ANALYZING THE TEST PROCESS USING STRUCTURAL COVERAGE

James Ramsey
Victor R. Basili

University of Maryland
Department of Computer Science
College Park, Maryland, 20742, USA
(301) 454-2002

Abstract

A large, commercially developed FORTRAN program was modified to produce structural coverage metrics. The modified program was executed on a set of functionally generated acceptance tests and a large sample of operational usage cases. The resulting structural coverage metrics are combined with fault and error data to evaluate structural coverage in the SEL environment.

We can show that in this environment the functionally generated tests seem to be a good approximation of operational use. The relative proportions of the exercised statement subclasses (executable, assignment, CALL, DO, IF, READ, WRITE) changes as the structural coverage of the program increases. We also propose a method for evaluating if two sets of input data exercise a program in a similar manner.

We also provide evidence that implies that in this environment, faults revealed in a procedure are independent of the number of times the procedure is executed and that it may be reasonable to use procedure coverage in software models that use statement coverage. Finally, the evidence suggests that it may be possible to use structural coverage to aid in the management of the acceptance test process.

Introduction

The goal of this study has been to understand and improve the acceptance test process in the NASA Goddard Space Flight Center, Software Engineering Laboratory (SEL) environment [1]. Towards this end, a SEL program has been modified to produce structural coverage metrics. The instrumented program, the MAL language preprocessor, is a subset of the RADMAS satellite attitude maintenance system. It has 68 functions and subroutines, 10k source lines of code and 4k executable statements. The program was modified to measure both procedure coverage and statement coverage. Coverage is also computed for the following statement subclasses: assignment statements, CALL, DO, IF, READ, and WRITE.

The modified program was executed on a set of seventeen functionally generated acceptance tests and on sixty samples of actual operational inputs [2]. Error, fault and failure data* were collected from the system test through operation phases [4]. Each execution of an acceptance test or an operational usage case provides a structural coverage statistic. These structural coverage statistics are first examined individually to understand the static properties of the acceptance test process. Randomly generated sequences of acceptance tests and operational usage cases are then used to explore the dynamic properties of structural growth (the increase in total structural coverage as the program is executed with different inputs). Finally the coverage data are combined with the error, fault and failure data to understand how faults are revealed.

Goals of the Study

The first goal of this study was to characterize structural coverage in the SEL environment. (see Figure 1). The first three questions address the simple, static properties of structural coverage for the different kinds of inputs. The final question addresses the properties of structural coverage growth of a set of input cases.

Some testing strategies [5, 6] and software reliability models [7] require a method for showing that two sets of inputs exercise a program in a similar fashion. This motivated goal II: "Can different input sets be differentiated using structural coverage metrics?" Questions II.A-II.D explore several methods of doing this.

The purpose of the functional tests is to reveal faults in the program yet some faults are still revealed in operation. What classes of faults does functional testing miss? Does operational use exercise the code differently than the functional tests? How is this related to structural coverage? This motivated the next goal: "How are faults and structural coverage related?" Questions III.A - III.D analyze the SEL error, fault, and failure data with respect to structural coverage [4].

In the final section these ideas are combined to suggest an improved method of managing acceptance tests.

This study is funded by NASA grant NSG-5123.

* We have tried to follow the IEEE Standard Glossary of Software Engineering Terminology definitions of *error*, *fault* and *failure*: An *error* is the "human action that results in software containing a fault." A *fault* is "a manifestation of an error." A *failure* is "a departure of program operation from program requirements" [3]. Some of the sources we cite were written before the standard; their use of *error* may differ from the standard.

I. Characterize structural coverage in the SEL environment.

I.A. What is the statement coverage of functional testing? What is the procedure coverage of functional testing?

I.B. What is the statement coverage of operational use? What is the procedure coverage of operational use?

I.C. What is the intersection / union of functional testing and operational use?

I.D. What are the properties of structural coverage growth?

II. Differentiate different input sets using their structural coverage.

II.A. Are heavily exercised procedures more likely to contain a fault?

II.B. Can they be differentiated using Venn diagrams?

II.C. Can they be differentiated using nonparametric statistics?

II.D. Can they be differentiated using the number of executions of prime sections of code?

III. Relate errors, faults, failures and structural coverage.

III.A. Are more heavily exercised procedures more likely to contain a revealed fault?

III.B. Is procedure coverage related to time to isolate?

III.C. Is procedure coverage related to time to understand and implement?

III.D. Is procedure coverage related to type of error?

IV. Use structural coverage to aid in the management of acceptance tests.

IV.A. Can structural coverage be used to suggest new acceptance tests?

IV.B. Can structural coverage be used to improve reliability models?

Goal / Question Hierarchy
Figure 1.

Data and Analysis

This section contains a description of the data and their analysis paralleling the outline in figure 1.

Structural Coverage in the SEL

Question:

What is the statement coverage of functional testing?
What is the procedure coverage of functional testing?

The acceptance tests used were functional or "black box" tests [8, 9]. Since exhaustive sampling of the input subdomains is impractical, the testers chose a few sample inputs that they felt were likely to reveal faults from the subdomains [2]. There are 17 acceptance tests.

Table 1 shows the structural coverage of the acceptance tests. Test 1 exercised 33 out of 68 possible procedures. It exercised 1069 of the 4300 executable statements. In total, the 17 tests exercised 51 procedures and 2408 executable statements (Union). There were 778 executable statements that were exercised by every test case (Intersection).

Statement Coverage of the MAL Preprocessor by 17 Benchmark Test Cases.

Case	Procs	Exec	Assign	Calls	Do	If	Reads	Writes
1	33	1069	530	76	52	246	6	13
2	30	913	446	84	37	203	6	10
3	33	1067	529	76	52	246	6	13
4	30	932	456	84	39	209	6	11
5	33	1049	519	77	51	241	6	12
6	37	1304	632	110	62	288	11	22
7	30	928	455	84	39	208	6	10
8	36	1228	622	101	61	278	6	14
9	30	928	455	84	39	208	6	10
10	44	1677	821	161	71	368	11	21
11	46	1786	855	216	76	375	9	16
12	38	1285	640	102	58	285	9	20
13	40	1448	691	166	57	324	7	12
14	45	1675	819	169	70	367	9	20
15	45	1959	957	209	85	414	13	25
16	45	1784	861	177	73	383	12	24
17	45	1728	840	171	71	379	12	24
Union	51	2408	1187	286	108	490	14	30
Intersect	29	778	389	42	35	186	6	10
Maximum	68	4300	1870	418	157	753	34	206

Please note that we did not measure the structural coverage of either system or unit tests. Statements which were not exercised during acceptance test might have been exercised during previous testing. Structural coverage measures were not available during either system or unit test. Procedures were not tested with the goal of achieving high structural coverage.

Question:

What is the statement coverage of operational use?
What is the procedure coverage of operational use?

We obtained 60 samples of actual operational inputs that we claim are representative of SEL operational usage. This is significantly different from other definitions of operational usage where the input domain is typically divided into subdomains, with each subdomain being assigned a probability of execution. Input cases are then chosen using the probabilities of execution [10, 11, and 6]. Our definition of operational usage lacks both the definition of subdomains and the assignment of probabilities. These probabilities are difficult to compute and verify. Rigorously derived or otherwise, these operational usage cases define how the program was exercised.

Question:

What is the intersection / union of functional testing and operational use?

Table 2 compares the structural coverage of functionally generated acceptance tests and operational usage. Together they exercised 55 procedures and and 2768 executable statements. Their intersection (the statements exercised by both sets of inputs) contains 51 procedures and 2397 executable statements. There are 360 executable statements that are exercised by operational usage but not by acceptance test; There are 11 executable statements that are exercised by

acceptance test but not by operational usage.

Comparison of Statement Coverage of the MAL Preprocessor by 17 Acceptance Test Cases and 60 Operational Usage Cases.

Case	Procs	Exec	Assign	Calls	Do	If	Reads	Writes
Acpt	51	2408	1187	286	108	490	14	30
Usage	55	2757	1345	327	120	581	19	36
Union	55	2788	1353	327	120	581	19	36
Intersect	51	2397	1179	286	108	490	14	30
A-OpU	0	11	8	0	0	0	0	0
OpU-A	4	360	166	41	12	91	5	6

Comparison of Statement Coverage of the MAL Preprocessor by 17 Acceptance Test Cases and 60 Operational Usage Cases. (by percentage of Maximum)

Case	Procs	Exec	Assign	Calls	Do	If	Reads	Writes
Acpt	75.0	56.0	63.5	68.4	68.8	65.1	41.2	14.6
Usage	80.9	64.1	71.9	78.2	76.4	77.2	55.9	17.5
Union	80.9	64.4	72.4	78.2	76.4	77.2	55.9	17.5
Intersect	75.0	55.7	63.0	68.4	68.8	65.1	41.2	14.6
A-OpU	0.0	0.3	0.4	0.0	0.0	0.0	0.0	0.0
OpU-A	5.9	8.4	8.9	9.8	7.6	12.1	14.7	2.9

Some interesting observations can be made. The I/O statements, especially the WRITE statements, are less likely to be executed than most other statement subclasses. This is reasonable considering the role WRITE statements play in debugging and error condition handling code. Also, as statement coverage increases, different statement subclasses are more likely to be exercised. In table 3 the line labeled "OpU-A" describes the statements that are executed in operational use but not in acceptance test. Operational usage exercised 8.4% of the code that acceptance test never exercised. This 8.4% is not an even cross section of the statement subclasses. One would reasonably expect the 8.4% to be similar for different statement subclasses but this is not so; as much as 12.1% of the IF statements and 14.7% of the READ statements are executed but only 2.9% of the WRITE statements are executed.

While this is an interesting result in its own right, this also has some significance to software reliability models. Assuming that statements from different statement subclasses have different likelihoods of being a "fault," then this result seems to imply that a representative reliability model should have a hazard function (see [9]) that varies over time.

Question:

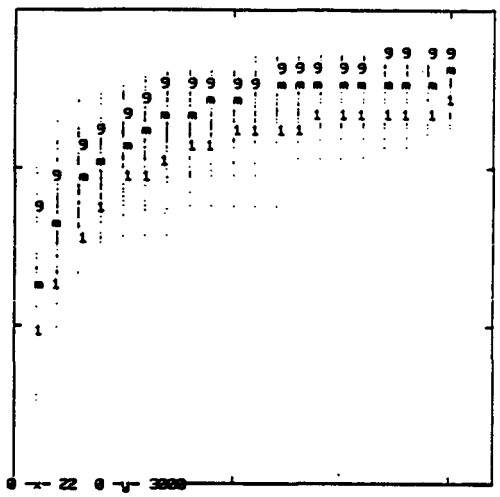
What are the properties of structural coverage growth?

For a set of input cases, structural coverage monotonically increases with the execution of each new input case (bound above by the number of reachable statements). This section examines the growth of structural coverage. It is important for two reasons:

(1) It provides a way to see if two sets of input cases exercise the program the same way. This provides a way to compare the equivalence of operational use and acceptance testing.

(2) It provides useful data for the reliability models. Assuming that increased coverage implies a higher failure rate, then anything we learn about the growth of structural coverage can be applied to the calculation of the reliability models' hazard functions.

With 17 acceptance tests and 60 operational usage cases, there are clearly too many sequences to exhaustively examine. In a personal communication, Dr. Amrit Goel proposed a solution: examine the structural coverage of a large, but manageable number of sequences. Plot 1 shows the structural coverage growth of 100 permutations of operational usage with median, 10th and 90th percentiles superimposed.



Plot 1. Structural Coverage of 100 Permutations of 60 Operational Usage Cases. (median, 10th, and 90th percentiles superimposed)

A variety of models were fitted to the structural coverage growth data in an attempt to learn more about structural coverage growth. A good mathematical model of structural coverage growth would provide insight into structural growth. Models were fitted to the first half of a sequence to evaluate their usefulness as predictors and to the entire sequence to evaluate their ability to characterize structural coverage growth. Plots of the residuals were examined visually to estimate goodness of fit.

The best fit was obtained using Goel and Okumoto's NHPP model [12]. The NHPP model was originally defined as a reliability model. Given a history of faults revealed over time, it can be used to estimate the number of faults to be revealed by time t. It is being used here as a model of structural coverage growth. Restated in terms of structural coverage growth, the model is:

$$m(t) = a(1-e^{-bt})$$

where

$m(t)$ is the expected value of the number of statements executed by test t.

a predicts the expected number of statements to be executed.

b determines the steepness of the curve.

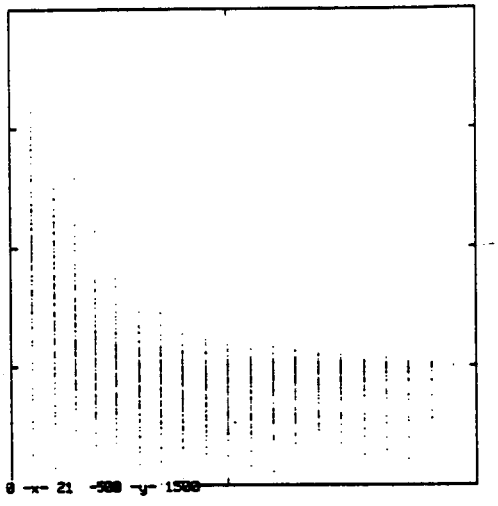
Given $m(1)$ through $m(t_{max})$, a and b can be calculated. Note

the following property:

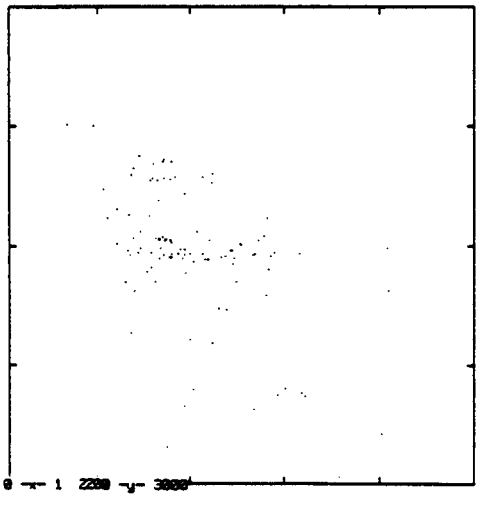
$$\lim_{t \to \infty} m(t) = a$$

$$= \textit{expected statement coverage}$$

It is the best of the models attempted, but its results are imperfect even when a variety of data transformations are applied. Plots 2-3 show one of the fitted models and its residual. This remains an area of future research.



Plot 2. NHPP Model Fitted to the First 20 Values of the 100 Operational Growth Sequences. (residuals)



Plot 3. NHPP Model Fitted to the First 20 Values of the 100 Operational Growth Sequences. (plot of a vs b)

In summary, we have used structural coverage to provide insight into how functional acceptance test and operational usage exercise a program's code; to suggest results that effect reliability models; to suggest a relationship between procedure coverage and statement coverage; and to move toward understanding statement coverage growth.

Comparison of Inputs Using Structural Coverage Metrics

Does functional testing have the same coverage profile as operational usage, or more generally, can structural coverage be used to compare two sets of program inputs? This question is interesting for two reasons:

(1) Some testing models require input sets that are "representative" of operational usage [10]. Structural coverage could provide a way of measuring this.

(2) Many reliability models, when using past failure data to predict failure rate or number of failures, assume that the past inputs are similar to the present inputs. Structural coverage could provide a method for confirming this.

Question:

Can the Venn diagram technique be used to differentiate input sets?

In an earlier section we compared functional test sets with operational usage using a Venn diagram technique (tables 2 and 3). We used this to compare how functional tests and operational usage exercised the program. Could this be extended to other input sets? For example, it seems plausible that tests generated with the goal of high branch coverage would execute different code than tests generated by test mutation on arithmetic expressions [13] or that boundary value functional tests would exercise different sections of code than statistical predictions of operational usage. We hypothesize that the code in the different sections of the Venn diagram would reflect the properties of the two sets of tests.

Question:

Can input sets be differentiated using nonparametric tests of structural coverage?

Acceptance test and operational usage were statistically compared using both the Mann-Whitney and Kruskal-Wallis tests*. The proposed hypotheses were: "For each of the structural coverage classes (procedures, executable statements, assignment statements...) the population represented by the 60 operational usage cases is similar to the population represented by the acceptance test cases."

The tests fail to reject the hypotheses for all statement types except READs. Since there are so few READ statements, a small, random difference in the tests could falsely manipulate the statistic. The other statement classes are less susceptible to small changes and represent a better population to examine.

The tests fail to reject the hypotheses that the two populations are similar, meaning that in this case, operational use and acceptance test cannot be distinguished by their structural coverage numbers.

Question:

Can the number of executions of prime sections of code be used to differentiate input sets?

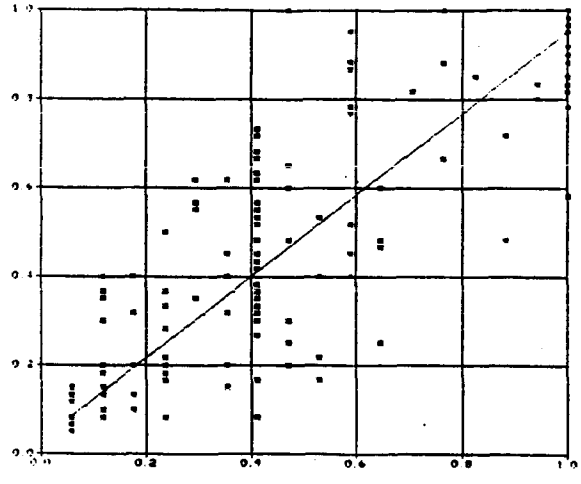
Are statements executed as thoroughly by acceptance test as they are by operational usage? For each statement in the

* The Mann-Whitney and Kruskal-Wallis tests were chosen because they are nonparametric tests; they make no assumptions about the distributions of source populations. The Mann-Whitney test is most sensitive to differences in "location (central tendency)." The Kruskal-Wallis test is sensitive to differences in "location or disper-

program, it is possible to count how many times it was exercised by a particular acceptance test or operational usage case. (In this paper we will distinguish between *exercise* and *execute*. A statement can be *executed* many thousands of times by a single acceptance test. Each acceptance test or operational usage case *exercises* the statement once). If acceptance test and operational usage are similar, then the percentage of acceptance test cases that executed a statement should be similar to the percentage of operational usage cases.

The two percentages were calculated for each prime section* of code. The plotted data are shown in scatter plot 4. The regression line has slope 0.921 and intercept 0.032. The r square value is 0.863.

Since the plot does not show any imbalance, one could conclude that acceptance test and operational usage exercise the code equally thoroughly. It is a future goal of this research to replace this empirical judgement by a statistical test.



Plot 4. Comparison of Execution Coverage of Acceptance Test and Operational Usage. (% Acceptance Test on X-axis. % Operational Usage on Y-axis.)

To summarize, we proposed three methods for comparing sets of program inputs: Venn diagram comparison of executed statements, statistical comparison, and thoroughness of execution of prime sections code. These methods may be able to differentiate input sets, a result that would be useful for understanding reliability models and some testing strategies.

Error, Faults, and Failures and Structural Coverage

The SEL has been collecting data on software development for eight years [1]. Error, fault and failure data are collected using the "Change Report Form" or CRF. A CRF is filed whenever a change, enhancement or fault repair is made to a subroutine or data file. This study examines three fields of the form, "time to isolate the error," "the time to understand and implement," and the section "type of error*."

There were eight faults found during operation. Each fault could be repaired by changing code in one procedure. One procedure contained two faults. With these data, we can address these questions:

Question:

Were heavily exercised sections of code more likely to contain faults?

Half of the procedures were exercised by more than 90% of the operational usage cases. About half of the revealed faults occurred in this group of procedures (3 of 8). With these data we reject the hypothesis that more heavily exercised subroutines are more likely to contain a revealed fault.

Question:

Is procedure coverage related to time to isolate?

Time to isolate the change seems to be independent of procedure coverage.

Question:

Is procedure coverage related to time to understand and implement?

Increased usage seems to be associated with longer time to understand and implement a change. This might be explained by suggesting that the lightly exercised procedures contain fairly simple code while the heavily exercised code is, by necessity, more complicated and requires more time to modify.

Question:

Is procedure coverage related to type of error?

There are too few faults to reveal any interesting patterns.

In summary, we have tried to relate procedure coverage to: "time to isolate an error," "time to understand an error," and "type of error." The data begins to suggest a relationship between "time to understand an error" and structural coverage. There were too few errors to make any firm statements about "time to isolate an error" and "type of error." This remains a promising area of study.

Structural Coverage and the Management of Acceptance Tests

Combined with failure data, structural coverage could aid the design of acceptance tests. Imagine a manager in charge of designing acceptance tests for a group of similar projects or for various releases of a single project. With the failure data from the previous project or release and the structural coverage of both the acceptance and operational usage cases he can suggest new acceptance tests for the next release. He could require tests to exercise previously unexercised sections of code. He could require new acceptance tests to explain the code missed by acceptance test but exercised in operational usage. If he is using a testing methodology or reliability model that requires inputs that are representative of operational usage, he

* Prime sections of code are sequences of executable statements that contain no statements that alter the flow of control. Thus if control reaches the first statement of a prime section, all the statements will be executed (barring run-time errors or interrupts).

* Time to isolate the error is classified as taking: less than one hour, one hour to one day, greater than one day, never found. Time to understand and implement the change is classified as taking: less than one hour, one hour to one day, one day to three days, or greater than three days. Faults are categorized as originating in the: requirements, functional specification, design (either involving data or ex-

can use these data to select more representative tests.

We see structural coverage being used by a manager in an iterative fashion:

(1) Gather structural coverage data on acceptance tests and release the project.

(2) Gather structural coverage data and failure data on operational usage. Use these data to adjust reliability models.

(3) Use structural coverage data to: suggest new tests and evaluate how the old tests were created.

(4) Restart the cycle with the new acceptance tests.

Conclusions and Criticisms

We conclude:

(1) We may be able to compare sets of inputs using statistical tests and Venn diagram techniques. This would be useful for examining some testing methods and reliability models.

(2) The structural coverage growth of different statement subclasses grows at different rates. This insight might be of interest to reliability model developers.

The data seem to imply:

(1) Faults are independent of the number of executions. We can (in our environment) reject the hypothesis that heavily exercised procedures are more likely to contain more revealed faults.

(2) Procedure coverage may be used instead of statement coverage.

(3) Structural coverage metrics can be used to aid in the management of the acceptance test process.

This study can be criticized on a number of points:

(1) There are too few faults to make any forceful statements about errors, faults, failures and structural coverage. (But then again we cannot fault NASA/GSFC for having programs with too few faults.)

(2) While the data suggests that it may be possible to differentiate test sets using structural coverage, we have never provided an example that shows that it can!

(3) Because the data was unavailable, this study does not address the order in which the functional tests were used, the order of the operational usage cases or which operational usage cases revealed the faults.

(4) The study did not produce a good model of structural coverage growth.

These points will be addressed when the study is replicated in the summer of 1985. The program being studied is DERBY [15], a large (300 routines, ~50k source lines of code), satellite simulator. The new project is larger and should have more faults. With the new project, we will gather more thorough information on the order of system tests, acceptance tests, and operational usage cases, plus the exact input that reveals a failure. The results of this new study should answer many of the questions raised by this study.

Acknowledgments

We would like to thank Frank McGarry, NASA/Goddard Space Flight Center, Dr. Gerald Page, Computer Sciences Corporation, and Dr. Amrit Goel, of Syracuse University, for their help in the production of this paper, Dr. David Hutchens, of Clemson University, for a clear-eyed review, and the University of Maryland's Software Engineering group for providing a fertile intellectual environment. We would also like to thank the reviewers for providing many useful comments.

References

SEL reports can be obtained from: Frank McGarry, Code 582.1, Room E231, Building 23, NASA/Goddard Space Flight Center, Greenbelt, Maryland, 20771, USA. University of Maryland Technical Reports are available from the authors.

[1] The Software Engineering Laboratory, SEL-81-104, Software Engineering Laboratory Series, February 1982.

[2] Acceptance Test Methods, TM-78/6296, Computer Sciences Corporation, October 1978.

[3] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 729-1983, IEEE Inc., February 1983.

[4] Guide to Data Collection, SEL-81-101, Software Engineering Laboratory Series, August 1982.

[5] Joe W. Duran and John J. Wiorkowski, Quantifying software validity by sampling, *IEEE Transactions on Reliability* **R-29**, 2, pp. 141-144, June 1980.

[6] M. Dyer and Harlan D. Mills, Developing electronic systems with certifiable reliability, *Proceedings of the Conference on Electronic Systems Effectiveness and Life Cycle Costing*, NATO Advanced Study Series, Springer-Verlag, Summer 1982.

[7] W. D. Brooks and R. W. Motley, Analysis of Discrete Software Reliability Models, RADC TR 80-84, RADC, April 1980.

[8] William E. Howden, A Survey of Dynamic Analysis Methods, *Tutorial: Software Testing & Validation Techniques, 2nd Ed.*, ed. E. Miller and W. E. Howden, pp. 209-231, 1981.

[9] G. J. Myers, *The Art of Software Testing*, John Wiley & Sons, New York, 1979.

[10] J. R. Brown and M. Lipow, Testing for software reliability, *Proceedings of the International Conference on Reliable Software*, Los Angeles, CA, pp. 518-527, April 1975.

[11] Joe W. Duran and John J. Wiorkowski, Towards models for probabilistic program correctness, *Proceedings of the ACM Software Quality Assurance Workshop*, pp. 39-44, 1978.

[12] Amrit L. Goel and K. Okumoto, A Time Dependent Error Detection Rate Model for Software Performance Assessment with Applications, annual report to RADC, Department of Industrial Engineering and Operations Research, Syracuse University, Syracuse, New York, March 1980.

[13] Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward, Hints on test data selection: Help for the practicing programmer, *Computer*, pp. 34-41, April 1978.

[14] Sidney Siegel, *Nonparametric Statistics for the Behavioral Sciences*, McGraw-Hill Book Company, Inc., New York, 1956.

[15] ERBS Dynamics Simulator User's Guide and System Description, SD-83/6044, Computer Sciences Corporation, August 1983.

[16] Victor R. Basili and David M. Weiss, A Methodology for Collecting Valid Software Engineering Data, TR-1235, Computer Science Technical Report Series, December 1982.

[17] Joe W. Duran and Simon Ntafos, A report on random testing, *Proceedings of the Fifth International Conference on Software Engineering*, pp. 179-183, March 1981.

[18] Amrit L. Goel, Software error detection model with applications, *Journal of Systems and Software* 1, 3, pp. 243-249, 1980.

[19] John B. Goodenough and Susan L. Gerhart, Toward a theory of test data selection, *IEEE Transactions on Software Engineering*, pp. 156-173, June 1975.

[20] John D. Musa, Software reliability measurement, *Journal of Systems and Software* 1, 3, pp. 223-241, 1980.