Lambda_i(\pi^*)^{d_i-1} \tag{12}$$

as the appropriate basis for inference rather than

$$P(D_1 = d_1, \ldots, D_k = d_k/N, p_1, \ldots, p_N)$$

$$= \sum_{\pi^* \in \underline{P}^*_N} w^*(\pi^*) \prod_{i=1}^{k} \Lambda_i(\pi^*)(1-\Lambda_i(\pi^*))^{d_i-1} \qquad (13)$$

which averages over all possible realizations of Q*, namely the one that pertained and all of those N(N-1)...(N-k+2)-1 which did not pertain in this particular data case.

Note that the form of the likelihood of $d_1, \ldots, d_k$ as given in (12) presents certain identifiability problems in that many different sets of parameters $N(\geq k)$, $p_1, \ldots, p_N$ and π^* lead to the same set of parameters $\Lambda_1 = \Lambda_1(\pi^*), \ldots, \Lambda_k = \Lambda_k(\pi^*)$. Since the likelihood (12) depends on the unknown parameters only through $\Lambda_1, \ldots, \Lambda_k$ it seems natural to reparameterize (12) in terms of its natural identifiable parameters $\Lambda_1, \ldots, \Lambda_k$ which satisfy the following constraint: $\Lambda_1 \geq \ldots \geq \Lambda_k$.

With this problem reformulation, the parameter N has been eliminated, which was unidentifiable anyway.

The objective of this inference is to learn something about Λ_k which presents an upper bound for the error probability Λ_{k+1} of the program after having observed $d_1, \ldots, d_k$. Inference concerning Λ_{k+1} cannot be made since Λ_{k+1} is not part of our likelihood. But since $\Lambda_k \geq \Lambda_{k+1}$ upperbounds on Λ_k will be useful conservative upper bounds on Λ_{k+1}.

The maximum likelihood estimates for $\Lambda_1 > \ldots > \Lambda_k$ were derived out of context by Barlow, et al., [9] p. 42, 43, and are equivalent to isotonic regression estimates of $\Lambda_1, \ldots, \Lambda_k$. The m.l.e.'s are

$$\hat{\Lambda}_i = \left(\max_{1 \leq s \leq i} \; \min_{k \geq t \geq i} \frac{\sum_{j=s}^{t} d_j}{t-s+1} \right)^{-1} \qquad i=1, \ldots, k \qquad (14)$$

In particular

$$\hat{\Lambda}_k = \left(\max_{1 \leq s \leq k} \sum_{j=s}^{k} d_j/(k-s+1) \right)^{-1} \qquad (15)$$

$$= \min_{1 \leq s \leq k} \frac{k-s+1}{\sum_{j=s}^{k} d_j}$$

The distribution of $\hat{\Lambda}_k$ involves all k unknown parameters $\Lambda_1 \geq ... \geq \Lambda_k$. Further work needs to be done to develop confidence upper bounds for Λ_k (and hence for Λ_{k+1}).

5.3.2 Exponential Case

The parallels with the preceding discrete model are obvious in spite of the fact that the waiting times in the discrete case are dependent through the multinomial model whereas in the continuous case they are assumed to be independent.

Again, only the first k ($\leq$ N) spacings $\tilde{D}_1, ..., \tilde{D}_k$ are observed. Their joint density obtained from (6) is

$$h_{k,N}(d_1, ..., d_k) = \sum_{\pi * \epsilon \underline{P}^*_N} w^*(\pi *) \prod_{i=1}^{k} \Lambda_i(\pi *) \exp(-\Lambda_i(\pi *)d_i) \qquad (16)$$

using the same * notational convention as in (10). Conditionally, given Q* = π *, the density of $\tilde{D}_1, ..., \tilde{D}_k$ is

$$g_{k,N}(d_1, ..., d_k/\pi *) = \prod_{i=1}^{k} \Lambda_i(\pi *) \exp(-\Lambda_i(\pi *)d_i) \ . \qquad (17)$$

The same arguments can be made for preferring (17) over (16) as the basis for inference. By reparametrizing as before, the identifiability problem is avoided, and the following likelihood as a basis for inference concerning the natural identifiable parameters is considered:

$$g_k(d_1, ..., d_k \mid \pi *) = \prod_{i=1}^{k} \Lambda_i \exp(-\Lambda_i d_i) \qquad (18)$$

$$\text{with } \Lambda_1 \geq ... \geq \Lambda_k.$$

The parallel between the two approaches is not accidental. It turns out that (18) and (12) approximate each other quite well as demonstrated in the following.

Let $D^*_i = \left[\tilde{D}_i\right] + 1$ $\quad$ ($\left[X\right]$ = greatest integer $\leq$ X).

Then

$$P(D_i^* = d_i,\ i=1, \ldots, k \mid \pi *) = P(d_i-1 \leq \widetilde{D}_i < d_i,\ i=1, \ldots, k \mid \pi *)$$

$$= \prod_{i=1}^{k} \exp(-\Lambda_i(d_i-1))(1-\exp(-\Lambda_i))$$

$$= \prod_{i=1}^{k} (1-\Lambda_i^*)^{d_i-1}\ \Lambda_i^* \tag{19}$$

with $\Lambda_i^* = 1 - \exp(-\Lambda_i)$, $i=1, \ldots, k$.

Note that (19) and (12) are identical except for the use of Λ_i^* vs. Λ_i. Further note that $\Lambda_1 \longrightarrow 0$ implies that $\max\limits_{1\leq i\leq k} |\Lambda_i - \Lambda_i^*| \longrightarrow 0$ which in turn implies the local approximation theorem:

$$|P(D_i = d_i,\ i=1, \ldots, k) - P(D_i^* = d_i,\ i=1, \ldots, k)| \longrightarrow 0$$

for all $d_i \geq 1$, $i = 1, \ldots, k$.

It is of interest to see to what extent a global approximation is also possible. To this end note that

$$\operatorname*{Sup}_{A} |P((D_1, \ldots, D_k) \epsilon A) - P((D_1^*, \ldots, D_k^*) \epsilon A)|$$

$$= \sum \left[P(D_i = d_i,\ i=1, \ldots, k) - P(D_i^* = d_i,\ i=1, \ldots, k) \right]^+$$

where $[x]^+ = x$ if $x \geq 0$

$= 0$ else .

Further

$$P(Di \;=\; di,\, i{=}1,\ldots,\, k) - P(D_i^* = d_i,\, i{=}1,\ldots,\, k)$$

$$= \sum_{j=1}^{k} \prod_{i=1}^{j} \Lambda_i (1-\Lambda_i)^{d_i - 1} \prod_{i=j+1}^{k} \Lambda_i^* (1-\Lambda_i^*)^{d_i - 1} \left(1 - \frac{\Lambda_j^*}{\Lambda_j} \left(\frac{1-\Lambda_j^*}{1-\Lambda_j} \right)^{d_j - 1} \right)$$

Since

$$\Delta = 1 - \frac{\Lambda^*}{\Lambda} \left(\frac{1-\Lambda^*}{1-\Lambda} \right)^{d-1} = 1 - \frac{\Lambda^*}{\Lambda} + \frac{\Lambda^*}{\Lambda} \left(1 - \left(\frac{1-\Lambda^*}{1-\Lambda} \right)^{d-1} \right)$$

$$\leq 1 - \frac{\Lambda^*}{\Lambda} = 1 - \left(\frac{1-e^{-\Lambda}}{\Lambda} \right) \leq \frac{\Lambda}{2}$$

we have

$$\sum_{d} \left[P(D_i = d_i,\, i = 1,\, \ldots,\, k) - P(D_i^* = d_i,\, i{=}1,\, \ldots,\, k) \right]^+ \leq \sum_{j=1}^{k} \Lambda_j/2 \,. \qquad (20)$$

Hence, it appears that the global approximation depends on $\sum_{j=1}^{k} \Lambda_j$, and not just on Λ_1. This is somewhat disturbing, especially if k is large and it is not clear whether the error bound (20) represents the best possible result.

Inference concerning $\Lambda_1 > \ldots > \Lambda_k$ based on the conditional likelihood (18) can again proceed along the lines of Barlow, et al., [9] and was indeed proposed by Campbell and Ott [10]. The m.l.e.'s of $\Lambda_1 \geq \ldots \geq \Lambda_k$ are

$$\hat{\Lambda}_i \;=\; (\max_{s \leq i} \;\; \min_{t \geq i} \sum_{j=s}^{t} d_j/(t-s+1))^{-1}$$

and in particular

$$\hat{\Lambda}_k = (\max_{s \leq k} \sum_{j=s}^{k} d_j/(k-s+1))^{-1}$$

$$= \min_{s \leq k} \frac{k-s+1}{\sum_{j=s}^{k} d_j} \tag{21}$$

To obtain a crude upper confidence bound for Λ_k note that

$$\hat{\Lambda}_k/\Lambda_k = (\max_{s \leq k} \sum_{j=s}^{k} D_j \Lambda_j \frac{\Lambda_k}{\Lambda_j} /(k-s+1))^{-1}$$

$$\geq (\max_{s \leq k} \sum_{j=s}^{k} D_j \dot{\Lambda}_j/(k-s+1))^{-1} = T_k.$$

Note that the distribution of T_k is independent of all unknown parameters. Thus, the distribution of T_k may be obtained through extensive simulations. Suppose $t_{p,k}$ is the p percentile of the T_k distribution then

$$\gamma = P(T_k \geq t_{1-\gamma,k}) \leq P(\hat{\Lambda}_k/\Lambda_k \geq t_{1-\gamma,k})$$

$$= P(\hat{\Lambda}_k/t_{1-\gamma,k} \geq \Lambda_k)$$

i.e., $\hat{\Lambda}_k/t_{1-\gamma,k}$ is a conservative 100 γ % upper confidence bound for Λ_k.

Another ad hoc procedure would be to obtain (by simulation perhaps) the distribution of

$$\max_{s \le k} \sum_{j=s}^{k} D_j \Lambda_j \frac{\Lambda_k}{\Lambda_j} \quad /(k-s+1)$$

for a fixed set of ratios Λ_k/Λ_j, j = 1, ..., k, where for lack of any other better choice $\hat{\Lambda}_k/\hat{\Lambda}_j$ would be substituted for Λ_k/Λ_j. Having obtained this distribution, we again proceed as above to obtain confidence bounds for Λ_k. The validity of such an approach would need some testing, of course.

5.3.3 Forecasting

Forecasts of the rate for Λ_{k+1} from the observations through the k'th error have been calculated using formula (21) as a function of selected program states or nodes as illustrated in the graphs of Section 3. Nodes were selected that were far enough down the structure to have relatively small rates of occurrence, and at the same time to be on the path of a large number of runs. Final nodes were excluded from consideration. These nodes were selected as interesting cases to forecast having enough replication to stabilize both the forecast and the estimate of the parameter being forecasted. Nodes were also selected from the data in reference [1].

Table 5.3.3-1 provides a list of the forecasts and the estimated parameter being forecast for the nodes selected from both experiments. In calculating this table, samples that had a multiple error leading directly to node entry or to an exit from the node were omitted from the sample. In interpreting this table, two things should be noted. There are many paths coming into a node, but there is only one parameter of interest leaving the node, namely the sum of the failure rates of all the errors still remaining in the program. It can be shown, in part from the distribution of d_k obtained by integration from (16), that the distribution of d_k conditioned on a set of detected errors in any order rather than on a particular π^*, is also exponential. This conclusion relies heavily on the fact that Λ_i depends only on the sum of the failure of the remaining errors and not on the order of the previous i - 1 detections. Thus, the time to next program failure leaving a particular node is exponential. Since it is exponential, the MLE estimator is the familiar number of failures over the total time on test computed only over those paths leading to and from the selected node. These computations appear in the last column of Table 5.3.3-1.

Replication is used in this table to improve the forecast. The first two of the three forecast columns of Table 5.3.3-1 are based on the isotonic regression estimator of equation (21). The first of these gives the average of the forecasts across all of the replications for that node as an upper bound on the exiting failure rate. This is a conservative estimate. Since each of the estimates represents an upper bound, it is also tempting to use the min of the upper bounds as an estimate. This is the second of the two forecasts given. Unfortunately, the minimum is a function of sample size and gets smaller and smaller as the sample size increases. As this process is understood more completely, some other statistic of these forecasts that compensates for the effect of sample size, might prove to have better properties than the average.

Table 5.3.3-1

FORECASTS OF THE (K + 1)ST PARAMETER CONDITIONED
ON A SPECIFIC K'TH NODE
WITH INDEPENDENT ESTIMATES OF THE PARAMETER

				Forecasts		Data Based
Program	Node	S. Size	Avg. U.B.	Min U.B.	Regr	Estimate
Current Study:						
A3*	127	15	3.74x10^{-2}	8.13x10^{-3}	5.4x10^{-3}	1.00x10^{-2}
B3*	125	25	2.15x10^{-2}	2.98x10^{-3}	9.9x10^{-4}	3.98x10^{-3}
D1	12	22	1.41x10^{-3}	1.68x10^{-4}	6.1x10^{-4}	2.27x10^{-4}
D1	13	25	1.77x10^{-3}	2.28x10^{-4}	6.1x10^{-4}	2.71x10^{-4}
E1	123	32	2.72x10^{-1}	2.50x10^{-2}	4.6x10^{-2}	5.05x10^{-2}
E1	1234	47	1.46x10^{-1}	1.43x10^{-2}	2.2x10^{-2}	7.36x10^{-4}
E1	12345	32	9.71x10^{-3}	2.16x10^{-4}	5.9x10^{-4}	4.41x10^{-4}
E1	12346	18	2.16x10^{-3}	8.83x10^{-4}	5.9x10^{-4}	5.52x10^{-4}
D3	1245	17	4.72x10^{-2}	8.62x10^{-3}	5.8x10^{-3}	8.18x10^{-3}
Study No. 1:						
A1	124	20	7.23x10^{-3}	7.63x10^{-4}	1.8x10^{-3}	1.54x10^{-3}
A1	1234	16	2.34x10^{-3}	6.55x10^{-4}	1.1x10^{-4}	9.10x10^{-4}
B1R**	14	39	2.88x10^{-2}	2.02x10^{-3}	5.1x10^{-4}	2.98x10^{-3}
B1R**	145	44	7.91x10^{-3}	8.68x10^{-4}	3.8x10^{-4}	4.80x10^{-4}
A3	1256	16	2.90x10^{-2}	6.54x10^{-3}	1.0x10^{-2}	1.85x10^{-2}
B3	1245	14	6.74x10^{-3}	1.52x10^{-3}	7.8x10^{-4}	2.44x10^{-3}

**Corrected Data Base for B1

The third forecast is a crude estimate somewhat related to Cox's model proposed in reference [1] which assumes that the stages are independent and exponentially distributed. Although these assumptions are known to be analytically false, they are approximately true as demonstrated empirically in Sections 4.7 and 4.8 with the reservations already noted. The forecasts are linear least squares regression estimates based on regressing the log of the unconditional stage probabilities existing at the time of the forecast on the number of errors corrected and forecasting the next stage.

The table indicates that in every case the average upper bound is a true upper bound although somewhat conservative as expected. In all but four cases, the minimum of the upper bounds underestimated the estimated parameter even with modest sample sizes. The regression estimator was less consistent giving a somewhat closer estimate than the average but without the safety margin. In one case, E1, node 1234, all of the estimates were high due to the substantial change in character of the error mix of the next error.

Table 5.3.3-2 provides forecasts of the same parameters based on another method of forecasting. From the estimates of the specific error probabilities in Table 4.1-1 and the estimate of the first stage probability (which estimates $1-p_0$), the total rate for the remaining parameters at a node can be estimated. That is, by subtracting from one the sum of p_0 plus the probabilities of the errors associated with the node, an estimate of the rate of the errors remaining is obtained. In applicable cases this estimate was improved by subtracting estimates of the joint probability when the intersection was observed in the data. Adjustments for three-way intersections were also made when observed. The last column in this table is an estimate of the remaining parameter based on all of the errors observed downstream from the node but unobserved prior to the node. This column offers another check on the validity of the forecasts in both tables, but to a degree is correlated with the node estimate. This is not the case with the data from the next spacing.

At times the estimates in this table had to be computed by omitting data from highly probable errors. The questionable importance of these errors has previously been discussed in Section 4. When they are used in this context at these sample sizes, a negative estimate sometimes occurs. By omitting these errors, valid estimates can often be obtained. This was not the case for the nodes for E1 omitted from Table 5.3.3-2. None of these nodes were estimable based on this method as all estimates were negative. Comparable estimates for the nodes of study #1 have not been computed.

Table 5.3.3-3 compares the regression estimator and the error probability forecast with a 95% confidence interval on the rate associated with the next spacing. 53% of the regression forecasts lie within the corresponding confidence intervals and 67% of the forecasts based on the error probabilities. All of the average upper bounds were above the upper confidence limit. Since the forecasts are also random variables, the sampling variation in the forecast contributes to the forecasting error, but no attempt has been made to compensate for this effect.

Table 5.3.3-2

FORECASTS OF THE (K + 1)ST PARAMETER CONDITIONED
ON A SPECIFIC K'TH NODE
BASED ON ERROR PROBABILITY ESTIMATES

			Observed	
			Next Spacing	Error Prob.
Prog.	Node	Forecast	Estimate	Estimate
Current Study:				
A3*	127	6.59×10^{-3}	1.00×10^{-2}	1.31×10^{-2}
B3*	125	1.98×10^{-3}	3.98×10^{-3}	2.70×10^{-3}
D1	12	1.23×10^{-4}	2.27×10^{-4}	2.67×10^{-4}
D1	13	1.07×10^{-4}	2.71×10^{-4}	2.54×10^{-4}
E1	123	1.40×10^{-2}	5.05×10^{-2}	2.82×10^{-2}
D3	1245	5.37×10^{-3}	8.18×10^{-3}	8.72×10^{-3}

Table 5.3.3-3

NEXT NODE RATE FORECASTS COMPARED TO 95% CONFIDENCE INTERVALS ON THE OBSERVED RATE

		Forecasts		Confidence Limits	
Program	Node	Regression	Error Prob.	Lower	Upper
Current Study:					
A3*	127	$5.4x10^{-3}$	$6.6x10^{-3}$	$3.60x10^{-3}$	$1.57x10^{-2}$
B3*	125	$9.9x10^{-3}$	$2.0x10^{-3}$	$1.87x10^{-3}$	$5.69x10^{-3}$
D1	12	$6.1x10^{-3}$	$1.2x10^{-4}$	$1.01x10^{-4}$	$3.31x10^{-4}$
D1	13	$6.1x10^{-3}$	$1.1x10^{-4}$	$1.27x10^{-4}$	$3.87x10^{-4}$
E1	123	$4.6x10^{-2}$	$1.4x10^{-2}$	$2.62x10^{-2}$	$6.94x10^{-2}$
E1	1234	$2.2x10^{-2}$		$4.33x10^{-4}$	$9.61x10^{-4}$
E1	12345	$5.9x10^{-4}$		$2.29x10^{-4}$	$6.06x10^{-4}$
E1	12346	$5.9x10^{-4}$		$2.21x10^{-4}$	$8.35x10^{-4}$
D3	1245	$5.8x10^{-3}$	$5.4x10^{-3}$	$3.17x10^{-3}$	$1.25x10^{-2}$
Study No. 1:					
A1	124	$1.8x10^{-3}$		$6.51x10^{-4}$	$2.29x10^{-3}$
A1	1234	$1.1x10^{-4}$		$3.41x10^{-4}$	$1.41x10^{-3}$
B1R**	14	$5.1x10^{-4}$		$1.66x10^{-3}$	$3.99x10^{-3}$
B1R**	145	$3.8x10^{-4}$		$2.77x10^{-4}$	$6.32x10^{-4}$
A3	1256	$1.0x10^{-2}$		$6.93x10^{-3}$	$2.86x10^{-2}$
B3	1245	$7.8x10^{-4}$		$8.42x10^{-4}$	$3.87x10^{-3}$

**Corrected Data Base for B1

6.0 CONCLUSIONS

This document reports on the second of two studies investigating issues associated with the software debugging process, by means of replicated software experiments. The current study differs from the previous study primarily with regard to the factors or experimental treatments explored in the design. While the data has been used to provide independent verification of some of the issues of the first study, such as 1) the wide range of probabilities with which errors occur, 2) the linearity of the log stage failure rates as a function of the number of errors corrected, and 3) the degree to which interfailure time is exponential, it has also been used to suggest relationships between the error structure of programs and such issues as the impact of programmer experience, the impact of a change in usage and the effect of programming in a low level language.

Although the number of subjects was necessarily low due to time and monetary constraints, the experimental evidence suggests the following very tentative conclusions:

1. Changes in usage change the detection rates of some individual errors but impact the stage probabilities with a recognizable pattern.

2. The use of a higher-order language by an inexperienced programmer is somewhat similar to a low-level language in the hands of an experienced professional.

3. The slope of the linear log stage probability function is distributed over a fairly narrow range over many different test treatments. The effect of experience is primarily to change the intercept rather than the slope.

Out of need to explain and compute summary estimates of the observations, an analytic framework was developed. This framework has proved valuable in explaining the results of replication as well as in explaining the traditional software experiment.

Although it can be demonstrated relative to this framework that stages are neither independent nor exponentially distributed, empirical estimates show that the exponential assumption is nearly valid for all but the extreme tails of the distribution. Empirical studies of the nature of the dependence of a stage on its past indicate that some of the estimated correlations are high and demonstrate a curious periodicity.

Except for the degree of dependence in the stage probabilities, it still appears that Cox's model approximates to a degree what is being observed. Additionally the slopes of the log stage probabilities are somewhat similar varying in data experiments from .5 to 1.5 except for two of the problem specifications that appeared to be too simple for error propagation.

A method of forecasting an upper bound on the rate of the next spacing has been developed. Numerical forecasts have been made, conditioned on nodes in the network of error states for several programs of both studies. These are compared to estimates of rate actually observed. Both estimates are improved due to replication and the bound, though conservative, appears useful. Other forecasting methods are compared, one based on a somewhat quick and dirty application of Cox's model appears to be in the neighborhood of the correct answer about 50% of the time, the others based on estimates of the specific error probabilities is in the neighborhood nearly 70% of the time for the cases examined.

This study has concentrated on theory developed simultaneously with experimental verification. The two together have led to a synergism of ideas that has strengthened both. Much work remains if the process is to be understood; building the analytic foundation has only just begun, understanding forecasting in this context is in its infancy and compensating for external factors has only been suggested with a bare minimum of subjects.

Unfortunately this study has created more issues for investigation than it has explained. It has, however, demonstrated the power of replication. Replication has provided the stability necessary for comparing test treatments and in testing the efficiency of new estimators. Whether this method of keeping books will be useful in its own right in providing better information with which to forecast the future requires further study.

REFERENCES

1. Nagel, P. M. and Skrivan, J. A., "Software Reliability: Repetitive Run Experimentation and Modeling," NASA CR-165836, 1982.

2. Brown, J. R. and Buchanan, H. N., "The Quantitative Measurement of Software Safety and Reliability," TRW SDP 1776, TRW Systems Group, Redondo Beach, California, 1973.

3. Page, R. L., "Algorithm 479 A Minimal Spanning Tree Clustering Method," Communications of the ACM, Vol. 17, No. 6, pp. 321-323, June 1974.

4. Dijkstra, E. W., "A Note on Two Problems in Connection with Graphs," Numer. Math. 1, pp. 269-271, October 1959.

5. Zahn, C. T., "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters," IEEE Transactions on Computers, Vol. C-20, No. 1, pp. 68-86, January 1971.

6. Littlewood, Bev, "Stochastic Reliability-Growth: A Model for Fault-Removal in Computer-Programs and Hardware-Designs," IEEE Transactions on Reliability, Vol. TR-80-51, October 1981.

7. Lehmann, E. L., Testing Statistical Hypotheses, John Wiley & Sons, 1959.

8. Pledger, Gorden and Proschen, Frank, "Comparisons of Order Statistics and of Spacings from Heterogeneous Distributions," In: Optimizing Methods in Statistics, J. S. Rustagi, Ed. Academic Press, New York 1971.

9. Barlow, R. E., Bartholomew, D. J., Bremner, J. M., and Brunk, H. D., Statistical Inference Under Order Restrictions, John Wiley and Sons, 1972.

10. Campbell, G. and Ott, K. O., "Statistical Evaluation of Major Human Errors during the Development of New Technological Systems," Nuclear Science and Engineering, 71, pp. 267-279, 1979.

APPENDIX A: SOFTWARE ERROR CATEGORIES

From Brown and Buchanan [2]

A000		COMPUTATIONAL ERRORS
	A100	Incorrect operand in equation
	A200	Incorrect use of parenthesis
	A300	Sign convention error
	A400	Units or data conversion error
	A500	Computation produces an over/under flow
	A600	Incorrect/inaccurate equation used
	A700	Precision loss due to mixed mode
	A800	Missing computation
	A900	Rounding or truncation error
B000		LOGIC ERRORS
	B100	Incorrect operand in logical expression
	B200	Logic activities out of sequence
	B300	Wrong variable being checked
	B400	Missing logic or condition tests
	B500	Too many/few statements in loop
	B600	Loop iterated incorrect number of times (including endless loop)
	B700	Duplicate logic
C000		DATA INPUT ERRORS
	C100	Invalid input read from correct data file
	C200	Input read from incorrect data file
	C300	Incorrect input format
	C400	Incorrect format statement referenced
	C500	End of file encountered prematurely
	C600	End of file missing
D000		DATA HANDLING ERRORS
	D050	Data file not rewound before reading
	D100	Data initialization not done
	D200	Data initialization done improperly
	D300	Variable used as a flag or index not set properly
	D400	Variable referred to by the wrong name
	D500	Bit manipulation done incorrectly
	D600	Incorrect variable type
	D700	Data packing/unpacking error
	D800	Sort error
	D900	Subscripting error

APPENDIX A: SOFTWARE ERROR CATEGORIES (Continued)

E000		DATA OUTPUT ERRORS
	E100	Data written on wrong file
	E200	Data written according to the wrong format statement
	E300	Data written in wrong format
	E400	Data written with wrong carriage control
	E500	Incomplete or missing output
	E600	Output field size too small
	E700	Line count or page eject problem
	E800	Output garbled or misleading
F000		INTERFACE ERRORS
	F100	Wrong subroutine called
	F200	Call to subroutine not made or made in wrong place
	F300	Subroutine arguments not consistent in type, units, order, etc.
	F400	Subroutine called is nonexistent
	F500	Software/data base interface error
	F600	Software/user interface error
	F700	Software/software interface error
G000		DATA DEFINITION ERRORS
	G100	Data not properly defined/dimensioned
	G200	Data referenced out of bounds
	G300	Data being referenced at incorrect location
	G400	Data pointers not incremented properly
H000		DATA BASE ERRORS
	H100	Data not initialized in data base
	H200	Data initialized to incorrect value
	H300	Data units are incorrect
I000		OPERATION ERRORS
	I100	Operating system error (vendor supplied)
	I200	Hardware error
	I300	Operator error
	I400	Test execution error
	I500	User misunderstanding/error
	I600	Configuration control error

APPENDIX A: SOFTWARE ERROR CATEGORIES (Continued)

J000		OTHER
	J100	Time limit exceeded
	J200	Core storage limit exceeded
	J300	Output line limit exceeded
	J400	Compilation error
	J500	Code or design inefficient/not necessary
	J600	User/programmer requested enhancement
	J700	Design nonresponsive to requirements
	J800	Code delivery or redelivery
	J900	Software not compatible with project standards
K000		DOCUMENTATION ERRORS
	K100	User manual
	K200	Interface specification
	K300	Design specification
	K400	Requirements specification
	K500	Test documentation
X0000		PROBLEM REPORT REJECTION
	X0001	No problem
	X0002	Void/withdrawn
	X0003	Out of scope - not part of approved design
	X0004	Duplicates another problem report
	X0005	Deferred

APPENDIX B
PROBLEM #4 SPECIFICATIONS

1.0 CONSTRUCTION OF MINIMAL SPANNING TREE

Given n 2-dimensional coordinates (nodes)

(Xi, Yi), i = 1,2,-,n $\quad 1 \leq n \leq 25$

Problem Connect the nodes to construct a network of direct node-to-node branches having the smallest possible total length (sum of the branch lengths). This network is called the Minimal Spanning Tree (MST).

Algorithm

The branches are subdivided into three sets:

I. Branches which are definitely assigned to the network under construction.

II. Branches from which the next branch to be added to set I will be selected.

III. The remaining branches (rejected or not yet considered).

The nodes are subdivided into two sets:

A. Nodes connected to the branches of set I.

B. The remaining nodes (one and only one branch of set II will lead to each of these nodes).

We start the construction by choosing node 1 as the only member of set A, and by placing all branches that end in this node in set II. To start with, set I is empty. From then onward, we perform the following two steps repeatedly.

Step 1. The shortest branch of set II is removed from this set and added to set I. As a result, one node is transferred from set B to set A.

Step 2. Consider the branches leading from the node, which have just been transferred to set A, to all the nodes which are still in set B. If the branch under consideration is equal to or longer than the corresponding branch (i.e., the branch with the same node) in set II, it is rejected; if it is shorter, it replaces the corresponding branch in set II, and the latter is rejected.

Let i represent node i and ij represent the branch from node i to node j. Then, for example, assume set I = (12), set II = (13, 14, 15), A = (1, 2), and B = (3, 4, 5), after Step 1 when node 2 was added to set A. We then compare 23 with 13, 24 with 14, and 25 with 15 and replace in set II any shorter corresponding branch.

2.0 DETERMINATION OF CLUSTER MEMBERSHIP

Problem Given a Minimal Spanning Tree, and parameters f, s and d, calculate the number of clusters among the n nodes and the cluster membership of each node.

Algorithm

Clusters are determined by separations of the nodes. We detect inherent separations in the data by deleting branches from the MST which are significantly longer than nearby branches. Such a branch is called inconsistent. (We will say a node P is nearby node Q if P is connected to Q by a path in the MST containing d or fewer branches.) The criteria to determine an inconsistent branch are: (1) the branch's length is more than f times the average length of nearby branches; and (2) its length is more than s standard deviations larger than the average of the lengths of nearby branches (standard deviation computed on the lengths of nearby branches).

Deleting the inconsistent branch(es) breaks the MST into several connected subnetworks. The points of each connected subnetwork are the members of a cluster.

Two FORTRAN subroutines are required: NETWRK and KLUSTR.

3.0 DESCRIPTION OF SUBROUTINE NETWORK

Communication

```
SUBROUTINE NETWRK (N,NODE,DIST,NBR)
INTEGER N, NBR(1)
REAL NODE(2,25), DIST(25,25)
```

Input

N, number of nodes, $1 \leq N \leq 25$
NODE(1,i), X coordinate of node i.
NODE(2,i), Y coordinate of node i.

Output

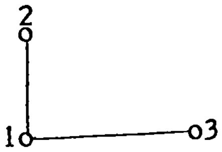
DIST(i,j), distance from node i to node j.
DIST(i,i) = 0. for all i.

NBR(1) = i, the node number
NBR(2) = m, the number of neighbors of node i in the minimum-total-length network. (A neighbor is any node connected to node i with a single branch.)
NBR(3) = i_1, the node number of the first neighbor of node i.
.
.
.
NBR(2+m) = i_m, the node number of the m^{th} neighbor of node i.

Note: i_1, i_2, · · ·, i_m shall appear in ascending order.

Repeat this set for all nodes, going in ascending node number order. The subscript of NBR increases with all entries.

Example of NBR for N = 3.



NBR(1) =	1	Node number 1
NBR(2) =	2	2 neighbors of node 1
NBR(3) =	2	Neighbor node number
NBR(4) =	3	Neighbor node number
NBR(5) =	2	Node number 2
NBR(6) =	1	1 neighbor of node 2
NBR(7) =	1	Neighbor node number
NBR(8) =	3	Node number 3
NBR(9) =	1	1 neighbor of node 3
NBR(10 =	1	Neighbor node number

4.0 DESCRIPTION OF SUBROUTINE KLUSTR

Communication

```
SUBROUTINE KLUSTR(N,NODE,F,S,D,DIST,NBR,C,CMEM)
INTEGER N,NBR(1),C,CMEM(N),D
REAL NODE(2,25),DIST(25,25),F,S
```

Input

N, NODE same as in subroutine NETWRK
F, length factor to determine inconsistent branch
S, standard deviation factor to determine inconsistent branch
D, number of branches to define nearby branches $1 \leq D \leq N$
DIST,NBR, same as in subroutine NETWRK

Output

C, number of clusters
CMEM(i), cluster number for node i $1 \leq \text{CMEM(i)} \leq C$

Number the clusters in increasing order; i.e., CMEM(1) = 1, then CMEM(2) = 1 if in same cluster or CMEM(2) = 2 if in different cluster. Continue for all N nodes.

APPENDIX C

PROBLEM #4 TEST CASES

1.0 Test Case 1

- **Input**

N = 5, F = 2.0, S = 1.5, D = 4

NODE (1,i):	3.058	2.821	3.062	6.193	6.249
NODE (2,*i*):	-1.713	-2.054	-3.976	-5.418	-2.694

- **Output**

DIST (*i*,*j*):

				j		
		1	2	3	4	5
	1	0.0	0.42	2.26	4.85	3.34
	2	0.42	0.0	1.94	4.76	3.49
i	3	2.26	1.94	0.0	3.45	3.44
	4	4.85	4.76	3.45	0.0	2.72
	5	3.34	3.49	3.44	2.72	0.0

NBR (*i*): 1 2 2 5 2 2 1 3 3 1 2 4 1 5 5 2 1 4

C = 1

CMEM (*i*): 1 1 1 1 1

2.0 Test Case 2

- **Input**

N = 10, F = 2.0, S = 1.5, D = 4

NODE (1,*i*):	-8.171 5.409	-6.485 3.237	-5.041 3.134	-5.146 3.851	-7.788 6.739
NODE (2,*i*):	0.564 5.707	-2.392 4.787	3.136 2.333	-2.099 4.745	-0.850 4.830

o Output

```
DIST (i,j):
                                         j

       1      2      3      4      5      6      7      8      9      10

   1   0.0    3.40   4.05   4.03   1.47   14.52  12.16  11.44  12.73  15.51
   2   3.40   0.0    5.71   1.37   2.20   14.39  12.09  10.72  12.56  15.07
   3   4.05   5.71   0.0    5.24   4.84   10.76  8.44   8.21   9.04   11.90
   4   4.03   1.37   5.24   0.0    2.92   13.13  10.85  9.39   11.30  13.76
   5   1.47   2.02   4.84   2.92   0.0    14.74  12.38  11.38  12.91  15.60
i  6   14.52  14.39  10.76  13.13  14.74  0.0    2.36   4.07   1.83   1.59
   7   12.16  12.09  8.44   10.85  12.38  2.36   0.0    2.46   0.62   3.50
   8   11.44  10.72  8.21   9.39   11.38  4.07   2.46   0.0    2.52   4.39
   9   12.73  12.56  9.04   11.30  12.91  1.83   0.62   2.52   0.0    2.89
  10   15.51  15.07  11.90  13.76  15.60  1.59   3.50   4.39   2.89   0.0

NBR (i):   1  2  3  5  2  2  4  5  3  2  1  8  4  1  2  5  2  1  2
           6  2  9 10  7  2  8  9  8  2  3  7  9  2  6  7 10  1  6

C = 2

CMEM (i):  1  1  1  1  1  2  2  2  2  2
```

APPENDIX D: EXPERIMENT DATA FOR SUBJECT PROGRAM A3*

Explanation: Repetition No. Failure Case No. (Error No.)
Time to Failure in Elapsed CRU's

1.	2 (1)	6 (7)	12 (9)		
	1	4	9		
2.	3 (1)	38 (4)			
	0	17			
3.	5 (1)	56 (3)	55 (2)	18 (8)	
	3	30	27	8	
4.	1 (1)	1 (2)	35 (3)	66 (8)	
	0	0	19	37	
5.	2 (1)	61 (2)	39 (6)	23 (7)	172 (9)
	1	28	17	12	93
6.	2 (1)	24 (4)			
	2	12			
7.	1 (1)	4 (2)	61 (7)	121 (4)	
	1	1	32	66	
8.	2 (1)	35 (2)	143 (8)		
	1	15	74		
9.	4 (1)	32 (7)	5 (2)	65 (4)	
	2	18	2	34	
10.	4 (1)	20 (2)	116 (7)	47 (8)	
	1	11	65	35	
11.	2 (1)	23 (6)	1 (10)		
	0	13	0		
12.	1 (1)	26 (6,10)			
	0	12			
13.	1 (1)	9 (7)	4 (2)	131 (4)	
	0	5	1	65	
14.	1 (1)	28 (7)	45 (2)	61 (6)	26 (10)
	1	15	28	34	14
15.	1 (1)	34 (2)	12 (4)		
	1	16	3		
16.	8 (1)	12 (3)	12 (7,9)		
	8	6	6		
17.	8 (1)	3 (2)	39 (7)	126 (3,8)	
	3	2	25	63	
18.	3 (1)	25 (2)	123 (7)	46 (3)	48 (6,10)
	2	15	58	27	26
19.	8 (1)	111 (2)	10 (6,10)		
	4	57	7		
20.	1 (1)	30 (3)	25 (2)	25 (7)	40 (4)
	1	14	13	12	22
21.	10 (1)	54 (7)	10 (2)	242 (6)	331 (10)
	5	27	3	128	191
22.	4 (1)	138 (2)	27 (6,10)		
	2	75	14		
23.	1 (1)	23 (2)	16 (7)	25 (6,10)	
	1	11	8	21	
24.	9 (1,2)	56 (7)	305 (4)		
	6	31	179		
25.	2 (1)	4 (6,10)			
	1	3			

APPENDIX D: EXPERIMENT DATA FOR SUBJECT PROGRAM A3*

Explanation: Repetition No. Failure Case No. (Error No.)
Time to Failure in Elapsed CRU's

26.	5 (7)	8 (1)	40 (2)	100 (4)	
	3	3	25	54	
27.	4 (1)	18 (4)			
	2	11			
28.	1 (1)	55 (2)	118 (4)		
	0	28	67		
29.	4 (1)	5 (6,10)			
	1	3			
30.	2 (1)	16 (2)	39 (9)		
	1	8	20		
31.	11 (1)	6 (2)	134 (3)	42 (6,10)	
	8	5	63	20	
32.	3 (1)	2 (2)	3 (9)		
	1	0	1		
33.	2 (1)	124 (2,7)	178 (3)	19 (6)	
	0	70	87	10	
34.	8 (1)	60 (3)	27 (4)		
	5	29	18		
35.	2 (1)	18 (2)	78 (4)		
	0	4	37		
36.	9 (2)	1 (1)	19 (7)	23 (6,10)	
	7	0	11	11	
37.	2 (1)	39 (7)	2 (6)	18 (2)	70 (10)
	0	16	1	11	39
38.	3 (1)	14 (6,10)			
	2	7			
39.	8 (1)	44 (6)	88 (2)	18 (10)	
	3	25	48	15	
40.	1 (1)	2 (7)	55 (2)	149 (3)	81 (4)
	1	1	33	92	42
41.	6 (1)	50 (4)			
	3	28			
42.	11 (9)				
	7				
43.	1 (1)	29 (2)	31 (6)	110 (7,10)	
	1	18	15	67	
44.	6 (1)	22 (7)	13 (2)	41 (8)	
	4	11	9	26	
45.	3 (1)	31 (2)	33 (4)		
	1	17	17		
46.	3 (1)	8 (3)	3 (2)	55 (7)	45 (9)
	3	4	0	32	25
47.	3 (1)	77 (2)	25 (8)		
	2	39	18		
48.	1 (1)	44 (3)	19 (2)	9 (7)	132 (4)
	0	28	13	5	67
49.	1 (1)	44 (3)	41 (7)	30 (4)	
	1	28	30	14	
50.	1 (1)	7 (2)	53 (7)	12 (8)	
	0	3	29	7	

APPENDIX E: EXPERIMENT DATA FOR SUBJECT PROGRAM B3*

Explanation: Repetition No. Failure Case No. (Error No.)
Time to Failure in Elapsed CRU's

1.	1 (1)	63 (2)	104 (4)	16 (7)	
	0	28	49	8	
2.	10 (1)	129 (2)	153 (4)	77 (7)	
	5	53	69	35	
3.	7 (1)	71 (5)	18 (2)	183 (7)	
	3	36	11	72	
4.	1 (1)	34 (4,5)	44 (2)	1147 (7)	
	0	14	18	483	
5.	1 (1)	15 (2)	72 (4)	181 (5)	357 (7)
	0	6	31	83	169
6.	2 (1)	45 (2)	32 (7)		
	1	22	17		
7.	2 (1)	122 (5)	48 (7)		
	1	50	22		
8.	5 (1)	106 (5)	97 (2)	237 (4)	1097 (7)
	2	47	39	100	472
9.	1 (1)	74 (5)	223 (2)	3 (4)	98 (7)
	0	30	88	1	49
10.	2 (1)	26 (2)	28 (5)	274 (7)	
	1	12	10	126	
11.	3 (1)	210 (2)	312 (5)	325 (6)	
	2	93	133	150	
12.	1 (1)	11 (5)	16 (2)	76 (4)	455 (7)
	0	3	6	36	188
13.	1 (1)	17 (2)	25 (5)	303 (4)	259 (7)
	0	7	12	130	119
14.	6 (1)	55 (2)	149 (4)	57 (7)	
	1	24	60	21	
15.	2 (1)	16 (2)	28 (4)	149 (5)	226 (7)
	0	9	12	68	99
16.	9 (1)	6 (2)	38 (7)		
	5	2	18		
17.	1 (1)	5 (2,7)			
	0	3			
18.	2 (1)	11 (2,4)	14 (5)	134 (7)	
	1	6	7	60	
19.	4 (1)	12 (2)	295 (5)	363 (7)	
	2	7	133	148	
20.	1 (1)	27 (5)	40 (2)	442 (7)	
	0	15	17	191	
21.	1 (1)	14 (5)	52 (2)	633 (4)	419 (7)
	1	5	25	268	187
22.	7 (1)	39 (2)	144 (5)	404 (6)	
	2	17	66	181	
23.	3 (2)	1 (1)	185 (5)	1 (6)	
	1	0	79	1	
24.	1 (1)	52 (2)	16 (5)	135 (7)	
	0	22	7	62	
25.	1 (1)	37 (2)	51 (5)	36 (4)	40 (7)
	0	21	23	14	16

APPENDIX E: EXPERIMENT DATA FOR SUBJECT PROGRAM B3*

Explanation: Repetition No. Failure Case No. (Error No.)
Time to Failure in Elapsed CRU's

26.	3 (1)	40 (2,4)	34 (7)			
	1	17	15			
27.	4 (1)	33 (2)	44 (7)			
	1	17	19			
28.	1 (1)	1 (2)	185 (6)			
	1	0	82			
29.	1 (1)	25 (7)				
	1	11				
30.	3 (1)	114 (7)				
	1	48				
31.	3 (1)	194 (2)	283 (5)	458 (4)	41 (7)	
	2	86	132	204	20	
32.	1 (1)	108 (2)	210 (4)	41 (7)		
	0	42	84	25		
33.	1 (1)	80 (2)	336 (5)	161 (7)		
	0	34	137	71		
34.	5 (1)	66 (5)	114 (2)	56 (4)	268 (7)	
	3	30	54	22	117	
35.	2 (2)	2 (1)	70 (4)	216 (5)	215 (7)	
	1	0	30	97	99	
36.	2 (1)	57 (5)	42 (2)	35 (4)	256 (3)	582 (7)
	1	25	19	15	113	266
37.	1 (1)	18 (2)	23 (4)	20 (5)	351 (7)	
	1	7	9	8	160	
38.	2 (1)	52 (5)	188 (2,4)	302 (6)		
	1	18	92	130		
39.	2 (1)	24 (7)				
	1	9				
40.	1 (1)	28 (7)				
	0	12				
41.	1 (1)	16 (2)	2 (5)	42 (7)		
	1	7	0	18		
42.	2 (1)	64 (2)	72 (5)	28 (4)	20 (7)	
	1	27	34	12	12	
43.	3 (1)	25 (5)	5 (7)			
	1	13	2			
44.	2 (1)	36 (2)	182 (5)	124 (4)	419 (3)	667 (7)
	0	18	79	56	193	269
45.	2 (1)	31 (2)	137 (5)	954 (7)		
	1	15	65	407		
46.	4 (1)	122 (2)	628 (3,5)	208 (4)	124 (7)	
	2	52	285	98	58	
47.	2 (1)	14 (2)	48 (5)	71 (4)	391 (7)	
	2	5	21	29	178	
48.	1 (1)	18 (2)	60 (5)	177 (7)		
	1	8	27	75		
49.	1 (1)	61 (2)	79 (7)			
	1	27	36			
50.	2 (1)	32 (5)	18 (2)	767 (7)		
	0	17	9	343		

APPENDIX F: EXPERIMENT DATA FOR SUBJECT PROGRAM C3

Explanation: Repetition No. Failure Case No. (Error No.)

1.	2 (3)	2 (1)	1 (2)	6 (4)	33 (5)	131 (6)
	248 (9)	636 (7)				
2.	3 (1)	2 (2,3)	3 (4)	97 (5)	143 (6,7)	2079 (8)
	1050 (10)					
3.	1 (3)	1 (1)	11 (2,4)	354 (6,7)	2 (9)	
4.	4 (1,2,3,4)	73 (5,6,7)	189 (9)			
5.	4 (1,2)	4 (6)	1 (3,4)	33 (5)	264 (9)	66 (7)
6.	2 (2,3,5)	2 (1,4)	215 (6)	12 (7)	151 (11)	
7.	1 (1)	2 (2,3)	1 (4)	101 (5)	462 (6,7)	910 (8)
	5263 (11)					
8.	1 (2)	1 (3)	4 (1)	2 (4)	96 (6)	24 (10)
	10 (9)					
9.	2 (1)	5 (2,4)	4 (3)	66 (7)	228 (5)	69 (6)
	240 (8)	674 (11)				
10.	1 (1,4)	3 (2,3)	120 (5)	319 (6)	62 (11)	
11.	2 (2,3)	4 (1,4)	124 (8)	33 (5)	2 (6,7)	1213 (9)
12.	1 (1,3)	14 (4)	1 (2)	521 (6,7)	86 (5)	104 (9)
13.	6 (3)	2 (2)	2 (1)	21 (4)	69 (6)	35 (7)
	28 (8)	128 (5)	4401 (11)			
14.	5 (1)	3 (2,3)	7 (4)	75 (7)	144 (9)	41 (5)
15.	1 (1,4)	1 (2,3)	4 (6,7)	58 (5)	159 (8)	332 (9)
16.	6 (1,2,3,4)	42 (7)	87 (5)	207 (6)	286 (9)	
17.	4 (1)	2 (2,3,4)	195 (7)	183 (5)	1019 (11)	
18.	3 (1)	3 (2,3)	26 (4)	67 (5)	137 (7)	54 (6)
	669 (8)	60 (9)				
19.	1 (3)	1 (1)	4 (2)	12 (4)	5 (7)	151 (5)
	559 (6)	778 (9)				
20.	4 (1)	2 (2,3)	2 (4)	74 (5,6)	644 (9)	43 (7)
21.	3 (1,2,3,4)	88 (5)	348 (7)	4 (6)	643 (8)	22 (11)
22.	4 (5)	3 (1)	3 (2,4)	1 (3)	73 (9)	1077 (7)
23.	1 (1)	7 (2,3)	12 (4)	89 (7)	65 (5)	281 (6)
	696 (9)					
24.	4 (3)	4 (1)	4 (2)	9 (4)	1 (7)	116 (6)
	195 (5)	1563 (11)				
25.	1 (1)	3 (2,3)	3 (4)	12 (7)	346 (11)	93 (8)
26.	4 (1)	2 (2)	4 (3)	4 (4)	31 (5)	89 (6,7)
	385 (8)	493 (9)				
27.	1 (3)	1 (1)	2 (7)	1 (2)	1 (4)	65 (5)
	1116 (6)	4160 (10)				
28.	1 (1)	6 (3,4)	6 (2)	129 (7)	455 (5)	261 (11)
29.	1 (1,2,3)	1 (4)	98 (5)	109 (6,8)	144 (7)	27 (9)
30.	3 (2,3)	1 (1)	9 (4)	305 (5)	91 (6,7)	1811 (13)
31.	3 (2,3)	5 (1)	1 (4)	2 (5)	69 (12)	207 (9)
32.	1 (1)	4 (3)	7 (2)	4 (4)	13 (6,7)	659 (5)
	891 (9)					
33.	1 (1,3,4)	8 (2)	5 (6)	85 (5)	377 (10)	
34.	3 (1,3,4)	4 (5)	2 (2)	9 (8)	107 (6,7)	510 (9)
35.	1 (1)	3 (2,3)	8 (4)	129 (7)	117 (6)	64 (5)
	2535 (9)					
36.	1 (1)	4 (2,3,4)	96 (5)	65 (9)	902 (11)	
37.	3 (1,3,4)	4 (2)	176 (6,7)	463 (5)	1140 (8)	356 (9)

APPENDIX F: EXPERIMENT DATA FOR SUBJECT PROGRAM C3

Explanation: Repetition No. Failure Case No. (Error No.)

38.	1 (1,2,3,4)	48 (5)	200 (6,10)			
39.	1 (1,2)	2 (3)	8 (4)	194 (7)	123 (5)	584 (9)
40.	1 (1,3)	7 (2)	2 (4)	201 (5)	512 (6,7)	599 (8)
	2687 (9)					
41.	3 (1)	4 (2,4)	12 (3)	222 (8)	97 (5)	292 (7)
	636 (11)					
42.	1 (1,3)	4 (2,4)	411 (6)	58 (8)	144 (11)	
43.	4 (1)	9 (3,4)	1 (2)	48 (5)	26 (6)	168 (7)
	12 (8)	39 (9)				
44.	1 (1,2,3,4)	11 (7)	173 (5)	571 (6)	628 (9)	
45.	2 (1,4)	1 (3)	2 (2)	51 (7)	496 (5)	539 (6,8)
	82 (9)					
46.	1 (1,2,3)	5 (4)	22 (6)	53 (7)	404 (5)	1102 (9)
47.	1 (2)	5 (3)	1 (1,4)	39 (6,7)	109 (11)	
48.	1 (1,2,3,4)	92 (5,6)	164 (7)	1021 (13)		
49.	4 (3)	1 (1,2)	5 (4)	59 (6,7)	46 (11)	
50.	1 (1)	4 (2,3)	1 (4)	28 (5)	598 (9)	267 (8)

APPENDIX G: EXPERIMENT DATA FOR SUBJECT PROGRAM D3

Explanation: Repetition No. Failure Case No. (Error No.)

1.	1 (1)	3 (2)	26 (4,5)	72 (7)	51 (3)	1039 (6)
	16 (8)	1898 (10)				
2.	1 (1)	3 (2)	45 (4)	10 (5)	257 (3)	112 (6)
	6 (8)	1032 (10)				
3.	2 (1,2)	23 (4)	115 (5)	500 (3)	111 (7)	1969 (10)
	9887 (14)					
4.	1 (1)	10 (2)	5 (3)	16 (4)	69 (5)	1059 (8)
	909 (10)	256 (7)				
5.	3 (2)	2 (1,4)	45 (3,6)	96 (5)	461 (8)	129 (7)
	404 (10)					
6.	1 (1)	1 (2)	29 (4)	16 (5)	100 (3)	2052 (8)
	1081 (7)	1025 (10)				
7.	1 (1,2)	23 (4)	8 (5)	17 (3)	83 (7)	4140 (6)
	8842 (8)	2120 (10)				
8.	1 (1)	1 (2)	55 (4)	75 (5)	17 (7)	33 (3)
	929 (8)	597 (10)				
9.	3 (1,2)	62 (4)	116 (5)	147 (3)	472 (7)	2551 (8)
	4353 (10)					
10.	1 (1)	4 (3)	1 (2)	88 (6)	20 (4)	40 (5)
	372 (8)	940 (10)				
11.	1 (1)	1 (2)	24 (4)	3 (3)	129 (7)	255 (5)
	3514 (10)	12613 (12)				
12.	1 (1,2)	58 (4)	36 (5)	105 (3)	777 (6)	83 (7)
	2380 (10)					
13.	1 (1)	4 (2)	13 (4)	37 (7)	43 (5)	6 (3)
	435 (6)	789 (10)				
14.	2 (1,2)	5 (4)	5 (5)	17 (3)	175 (6)	323 (7)
	3865 (9)					
15.	1 (2)	1 (1)	1 (3)	44 (4,5)	698 (7)	210 (8)
	462 (10)					
16.	2 (1,2)	36 (4)	112 (3)	57 (5)	348 (6)	857 (7)
	248 (8)	440 (10)				
17.	1 (1)	3 (2)	82 (4,5)	220 (3)	227 (6)	898 (7)
	4906 (8)	310 (10)				
18.	1 (1,2)	4 (4)	25 (5)	179 (7)	46 (3)	634 (6)
	2261 (8)	2947 (10)				
19.	1 (1)	2 (2)	24 (4)	23 (3)	237 (7)	132 (5)
	866 (10)	27273 (11)				
20.	3 (1)	4 (2)	13 (5)	17 (4)	65 (3)	1165 (6)
	70 (8)	1236 (10)				
21.	2 (1,2)	50 (4,5)	62 (7)	64 (3)	89 (8)	2504 (10)
22.	1 (1)	1 (2)	1 (3)	26 (4)	109 (5)	137 (7)
	5662 (8)	2359 (10)				
23.	1 (1)	3 (2)	20 (4,5)	9 (3)	2285 (6)	171 (10)
	5491 (12)					
24.	2 (2)	3 (1)	26 (4,5)	164 (3)	434 (7)	1040 (8)
	2890 (10)					
25.	1 (1)	3 (2)	24 (4)	21 (3)	3 (5)	375 (7)
	108 (6)	445 (8)	330 (9)			
26.	3 (2)	2 (1)	14 (4,5)	210 (3)	390 (7)	412 (6)
	308 (8)	2162 (10)				

APPENDIX G: EXPERIMENT DATA FOR SUBJECT PROGRAM D3

Explanation: Repetition No. Failure Case No. (Error No.)

27.	1 (1)	2 (2)	32 (4)	24 (5)	133 (7)	213 (3)
	292 (6)	1532 (8)	1771 (10)			
28.	1 (1)	1 (2)	24 (4)	93 (3)	118 (5)	833 (7)
	462 (8)	758 (10)				
29.	1 (1)	3 (4)	5 (2)	73 (5)	130 (3)	393 (6)
	361 (7)	8285 (8)	622 (10)			
30.	1 (2)	3 (1)	52 (4)	26 (3)	115 (5)	707 (6)
	22 (7)	1855 (8)	1252 (10)			
31.	1 (2)	1 (1)	15 (5)	60 (4)	114 (3)	135 (6)
	110 (8)	38 (10)				
32.	1 (1)	4 (2)	2 (4)	55 (5)	77 (3)	1007 (8)
	89 (10)	391 (13)				
33.	1 (1,2)	5 (4)	89 (7)	173 (3)	351 (5)	80 (6)
	765 (8)	4801 (10)				
34.	1 (1)	3 (2)	17 (3)	2 (4,5)	543 (10)	4171 (7)
	4262 (11)					
35.	1 (1)	6 (2)	1 (5)	44 (4)	28 (3)	771 (9)
	862 (8)	2292 (10)				
36.	1 (2)	10 (1)	104 (3,6)	39 (4)	113 (5)	1058 (10)
	622 (7)					
37.	1 (1)	2 (2)	41 (3)	12 (4,5)	258 (7)	120 (8)
	3207 (10)					
38.	1 (2)	1 (1)	56 (4)	98 (3)	91 (5)	949 (7)
	455 (10)	9328 (15)				
39.	1 (2)	2 (1)	1 (4)	8 (5)	120 (3)	2232 (7)
	42 (6)	137 (8)	444 (10)			
40.	3 (2)	1 (1)	57 (4)	29 (3)	272 (10)	254 (5)
	1281 (7)	5914 (9)				
41.	1 (2)	2 (4)	1 (1)	7 (6)	34 (5)	35 (3)
	1717 (9)	714 (7)				
42.	1 (1)	2 (2)	55 (3)	25 (4)	87 (7)	172 (5)
	143 (8)	1382 (10)				
43.	3 (2)	1 (1)	11 (4)	25 (3)	88 (5)	344 (7)
	614 (6)	675 (8)	294 (10)			
44.	2 (2)	4 (1)	23 (4)	98 (6)	81 (3)	89 (5)
	1057 (8)	3549 (7)	43 (10)			
45.	1 (1,2)	6 (4)	17 (3)	215 (7)	186 (5)	1061 (6)
	1789 (10)					
46.	3 (2)	1 (1)	26 (4)	18 (3)	123 (8)	13 (5)
	1088 (7)	710 (10)				
47.	2 (1)	3 (2)	5 (4)	8 (3)	3 (5)	743 (6)
	199 (7)	2477 (10)				
48.	1 (1,2)	16 (4)	8 (3)	84 (5)	648 (6)	86 (7)
	62 (11)					
49.	1 (2)	1 (1)	25 (4)	22 (3)	70 (5)	335 (10)
	1037 (7)	4270 (9)				
50.	1 (1)	2 (2)	13 (4)	63 (8)	7 (3)	77 (5)
	306 (10)	924 (7)				

APPENDIX H: EXPERIMENT DATA FOR SUBJECT PROGRAM D1

Explanation: Repetition No. Failure Case No. (Error No.)

1.	349 (2)	71 (1)	2091 (3)
2.	2852 (3)	879 (1)	7692 (2)
3.	507 (2)	1342 (1)	1668 (3)
4.	435 (1)	51 (3)	10 (4)
5.	605 (1)	812 (3)	2015 (2)
6.	300 (1)	2098 (2)	429 (3)
7.	611 (1)	85 (2)	17170 (3)
8.	364 (1)	12 (3)	7399 (2)
9.	212 (2)	653 (3)	513 (1)
10.	279 (1)	854 (2)	9283 (3)
11.	270 (1)	728 (2)	2653 (3)
12.	59 (1)	364 (2)	4697 (3)
13.	418 (2)	227 (1)	2751 (3)
14.	1317 (1,2)	8152 (3)	
15.	379 (1)	3049 (2)	4795 (3)
16.	462 (1)	5949 (2)	515 (3)
17.	128 (3)	2376 (1)	411 (2)
18.	1206 (1)	89 (3)	4964 (2)
19.	3501 (1)	1345 (3)	5341 (2)
20.	719 (3)	18 (1)	131 (2)
21.	423 (3)	1112 (1)	773 (2)
22.	657 (1)	2180 (2)	4659 (3)
23.	702 (1)	81 (2)	950 (3)
24.	1353 (3)	213 (1)	10629 (2)
25.	1201 (1)	3140 (2)	9277 (3)
26.	2396 (1)	1087 (2)	797 (3)
27.	1496 (1)	934 (3)	1599 (2)
28.	433 (1)	1671 (3)	2265 (2)
29.	588 (2)	4 (3)	234 (1)
30.	150 (1)	1359 (2)	1126 (3)
31.	33 (3)	130 (1)	5651 (2)
32.	50 (1)	247 (3)	1854 (2)
33.	1290 (1)	305 (2)	4230 (3)
34.	2274 (1)	1301 (2)	168 (3)
35.	1601 (1)	63 (2)	11952 (3)
36.	1319 (1)	1941 (3)	2663 (2)
37.	1553 (3)	982 (1)	527 (2)
38.	12 (1)	685 (2)	3244 (3)
39.	1851 (1)	828 (3)	1541 (2)
40.	346 (1)	1696 (3)	9764 (2)
41.	132 (2)	1313 (1)	12957 (3)
42.	301 (3)	1469 (1)	3025 (2)
43.	43 (1)	510 (2)	646 (3)
44.	615 (3)	12 (1)	540 (2)
45.	4 (1)	2066 (3)	11732 (2)
46.	66 (3)	1781 (1)	4864 (2)
47.	504 (3)	1623 (1)	3480 (2)
48.	297 (1)	189 (3)	2653 (2)
49.	280 (1)	4384 (3)	481 (2)
50.	1183 (2)	270 (1)	980 (3)

APPENDIX I: EXPERIMENT DATA FOR SUBJECT PROGRAM E1

Explanation: Repetition No. Failure Case No. (Error No.)

1.	2 (2,4)	4 (1)	3 (3)	376 (5)	939 (6)	
2.	4 (2)	2 (3)	3 (1)	45 (4)	1045 (6)	4055 (5)
3.	1 (2)	1 (1)	48 (4)	1 (3)	2591 (5)	412 (7)
4.	1 (2,3)	1 (1)	21 (4)	1119 (6)	529 (5)	
5.	2 (2)	2 (1)	5 (3)	9 (4)	2286 (5)	2080 (6)
6.	1 (2)	1 (1)	13 (4)	11 (3)	1738 (5)	388 (6)
7.	1 (1)	2 (2)	7 (3)	38 (4)	822 (5)	5827 (6)
8.	1 (2)	2 (1)	3 (3)	16 (4)	33 (5)	4108 (6)
9.	1 (2)	1 (1)	9 (4)	12 (3)	543 (6)	3889 (5)
10.	1 (1)	4 (2)	12 (4)	3 (3)	1167 (5)	5782 (6)
11.	1 (1)	1 (2)	35 (3)	6 (4)	49 (5)	1212 (6)
12.	1 (2)	3 (1)	2 (4)	6 (3)	2093 (6)	5507 (5)
13.	1 (1)	5 (2)	8 (3,4)	1369 (6)	373 (5)	
14.	1 (1,2)	17 (3)	23 (4)	444 (5)	1895 (6)	
15.	1 (2)	1 (1)	4 (3)	25 (5)	251 (4)	2989 (6)
16.	2 (2,4)	1 (1)	3 (3)	130 (5)	349 (6)	
17.	2 (1,2)	38 (3)	26 (4)	172 (6)	676 (5)	
18.	2 (1)	1 (2)	3 (3)	15 (4)	4632 (5)	670 (6)
19.	1 (1,2)	22 (3)	6 (4)	1302 (5)	250 (6)	
20.	2 (1)	2 (2)	11 (4)	23 (3)	1314 (5)	3130 (6)
21.	1 (2)	3 (1)	3 (3)	7 (4)	762 (6)	261 (5)
22.	1 (1)	2 (2)	1 (3)	3 (4)	629 (6)	5098 (5)
23.	1 (1)	2 (2)	40 (3)	18 (4)	846 (5)	1361 (6)
24.	1 (2)	1 (1)	20 (3)	7 (4)	1384 (5)	759 (6)
25.	1 (1)	3 (2)	5 (3)	57 (4)	3653 (6)	1279 (5)
26.	1 (3,4)	1 (1)	2 (2)	397 (5)	3217 (6)	
27.	1 (1,2)	6 (4)	45 (3)	466 (6)	1280 (5)	
28.	3 (1)	2 (2)	1 (3)	22 (4)	2075 (5)	4811 (6)
29.	2 (2)	1 (1)	6 (3)	16 (4)	887 (5)	977 (6)
30.	1 (2)	2 (1)	8 (3)	3 (4)	1828 (5)	5086 (6)
31.	1 (1)	1 (2)	13 (4)	14 (3)	2057 (5)	1921 (6)
32.	1 (1)	1 (2)	3 (3)	17 (4)	3337 (6)	1014 (5)
33.	1 (1,2)	11 (3)	9 (4)	134 (5)	3412 (6)	
34.	1 (2)	1 (1)	12 (3,4)	2 (5)	298 (6)	
35.	1 (2)	3 (1)	17 (3)	70 (4)	230 (5)	617 (6)
36.	1 (1)	5 (2)	15 (3)	12 (4)	110 (6)	3723 (5)
37.	1 (2)	3 (1)	3 (3)	53 (4)	3765 (5)	11434 (6)
38.	1 (3)	1 (1)	1 (2)	26 (4)	1991 (5)	1022 (6)
39.	1 (2)	3 (1)	11 (3)	8 (4)	985 (5)	522 (6)
40.	1 (2)	3 (1)	35 (3)	13 (4)	1136 (5)	3245 (6)
41.	1 (2)	1 (3)	2 (1)	28 (4)	173 (6)	123 (5)
42.	1 (1)	1 (2)	10 (3)	2 (4)	4310 (6)	982 (5)
43.	1 (1,2)	13 (3)	14 (4)	725 (6)	2038 (5)	
44.	3 (1)	1 (3)	2 (4)	1 (2)	1206 (5)	746 (6)
45.	2 (1,2)	1 (4)	1 (3)	429 (6)	1684 (5)	
46.	1 (1,2)	2 (4)	11 (3)	349 (6)	20 (5)	
47.	1 (1)	2 (2)	5 (3)	6 (4)	2665 (5)	1525 (6)
48.	1 (1,2)	1 (4)	1 (3)	3056 (5)	1105 (6)	
49.	1 (1)	1 (2)	2 (4)	23 (3)	1302 (5)	513 (6)
50.	1 (2)	1 (1)	1 (3)	13 (4)	1133 (6)	96 (5)

APPENDIX J: EXPERIMENT DATA FOR SUBJECT PROGRAM D4

Explanation: Repetition No. Failure Case No. (Error No.)

1.	19 (1)
2.	53 (1)
3.	46 (1)
4.	39 (1)
5.	9 (1)
6.	4 (1)
7.	15 (1)
8.	223 (1)
9.	44 (1)
10.	13 (1)
11.	5 (1)
12.	37 (1)
13.	27 (1)
14.	21 (1)
15.	13 (1)
16.	4 (1)
17.	141 (1)
18.	121 (1)
19.	50 (1)
20.	2 (1)
21.	36 (1)
22.	35 (1)
23.	11 (1)
24.	15 (1)
25.	4 (1)
26.	1 (1)
27.	32 (1)
28.	10 (1)
29.	16 (1)
30.	19 (1)
31.	24 (1)
32.	7 (1)
33.	6 (1)
34.	80 (1)
35.	30 (1)
36.	15 (1)
37.	10 (1)
38.	12 (1)
39.	20 (1)
40.	27 (1)
41.	31 (1)
42.	77 (1)
43.	10 (1)
44.	34 (1)
45.	26 (1)
46.	29 (1)
47.	3 (1)
48.	7 (1)
49.	1 (1)
50.	5 (1)

Note: 25,000 additional cases were executed with no second error detected.

APPENDIX K: EXPERIMENT DATA FOR SUBJECT PROGRAM E4

Explanation: Repetition No. Failure Case No. (Error No.)

1.	6 (1)	26.	11 (1)
2.	3 (1)	27.	2 (1)
3.	11 (1)	28.	4 (1)
4.	1 (1)	29.	1 (1)
5.	10 (1)	30.	2 (1)
6.	1 (1)	31.	8 (1)
7.	9 (1)	32.	32 (1)
8.	5 (1)	33.	4 (1)
9.	2 (1)	34.	9 (1)
10.	6 (1)	35.	4 (1)
11.	5 (1)	36.	1 (1)
12.	2 (1)	37.	16 (1)
13.	5 (1)	38.	7 (1)
14.	2 (1)	39.	4 (1)
15.	3 (1)	40.	2 (1)
16.	4 (1)	41.	4 (1)
17.	14 (1)	42.	1 (1)
18.	14 (1)	43.	3 (1)
19.	9 (1)	44.	4 (1)
20.	3 (1)	45.	1 (1)
21.	6 (1)	46.	1 (1)
22.	1 (1)	47.	2 (1)
23.	5 (1)	48.	17 (1)
24.	8 (1)	49.	2 (1)
25.	3 (1)	50.	5 (1)

Note: 22,000 additional cases were executed with no second error detected.

1. Report No.
NASA CR-172378

2. Government Accession No.

3. Recipient's Catalog No.

4. Title and Subtitle

Software Reliability: Additional Investigations Into Modeling with Replicated Experiments

5. Report Date
June 1984

6. Performing Organization Code

7. Author(s)
P. M. Nagel
F. M. Scholz
J. A. Skrivan

8. Performing Organization Report No.
BCS-40446

9. Performing Organization Name and Address

Boeing Computer Services Company
Space and Military Applications Division
P.O. Box 24346
Seattle, WA 98124

10. Work Unit No.

11. Contract or Grant No.
NAS1-16481

12. Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Washington, D.C. 20546

13. Type of Report and Period Covered
Contractor

14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Gerard E. Migneault

16. Abstract

This report documents the second of two studies on modeling the process of software error detection from the results of experiments specifically designed for these studies. Experiments consist of simulations conducted on code prepared under controlled conditions. Six separate codes were prepared and tested in each of the two studies. Each code was initialized to an original state, then executed with independent random inputs. Replication is introduced by repeating the entire process from initialization.

The current study enlarges on the previous one by exploring the effects of programmer experience level, different program usage distributions, and programming languages. All these factors affect performance, and some tentative relational hypotheses are presented.

An analytic framework for replicated and non-replicated (traditional) software experiments is presented. A method of obtaining an upper bound on the error rate of the next error is proposed. The method was validated empirically by comparing forecasts with actual data. In all 14 cases the bound exceeded the observed parameter, albeit somewhat conservatively. Two other forecasting methods are proposed and compared to observed results.

Although it can be demonstrated relative to this framework that stages are neither independent nor exponentially distributed, empirical estimates show that the exponential assumption is nearly valid for all but the extreme tails of the distribution. Except for the dependence in the stage probabilities, it appears that Cox's model approximates to a degree what is being observed.

17. Key Words (Suggested by Author(s))

Software reliability, software errors, software testing, reliability modelling, software experimentation

18. Distribution Statement

Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	122	

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