Computer Science
Technical Report

Software Test Coverage and Reliability

Yashwant K. Malaiya*
Naixin Li
James M. Bieman†
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
malaiya@cs.colostate.edu

Rick Karcich
Bob Skibbe
StorageTek
2270 South 88th Street
Louisville, CO 80028-2286
(303) 673-6223
Rick_Karcich@stortek.com

Technical Report CS-96-128

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

*Y. Malaiya and N. Li are partly supported by a BMDO funded project monitored by ONR

†J. Bieman is supported, in part, by the NASA Langley Research Center, the Colorado Advanced Software Institute (CASI) and Storage Technology Inc. CASI is supported in part by the Colorado Advanced Technology Institute (CATI). CATI promotes advanced technology teaching and research at universities in Colorado for the purpose of economic development.

Software Test Coverage and Reliability

Yashwant K. Malaiya*
Naixin Li
James M. Bieman†
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
malaiya@cs.colostate.edu

Rick Karcich
Bob Skibbe
StorageTek
2270 South 88th Street
Louisville, CO 80028-2286
(303) 673-6223
Rick_Karcich@stortek.com

Abstract

Tools are now available that measure test coverage in terms of blocks, branches, c-uses, p-uses, etc. covered. Here we consider the question of how a coverage measure is related to the number of faults discovered. We model the relation among test effort, coverage and reliability, and present a logarithmic model that relates test effort to test coverage. The model is based on the hypothesis that the enumerables (like branches or blocks) for any coverage measure have different detectability, just like the individual defects. This model allows us to relate a test coverage measure directly with defect coverage. Data sets for programs with real defects are used to validate the model. The results are consistent with the known inclusion relationships among block, branch and p-use coverage measures. We show how defect density controls time to next failure.

The model can eliminate the variables like test application strategy from consideration. It is suitable for high reliability applications where automatic (or manual) test generation is used to cover enumerables which have not yet been tested.

Index Terms — Software testing, test coverage, operational profile, detectability profile, software reliability modeling.

1 Introduction

Developers can achieve the target reliability of software systems in a predictable way by evaluating reliability during development. By evaluating and projecting reliability growth, the developers can optimally allocate resources to meet a deadline with the target reliability [20].

Reliability can be estimated as soon as running code exists. To quantify reliability during testing, the code (or portion of code) is executed using inputs randomly selected following some distribution.

*Y. Malaiya and N. Li are partly supported by a BMDO funded project monitored by ONR

†J. Bieman is supported, in part, by the NASA Langley Research Center, the Colorado Advanced Software Institute (CASI) and Storage Technology Inc. CASI is supported in part by the Colorado Advanced Technology Institute (CATI). CATI promotes advanced technology teaching and research at universities in Colorado for the purpose of economic development.

Then, a reliability growth model can be used to predict the amount of effort required to satisfy product reliability requirements.

The needs of early reliability measurement and modeling are not met by common testing practices. The focus of testing is on finding defects, and defects can be often found much faster by non-random methods [3]. Testing is directed towards inputs and program components where errors are more likely. For example, testing may be conducted to ensure that particular portions of the program and/or boundary cases are covered.

Models that can measure and predict reliability based on the status of non-random testing are clearly needed. Reliability achieved will be affected by:

- the testing strategy: Test coverage may be based on the functional specification (black-box), or it may be based on internal program structure (white-box). Strategies can vary in their ability to find defects.
- the relationship between calendar time and execution time: The testing process can be accelerated through the possibly parallel, intensive execution of tests at a faster rate than would occur during operational use.
- the testing of rarely executed modules: Such modules include exception handling or error recovery routines. These modules rarely run [12], and are notoriously difficult to test. Yet, they are critical components of a system that must be highly reliable. Only by forcing the coverage of such critical components, can reliability be predicted at very high levels.

Intuition and empirical evidence suggests that test coverage must be related to reliability. Yet, the connection between structure based measurements, like test coverage, and reliability is still not well understood.

There are several motivations for investigating the relation between test coverage and reliability. Test coverage, rather than test effort is a direct measure than of how thoroughly a system has been exercised. With the same test effort (measured in CPU execution time or calendar time), a less effective test strategy may be less efficient in finding defects. Measuring test-coverage is usually an intrusive approach, however the available tools now allow it to be done automatically.

Ramsey and Basili [25] investigate different permutations of the same test set and collected data relating the number of tests to statement coverage growth. They try a variety of models to fit the data. The best fit is obtained using the Goel and Okumoto's exponential model.

Dalal, Horgan and Ketterring [7] examine the correlation between test coverage and the error removal rate. They give a scatter plot of the number of faults detected during system testing versus the block coverage achieved during unit testing. The plot shows that modules covered more thoroughly during unit testing are much less likely to contain errors.

Vouk [30] finds that the relation between structural coverage and fault coverage is a variant of the Rayleigh distribution. He assumes that the fault detection rate during testing is proportional to the number of faults present in the software and test coverage values including block, branch, data-flow, and functional group coverage. Vouk's experimental results suggest the use of a more general Weibull distribution. He observes that, in terms of error removal capability, the relative power of the coverage measures {block:p-use:DUD-chains} is {1:2:6}.

Chen et al [5] add structural coverage to traditional time-based software reliability models (SRMs). Their model excludes test cases that do not increase coverage. The adjusted test effort data is used to fit existing time-based models to avoid the overestimation of traditional time-based SRMs due to the saturation effect of testing strategies.

Assuming random testing, Piwowarski, Ohba and Caruso [23] analyze block coverage growth during function test, and derive an exponential model relating the number of tests to block coverage. Their model is equivalent to the Goel-Okumoto model [25]. They also derive an exponential model relating the covering frequency to the error removal ratio. The model relies on prior knowledge of the error distribution over different functional groups in a product.

Frankl and Weiss [8] compare the fault exposing capability of branch coverage and data flow coverage criteria. They find that for 4 out of 7 programs, the effectiveness of a test in exposing an error is positively correlated with the two coverage measures. They observed complex relationships between test coverage growth and the probability of exposing an error for a test set. Since the seven programs they used are very small and they only considered subtle errors, the result may not be directly applicable to large commercial software systems.

Hutchins, et al [13] study detection effectiveness of test sets with different coverage values for realistic seeded faults. They find that a test set with higher coverage has higher per-test detection probability. They also showed that 100% coverage using a specific measure may not detect all the faults.

The Leone test coverage model given in [21] is a weighted average of four different coverage metrics achieved during test phases: lines of executable code, independent test paths, functions/requirements, and hazard test cases. The weighted average is used as an indicator of software reliability. The model assumes that full coverage of all four metrics implies that the software tested is 100% reliable. In reality, such software may have some remaining faults. A similar approach, but with different coverage metrics, was taken to provide a test quality report [24].

In this paper, we explore the connection between test coverage and reliability. We develop a model that relates test coverage to defect coverage. With this model we can estimate the defect density. With knowledge of the operational profile, we can predict reliability from test coverage measures.

2 Coverage of Enumerables

The concept of test coverage is applicable for both hardware and software. In hardware, coverage is measured in terms of the number of possible faults covered. For example, each node in a digital system can possibly be stuck-at 0 or stuck-at 1. A stuck-at test coverage of 80% means that the tests applied would have detected any one of the 80% faults covered.

In contrast, the number of all possible software faults in not known. Test coverage in software is measured in terms of structural or data-flow units that have been exercised. These units can be statements (or blocks), branches, etc. as defined below:

- Statement (or block) coverage: the fraction of the total number of statements (blocks) that have been executed by the test data.
- Branch (or decision) coverage: the fraction of the total number of branches that have been executed by the test data.
- C-use coverage: the fraction of the total number of computation use (c-uses) that have been

covered by one c-use path during testing. A c-use path is a path through a program from each point where the value of a variable is modified to each c-use (without the variable being modified along the path) [26].

- P-use coverage: the fraction of the total number of p-uses that have been covered by one p-use path during testing. A p-use path is a path from each point where the value of a variable is modified to each p-use, a use in a predicate or decision (without modifications to the variable along the path) [26].

Test coverage can also be measured in other units, such as context, required k-tuple, etc. A formal evaluation of different test coverage criteria was presented by Clarke [6]. When such a unit is exercised, it is possible that one or more associated faults may be detected. Counting the number of units covered gives us a measure of the extent of sampling. The defect coverage in software can be defined in an analogous manner; it is the fraction of actual defects initially present that would be detected by a given test set.

To keep the following discussion general, we will use the term *enumerable*. For branch coverage, the enumerables are branches, for defect coverage the enumerables are defects and so on. We use the term "enumerable-type" to imply defects, blocks, branches, c-uses or p-uses. We use superscript i, $i = 0$ to 4, to indicate one of the five types in this sequence: 0: defects, 1: blocks, 2: branches, 3: c-uses, 4: p-uses.

In general, test coverage increases when more tests are applied, provided that the test cases are not repeated and complete test coverage has not already been achieved. A small number of enumerables may not be reachable in practice. We assume that the fraction of such enumerables is negligible.

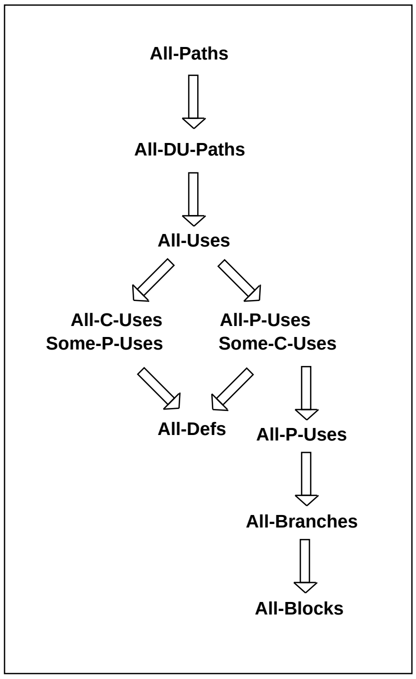

Figure 1: The subsumption relationships of different complete coverage criteria [26]

Figure 1 [26] shows the relationship among some well-known criteria as proven by Rapps and

Table 1: The complexity (test length) for achieving different coverage criteria [32]

Coverage Criterion	Upper bound
All-Blocks	$d+1$
All-Branches	$d+1$
All-P-Uses	$\frac{1}{4}(d^2+4d+3)$
All-Defs	$m+(i \times n)$
All-P-Uses/Some-C-Uses	$\frac{1}{4}(d^2+4d+3)$
All-C-Uses/Some-P-Uses	$\frac{1}{4}(d^2+4d+3)$
All-Uses	$\frac{1}{4}(d^2+4d+3)$
All-DU-Paths	2^d
All-Paths	∞

n: variables
m: assignments
i: input statements
d: binary decisions.

Weyuker. If there is a directed path from criteria A to criteria B, then test sets that meet criteria A (achievement of complete coverage) are guaranteed to satisfy criteria B. Table 1 shows the maximum number of tests [32] to satisfy these criteria. The upper bound for all-du-paths was reached in one subroutine out of 143 considered by Bieman and Schultz [4].

3 Detectability Profiles of Enumerables

The coverage achieved by a set of tests depends not only on the number and types of tests applied (or, equivalently, the testing time) but also on the distribution of *testability* values of the enumerables. A statement which is reached more easily is more testable. Such statements are likely to get covered (i.e. exercised at lease once) with only a small number of tests. Testability also depends on the likelihood that a fault that is reached actually causes a failure [29]. A statement that is executed only in rare situations has low testability. It may not get exercised by most of the tests that are normally applied. As testing progresses, the distribution of testability values will shift. The easy-to-test enumerables are covered early during testing, and are removed from consideration. The remaining enumerables include a larger fraction of hard-to-test enumerables. Thus, the growth of coverage will be slower.

Definition: Detectability of an enumerable d_l^j is the probability that the l-th enumerable of type j will be exercised by a randomly chosen test.

The *detectability profile* is the distribution of detectability values in the system under test. The detectability profiles were introduced by Malaiya and Yang [19]. They have been used to characterize testing of hardware [31] and software [18]. A continuous version of the detectability profile was defined by Seth, Agrawal and Farhat [27]. For convenience, we use the normalized detectability profile (NDP) as defined below.

Definition: The discrete NDP for the system under test is given by the vector,

$$P^j = \{p_{d1}^j, p_{d2}^j, ..., p_{di}^j, ..., p_{du}^j\}$$

$$d_{i-1}^j < d_i^j < d_{i+1}^j \tag{1}$$

where p_{di}^j is the fraction of all enumerables of type j which have detectability equal to di. Thus $p_{0.3}^2$ represents the fraction of all branches with detectability of 0.3. In Equation 1, du is the detectability value of unity (1), the highest value possible. $\sum_{di=0}^{1} p_{di}^j = 1$ since all fractions added will be unity.

A detectability value of 0 is possible, since a branch might be infeasible, or a defect might not be testable due to redundancy in implementation. Researchers have compiled detectability profiles of several digital circuits [19, 31] and software systems [28].

If the number of enumerables is large, a continuous function can approximate the discrete NDP.

Definition: The continuous NDP, for the system under test is the function $p^j(x)$, $0 \leq x \leq 1$

$$p^j(x)dx = \frac{nr\_enumerables^j(x, x+dx)}{all\_enumerables^j} \tag{2}$$

where $nr\_enumerables^j(x, x+dx)$ denotes all enumerables of type j with detectability values between x and $x + dx$.

Note that $\int_0^1 p^j(x)dx = 1$, just like the discrete NDP case.

4 Coverage of Enumerables

The detectability profile gives the probability of exercising each enumerable. Hence, it can be used to calculate expected coverage when a given number of tests have been applied. Here, we assume that testing is random, i.e. any single test is selected randomly with replacement. Malaiya and Yang [19], and Wagner et al [31] show that the expected coverage of the enumerables of type j is

$$C^j(n) = 1 - \sum_{i=1}^{n}(1 - d_i^j)^n p_i^j \tag{3}$$

provided testing is random. The same result holds for continuous NDP [27]

$$C^j(n) = 1 - \int_0^1 (1-x)^n p(x)dx \tag{4}$$

Actual testing of software is more likely to be pseudo-random, since a test once applied will not be repeated. In such cases, random testing is an approximation. This approximation is reasonable, except when coverage approaches 100%.

Equations 3 and 4 give expected coverage. In a specific case, the coverage can be different. The central limit theorem suggests that results obtained should be close to these given by Equations 3 and 4 when a large number of vectors are applied.

The use of Equations 3 and 4 requires knowledge of detectability profiles. Obtaining exact detectability profiles requires a lot of computation since the detectability of each defect needs to be computed. Discrete detectability profiles have been calculated for several small and large combinational circuits. Continuous detectability profiles for some benchmark circuits have been estimated [27]. Similar data for some software systems is also available [1].

Fortunately, it is possible to obtain an approximation for the detectability profiles. When one test is applied, the probability that an enumerable with detectability d_i^j will not be covered is $(1-d_i^j)$. The probability that an enumerable will not be covered by n tests and thus remain a part of profile is $(1 - d_i^j)^n$. Thus if the initial profile was given by Equation 1, the profile after having applied n tests, will be given by

$$P_n^j = \{p_{d1}^j(1 - d1^j)^n, p_{d2}^j(1 - d2^j)^n, ...\}$$

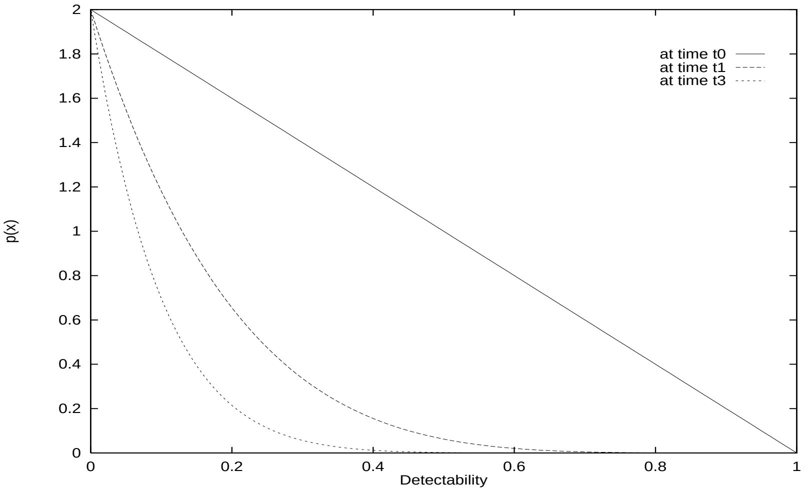

Figure 2: Distribution of detectability

Equivalently the continuous profile is given by

$$p_n^j(x) = p_n(x)(1-x)^n$$

Thus enumerables with high testability are likely to get covered earlier. The profile will "erode" as testing progresses (see Figure 2). Enumerables with low testability get removed at a much lower rate, and thus will soon dominate. During much of the testing, the shape of the profile will appear like the bottom curve in Figure 2, regardless of the initial profile. Trachtecnberg's justification of the Zipf's law supports a similar shape [28].

Available results for hardware components suggest that initial detectability profiles may be of the form

$$p^j(x) = (mj+1)(1-x)^{mj} \tag{5}$$

where mj is a parameter. The factors $(mj+1)$ ensure that the area under the initial profile curve is unity. By substituting the right hand side of Equation 5 in Equation 4, we get

$$\begin{aligned} C^j(n) &= 1-(mj+1)\int_0^1 (1-x)^{mj+n}dx \\ &= 1-\frac{mj+1}{mj+n+1} = \frac{n}{mj+n+1} \end{aligned} \tag{6}$$

The curve given by Equation 6 matches the shape of experimental data. However it does not provide a good fit. One problem is that Equation 6 includes only a single parameter which can be adjusted for fitting. We can assume a more general initial detectability in Equation 5, involving two parameters, but even that may not be accurate due to non-randomness of testing, as we discuss in the next section. The approach considered next, yields a much better model.

5 A New Logarithmic Coverage Model

Random testing implies that a new test is selected regardless of the tests that have been applied thus far, and that tests are selected based only on the operational distribution. In actual practice, a test case is selected in order to exercise a functionality or enumerable that has remained untested so far. This process makes actual testing more directed and hence more efficient than random testing.

Malaiya, von Mayrhauser and Srimani [18] show that this non-random process leads to a defect finding behavior described by the logarithmic growth model [20]. Their analysis gives an interpretation for the model parameters. The coverage growth of an enumerable-type depends on the detectability profile of the type and the test selection strategy. We hypothesize that the defect coverage growth follows the logarithmic model:

$$C^0(t) = \frac{1}{N^0}\beta_0^0 \ln(1+\beta_1^0 t), \qquad C^0(t) \le 1 \tag{7}$$

where $C^0(t)$ is the defect coverage at time t and N^0 is the total number of initial defects. This hypothesis follows directly from the Logarithmic model, which has been shown to have superior predictive capability compared with other 2 parameter models [17]. For convenience, we can rewrite the above by using parameters b_0^0 and b_1^0 in place of β_0^0 and β_1^0,

$$C^0(t) = b_0^0 \ln(1+b_1^0 t), \qquad C^0(t) \le 1 \tag{8}$$

We also hypothesize that the coverage growth of enumerable i also follows the logarithmic model ($i = 1, 2, 3, 4$),

$$C^i(t) = b_0^i \ln(1+b_1^i t), \qquad C^i(t) \le 1 \tag{9}$$

Since the values of $c^i(t)$ are unchanged once they reach unity, when $C^1 = 100\%$, C^2 can be less than 100%. This thus takes into account the fact that 100% coverage of one enumerable does not mean exhaustive exercising of the system.

For defects ($i = 0$), the parameters β_0^0 and β_1^1 have the following interpretation [18].

$$\beta_0^0 = \frac{K^0(0)N^0(0)}{a^0 T_L} \tag{10}$$

and

$$\beta_1^0 = a^0 \tag{11}$$

where $K^0(0)$ is the exposure ratio at time $t = 0$, T_L is the linear execution time and a^0 is a parameter that describes the variation in the exposure ratio.

Equation 9 relates coverage C^i to the number of tests applied. We use it to obtain an expression giving defect coverage C^0 in terms of one of the coverage metrics C^i, $i = 1$ to 4. Using Equation 9, we solve for t,

$$t = \frac{1}{b_1^i}[exp(\frac{C^i}{b_0^i}) - 1], \qquad i = 1 \ \ to \ \ 4$$

Substituting for C^0, again using Equation 9,

$$C^0 = b_0^0 \ln[1 + \frac{b_1^0}{b_1^i}(exp(\frac{C^i}{b_0^i}) - 1)], \qquad i = 1 \ \ to \ \ 4$$

Defining $a_0^i = b_0^0$, $a_1^i = \frac{b_1^0}{b_1^i}$ and $a_2^i = \frac{1}{b_0^i}$, we can write the above using three parameters as,

$$C^0 = a_0^i \ln[1 + a_1^i(exp(a_2^i C^i) - 1)] \qquad i = 1 \ \ to \ \ 4 \tag{12}$$

Equation 12 gives a convenient three-parameter model for defect coverage in terms of a measurable test coverage metric. Equation 12 is applicable for only $C^0 \leq 1$. It is possible to approximate Equation 12 using a linear relation, but it would be accurate for only a small range, as seen later.

6 Analysis of Data

We evaluate the proposed model, as given by Equations 9 and 12, using five data sets listed in Table 2. The first data set, DS1, is from a multiple-version automatic airplane landing system [16]. It was

Table 2: Data Sets Used

DataSet	KLOC	#Tests	Defects	Tool
DS1(Lyu)[2]	30	21k	66	ATAC
DS2(Vouk)	5	1196	9	BCG[1]
DS3(Vouk)	5	796	9	BCG[1]
DS4(Vouk)	5	796	7	BCG[1]
DS5(StoTech)[3]	400	–	–	MFT[1]

1. internal tool
2. limited data points
3. evolving program

collected using the ATAC tool developed at Bellcore. The twelve versions have a total of 30,694 lines. The data used is for integration and acceptance test phases, where 66 defects were found. One additional defect was found during operational testing. The next three data sets, DS2, DS3, and DS4 are from a NASA supported project implementing sensor management in inertial navigation system [30]. The last data set DS5 is from an evolving software system containing a large number of modules. As an example, the data set DS3 is reproduced in Table 3.

Table 3: Coverage Data: DS3
NASA project: Sensor management in inertial management [Vouk]
(integration/acceptance test phase: 9 faults found with 796 tests)

Cumulative Faults	Number of Test Cases	%Coverage			
		blocks	branches	p-uses	c-uses
1	1	57.01	50.94	36.24	69.60
2	2	58.50	53.40	39.80	70.30
3	4	61.30	57.00	45.50	72.00
4	10	69.39	64.15	51.64	78.81
5	20	77.80	72.00	57.10	83.30
6	30	85.61	79.01	62.85	87.63
7	44	87.00	81.00	65.10	88.10
8	114	92.40	89.00	74.70	91.25
9	160	93.50	90.40	77.20	91.50
9	796	95.99	93.87	84.59	92.29

Table 4: Summary table for DS1
(total 21,000 tests applied)

	Blocks i=1	Decisions i=2	c-uses i=3	p-uses i=4	Defects i=0
Total enum.	6977	3524	8851	4910	67
Final cov.	91.8%	83.9%	91.7%	73.5%	98.4%
b_0^i	0.031	0.049	0.036	0.041	0.184
b_1^i	2E+8	1234	3.4E+6	2439	0.01
LSE	5.7E-4	3.5E-5	5.8E-4	8.1E-5	7.3E-7

Table 5: Summary table for DS2
(total 1196 tests applied)

	Blocks i=1	Branches i=2	c-uses i=3	p-uses i=4	Defects i=0
Final cov.	89%	84%	76%	61%	90%
b_0^i	0.032	0.060	0.034	0.039	0.166
b_1^i	2E+8	870	3E+7	2500	0.11
LSE	0.02	6.2E-4	3.5E-3	4.9E-3	0.025
a_0^i	1.31	0.46	0.23	0.29	
a_1^i	1.8E-3	4.6E-3	9.11E-7	5.2E-3	
a_2^i	6.95	3.84	23.12	13.46	
LSE	0.017	0.018	0.041	0.025	

The results for data set DS1 are summarized in Table 4. The first row gives the total number of enumerables for all versions. The second row gives the average coverage when 21,000 tests had been applied. The values of the estimated parameters b_0^i and b_1^i and the least square error are given in the rows below. The model given by Equation 9 fits the data well as indicated by the LSE values. The data shows that $C^1 > C^2 > C^4$, which is expected. Complete decision coverage implies complete block coverage, and complete p-uses coverage implies complete decision coverage [3, 9, 22]. The c-uses coverage has no such relation relative to the other metrics.

Table 5 summarizes the result for DS2. Nine faults were revealed by application of 1196 tests; we assume that one fault (i.e. 10%) is still undetected. In spite of the small number of faults, the model given in Equations 12 fits the data well.

Figure 3 shows actual and computed values for fault coverage. The computed values have been obtained using branch coverage and Equation 12. At 50% branch coverage the fault coverage is still quite low (about 10%), however with only 84% branch coverage, 90% fault coverage is obtained. The branch coverage shows saturation at about 84%. This supports the view that 80% branch coverage is often adequate [11].

Figure 4 compares the computed values of defect coverage against actual values from a different perspective. The computed values are from the number of tests (traditional reliability growth modeling), and test coverage measures C^1, C^2, C^3 and C^4 using Equation 12. The calculated values are all quite close, showing that coverage based modeling can replace time-based modeling.

Table 6 presents the results for DS3. The coverage growth of different enumerables are plotted in Figure 5. The curves for block coverage, branch coverage and p-use coverage have the same shapes and their values have the relationship: $C^1 > C^2 > C^4$. The c-uses coverage does not have the same relation with other three. With the logarithmic x-axis (number of tests), C^1, C^2, C^3, C^4 appear as straight lines. From Equation 9, if $b_1^i n \gg 1$ is true for $i = 1, 2, 3, 4$, then we write,

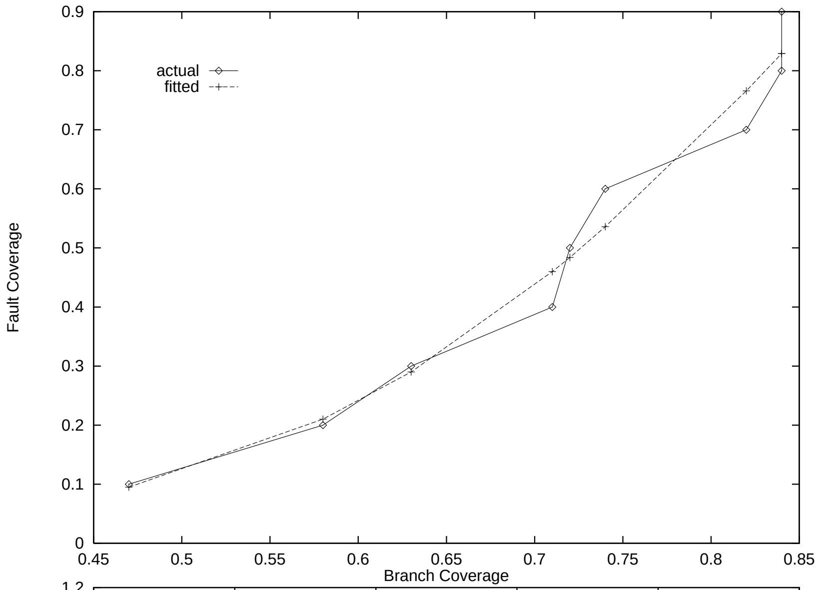

Figure 3: Actual and fitted values of defect coverage for DS2

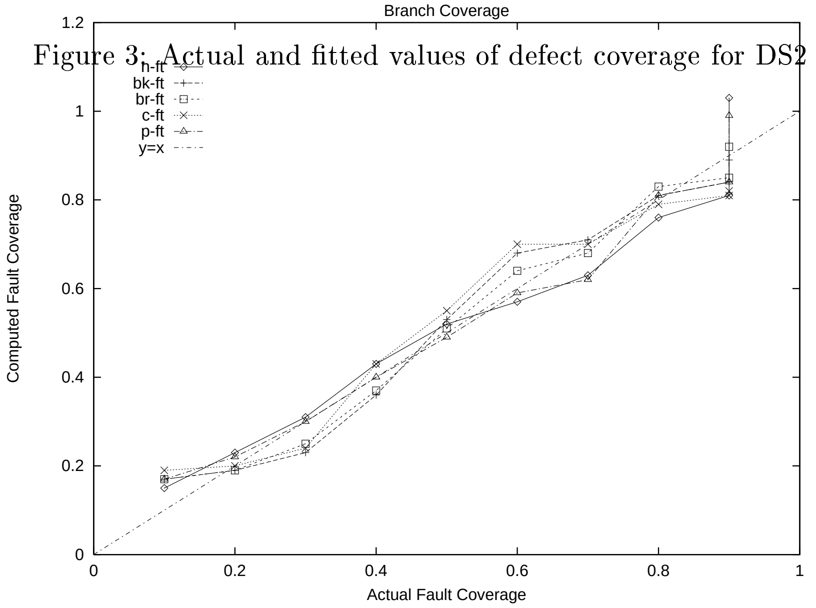

Figure 4: Actual defect coverage vs computed values for DS2

$$C^i(n) = b_0^i \ln(b_1^i n)$$

Table 6: Summary table for DS3(796 test cases)

	Blocks	Branches	C-uses	P-uses	Defects
Final Coverage	96%	94%	92%	85%	90%
b_0^i	0.07	0.074	0.044	0.079	0.139
b_1^i	2725	870	6.6E6	86	2.03
LSE	0.015	0.01	0.008	0.002	0.038
a_0^i	0.139	0.139	0.14	0.189	
a_1^i	7E-4	2.4E-3	9E-7	0.042	
a_2^i	14.13	13.14	21.46	9.88	
LSE	0.023	0.014	0.04	0.023	

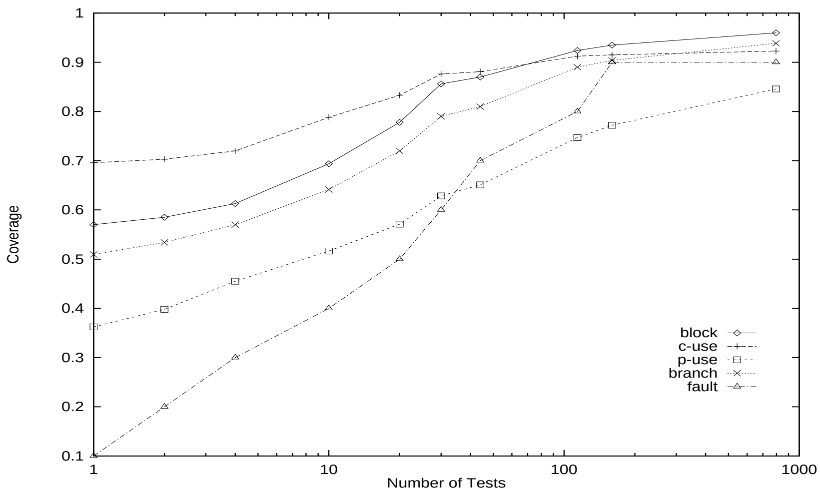

Figure 5: Coverage Growth of Different Enumerables (DS3)

or

$$C^i(n) = b_0^i \ln(b_1^i) + b_0^i \ln(n)$$

giving the linear curves of Figure 5. Since $b_1^0 < 1$, the above approximation is not applicable for $C^0(n)$ as observed in Figure 5.

Figure 6 plots defect coverage and relative defect density against branch coverage for DS3. For

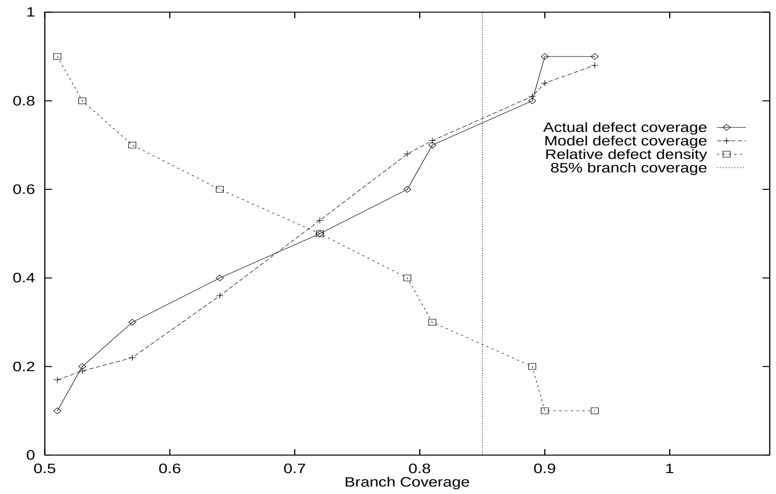

Figure 6: Fault Coverage & Relative Defect Density (DS3)

$C^i > 0.5$, a linear approximation can be made:

$$C^0 = a_0^i \ln(a_1^i \exp(a_2^i c^i)) = -A + BC^i \qquad C^i > 0.5$$

Figure 7 plots the relationship of defect coverage versus test coverage, as given by Equation 12. The overall curve is nonlinear, although the initial segment may not be observed in small programs because even a single test execution may provide close to 50% enumerable coverage.

Table 7 summarizes the result for DS4. Figure 8 shows the correlation of other test coverage measures C^2, C^3 and C^4 with block coverage C^1. As we expect, branch coverage, and to a lesser

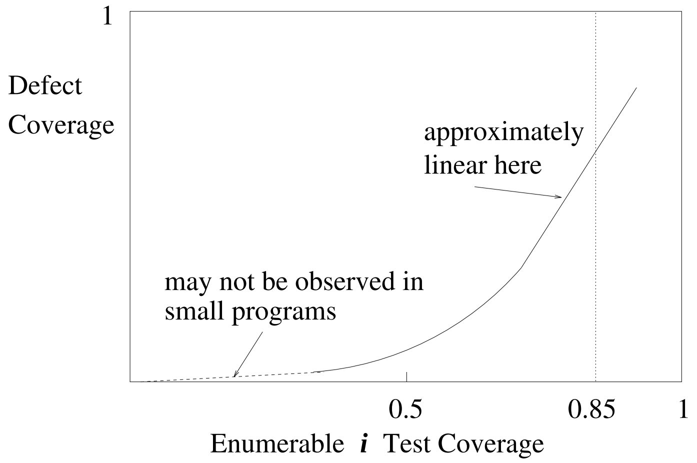

Figure 7: Defect Coverage vs Test Coverage

extent p-use coverage, are both strongly correlated with block coverage. The correlation with c-use coverage is weaker.

Table 7: Summary table for DS4(796 test cases)

	Blocks	Branches	C-uses	P-uses	Defects
Final Coverage	94%	93%	94%	87%	90%
b_0^i	0.063	0.072	0.051	0.077	0.116
b_1^i	9759	1400	4.4E5	214	3.78
LSE	0.013	0.017	0.012	0.011	0.01
a_0^i	0.116	0.116	0.11	0.116	
a_1^i	6E-4	3.8E-3	1E-5	0.017	
a_2^i	15.23	13.4	19.20	12.95	
LSE	0.022	0.022	0.04	0.01	

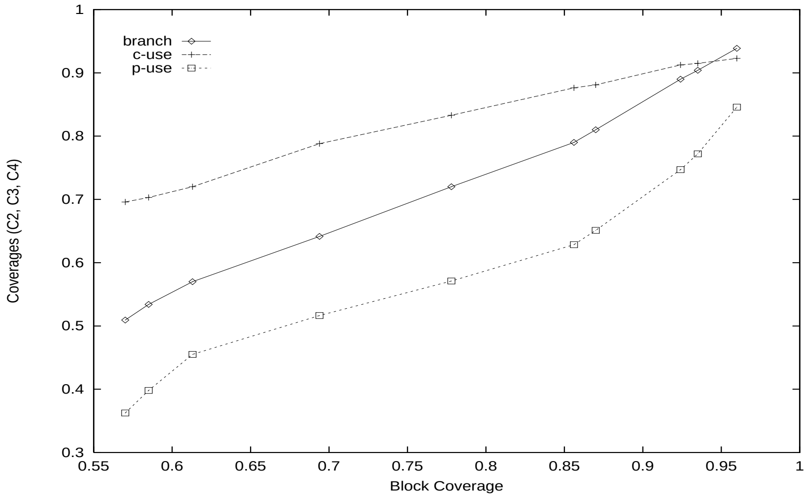

Figure 8: Plot of C2, C3 and C4 against C1 (DS4)

In evolving programs, significant changes are being made while testing is in progress. Because new modules are being added, new defects as well as non-covered enumerables are also being added. The coverage obtained by a test set can actually go down in some cases. From DS5, (see Figure 9), the linear correlation between coverage measures can still be applicable. The data used here covers an intermediate phase of the process. The analysis of evolving programs is more complex and is the subject of future research. In very small programs, if the bugs are being fixed as they found, the behavior will be similar to evolving programs. This is because fixing each bug can replace a significant fraction of the code, thus possibly reducing coverage [10].

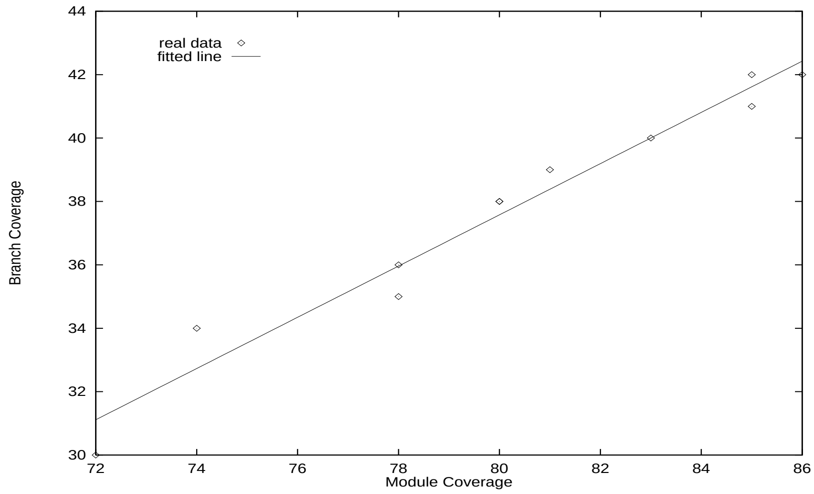

Figure 9: Scatter chart for module & branch coverages for an evolving program (DS5)

7 Model Parameters

Researchers have shown that the logarithmic model generally works well [17, 20], however interpretation of its parameters is difficult. One interpretation given by Malaiya et al [18] is described by Equations 10 and 11. The same interpretation may be applicable for enumerables other than defects. The first parameter of Equation 9 is,

$$b_0^i = \frac{K^i(0)N^i}{a^i T_L N^i} = \frac{K^i(0)}{a^i T_L} \tag{13}$$

The linear execution time is given by the number of lines of code multiplied by the average execution time of each line. An empirical method to estimate the initial fault exposure ratio $K^0(0)$ has been suggested by Li and Malaiya [14]. Estimation of a^i remains an open problem. The second parameter is given by,

$$b^i = a^0 T_s \tag{14}$$

The single test execution time T_s depends on the program size and its structure. The product $b_0^i b_1^i$ then should be independent of the program size.

The parameters a_0^i, a_1^i, and a_2^i are defined in terms of b_0^i and b_1^i above. When this definition for a_0^i, a_1^i, and a_2^i is used as an initial estimate for numerically fitting Equation 12, the initial estimate itself may provide a least-square fit. If the initial estimates are significantly different, then the least square fit may yield somewhat different parameter values.

A-priori estimation of model parameters remains a partly unsolved problem. Currently we must rely on curve fitting based approaches. As can be seen from the tables, the parameter values for a_0^i and a_2^i appear to be relatively stable. Thus it is possible that they may be estimable with reasonable accuracy. Extensive data collection is needed to examine this question in detail.

8 Defect density and reliability

Since the failure intensity is proportional to the number of defects [18], we have,

$$\lambda = \frac{K}{T_L} N$$

Where K is the overall value of fault exposure ratio.

Let N_0 be the total number of faults initially present in the program and there is no new fault introduced during testing process. Then N can be computed as:

$$N = N_0(1 - C^0)$$

Substituting C^0 using Equation 12,

$$N = N_0(1 - a_0^i \ln[1 + a_1^i(exp(a_2^i C^i) - 1)])$$

Hence, the expected duration between successive failures can be obtained as

$$\frac{1}{\lambda} = \frac{T_L}{K} \frac{1}{N_0(1 - a_0^i \ln[1 + a_1^i(exp(a_2^i C^i) - 1)])} \tag{15}$$

Equation 15 can also be used for operational period using the appropriate value for the fault exposure ratio. Notice that K will depend on the operational profile used. Figure 10 depicts the MTTF trend described by Equation 15. MTTF rises sharply at coverage values higher than 80%, suggesting higher coverage can significantly reduce rate at which failures are encountered.

9 Conclusions and Discussions

We developed a modeling scheme that relates defect density to measurable coverage metrics. Defects have a detectability distribution like other coverage enumerables, and the same model may govern them. The proposed model is consistent with the time-based Logarithmic model. It takes into account the fact that at 100% coverage for a given enumerable, all the defects may not yet have been found.

The data sets used suggest that the model works well. The results are consistent with the analytical coverage inclusion relationships. Our model is simple and easily explained, and is thus

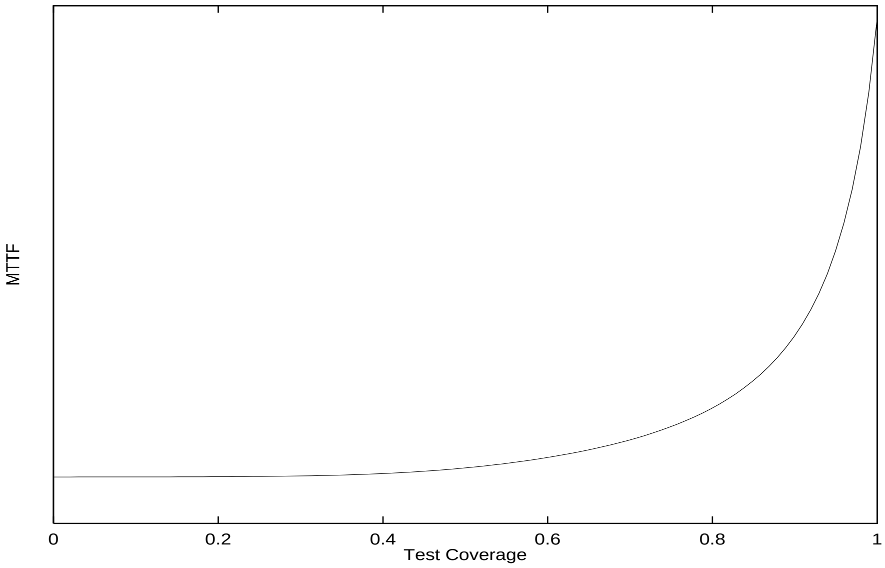

Figure 10: MTTF with respect to test coverage

suitable for industrial use. A preliminary implementation of the model as part of the ROBUST tool is presented in [15].

The model given by Equation 12 can be used in two different ways. *Extrapolation* requires collecting data for part of the testing process, which is then used to estimate the applicable parameter values. These are used for making projections for planning the rest of the test effort. *A priori* parameter estimation requires empirical estimation of parameters even before testing begins. We have some observations on what factors control the parameter values. Further work is needed to fully develop these techniques and would include a careful study of enumerable exposure ratios.

As we show, any test coverage measure can be used to estimate the defect density, by using applicable parameter values. We need further studies to determine which coverage measure provide the best estimates of the number of defects. For very high reliability, we may need a "scale" that works in that region. If the requirements are such that 100% block coverage is not enough, branch or p-use coverage may be more appropriate. Branch coverage may be an adequate measure in many cases, since about 80% branch coverage often produces acceptable results [11]. However for testing of individual modules or for highly reliable software p-use may be better.

Researchers suggests the use of a weighted risk measure [2, 24, 21]. Weights are chosen on the basis of relative significance of each measure. As we find, structural coverage measures tend to be strongly correlated, and thus a weighted average may not provide more information than a single measure. We need to identify more independent measures. Other types of coverage measures like functional coverage may be suitable in some cases.

Our results can serve as a basis for further data collection and analysis. We need to examine the behavior at different fault densities, especially at very low defect densities (for highly reliable applications). We also need to evaluate the effect of different testing strategies on the modeling scheme and the parameter values.

10 Acknowledgement

We would like to thank Mladen Vouk for providing us some the data sets, and Alberto Pasquini, Bob Horgan and Aditya Mathur for discussions on this subject.

References

[1] J. R. Adams. Software reliability predictions are practical today. In *7th Annual Software Reliability Symposium*, Denver, Colorado, May 1989.

[2] H. Agrawal, J. Horgan, E. Krauser, and S. London. A testing-based model and risk browser for C. In *Int'l Conf. Rel. Qual. Control & Risk Assessment*, pages 1–7, Oct. 1993.

[3] B. Beizer. *Software Testing Techniques.* Van Nostrand Reinhold, 2nd edition, 1990.

[4] J. Bieman and J. Schultz. An empirical evaluation (and specification) of the all-du-paths testing criterion. *Software Engineering Journal*, pages 43–51, Jan. 1992.

[5] M. Chen, J. Horgan, A. Mathur, and V. Rego. A time/structure based model for estimating software reliability. Technical Report SERC-TR-117-P, Purdue University, Dec. 1992.

[6] L. A. Clarke, A. Podgurski, D. J. Richardson, and S. J. Zeil. A formal evaluation of data flow path selection criteria. *IEEE Trans. Software Engineering*, SE-15(11):1318–1332, Nov. 1989.

[7] S. Dalal, J. Hogan, and J. Kettenring. Reliable software and communications: software quality, reliability and safety. In *15 Int'l Conf. Software Engineering*, pages 425–435, May 1993.

[8] P. Frankl and N. Weiss. An experimental comparison of the effectiveness of branch testing and data flow testing. *IEEE Trans. Software Engineering*, pages 774–787, Aug. 1993.

[9] P. Frankl and E. Weyuker. An applicable family of data flow testing criteria. *IEEE Trans. Software Engineering*, pages 1483–1498, Oct. 1988.

[10] F. D. Frate, P. Garg, A. P. Mathur, and A. Pasquini. Experiments to investigate the correlation between code coverage and software reliability. Technical Report SERC-TR-162-P, Purdue University, Software Engineering Research Center, Purdue University, West Lafayette, Indiana 47907, Apr. 1995.

[11] R. E. Grady. *Practical Software Metrics for Project Management and Process Improvement.* Prentice-Hall, 1992. pp. 58-60.

[12] H. Hecht and P. Crane. Rare conditions and their effect on software failures. In *Annual Reliability and Maintainability Symp.*, pages 334–337, Jan. 1994.

[13] M. Hutchings, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Int'l Conf. Software Engineering*, pages 191–200, 1994.

[14] N. Li and Y. K. Malaiya. Fault exposure ratio and software reliability. In *3rd Workshop on Issues in Software Reliability and Testing*, pages 6.3.1–6.3.18, Boulder, Colorado, Oct. 1993.

[15] N. Li and Y. K. Malaiya. Robust: A next generation software reliability engineering tool. In *Proc. Int'l Symp. on Software Reliability Engineering*, France, Oct. 1995 (to appear).

[16] M. Lyu, J. Horgan, and S. London. A coverage analysis tool for the effectiveness of software testing. In *Int'l Symposium on Software Reliability Engineering*, pages 25–34, 1993.

[17] Y. Malaiya, N. Karunanithi, and P. Verma. Predictability of software reliability models. *IEEE Trans. Reliability*, pages 539–546, Dec. 1992.

[18] Y. Malaiya, A. von Mayrhauser, and P. Srimani. An examination of fault exposure ratio. *IEEE Trans. Software Engineering*, pages 1087–1094, Nov. 1993.

[19] Y. Malaiya and S. Yang. The coverage problem for random testing. In *Int'l Test Conference*, pages 237–242, Oct. 1984.

[20] J. D. Musa, A. Iannino, and K. Okumoto. *Software Reliability - Measurement, Prediction, Applications.* McGraw-Hill, 1987.

[21] A. M. Neufelder. *Ensuring Software Reliability.* Marcel Dekker Inc., 1993. pp. 137-140.

[22] S. Ntafos. A comparision of some structural testing strategies. *IEEE Trans. Software Engineering*, pages 868–874, June 1988.

[23] P. Piwowarshki, M. Ohba, and J. Caruso. Coverage measurement experience during function test. In *International Conference on Software Engineering*, pages 287–300, Baltimore, Maryland, May 1993.

[24] R. Poston. The power of simple software testing metrics. In *Int'l Conf. on Software Engineering*, 1993.

[25] J. Ramsey and V. Basili. Analyzing the test process using structural coverage. In *Int'l Conf. on Software Engineering*, pages 306–312, 1985.

[26] S. Rapps and E. Weyuker. Selecting software test data using data flow information. *IEEE Trans. Software Engineering*, SE-11(4):367–375, Apr. 1985.

[27] S. Seth, V. Agrawal, and H. Farhat. A statistical theory of digital circuit testability. *IEEE Trans. on Computer*, pages 582–586, Apr. 1990.

[28] M. Trachtenberg. Why failure rates observe Zipf's law in operational software. *IEEE Trans. Reliability*, pages 386–389, Sept. 1992.

[29] J. Voas and K. Miller. Improving the software development process using testability research. In *3rd Int'l Symp. Software Reliability Engineering*, pages 114–121, 1992.

[30] M. Vouk. Using reliability models during testing with non-operational profiles. In *2nd Bellcore/Purdue Workshop Issues in Software Reliability Estimation*, pages 103–111, Oct. 1992.

[31] K. Wagnor, C. Chin, and E. McCluskey. Pseudorandom testing. *IEEE Trans. Computer*, pages 332–343, Mar. 1987.

[32] E. Weyuker. More experience with data flow testing. *IEEE Trans. Software Engineering*, pages 912–919, Sept. 1993.