Software Reliability Growth With Test Coverage

Yashwant K. Malaiya, *Senior Member, IEEE*, Michael Naixin Li, James M. Bieman, *Senior Member, IEEE*, and Rick Karcich

***Abstract*—"Software test-coverage measures" quantify the degree of thoroughness of testing. Tools are now available that measure test-coverage in terms of blocks, branches, computation-uses, predicate-uses, etc. that are covered. This paper models the relations among testing time, coverage, and reliability. An LE (logarithmic-exponential) model is presented that relates testing effort to test coverage (block, branch, computation-use, or predicate-use). The model is based on the hypothesis that the enumerable elements (like branches or blocks) for any coverage measure have various probabilities of being exercised; just like defects have various probabilities of being encountered. This model allows relating a test-coverage measure directly with defect-coverage. The model is fitted to 4 data-sets for programs with real defects. In the model, defect coverage can predict the *time to next failure*.**

The LE model can eliminate 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. The data-sets used suggest the potential of the proposed model. The model is simple and easily explained, and thus can be suitable for industrial use. The LE model is based on the time-based logarithmic software-reliability growth model. It considers that: at 100% coverage for a given enumerable, all defects might not yet have been found.

***Index Terms*—Defect density, reliability-growth model, software reliability, software testing, test coverage.**

DEFINITIONS

- enumerable: structural or data-flow components of a program that can be counted (like branches or p-uses).
- subsumption: if complete coverage of enumerable i implies complete coverage of enumerable j, then the coverage of i subsumes the coverage of j.

ACRONYMS[1]

c-	computation-
p-	predicate-
s-	implies the technical statistical meaning
LE	logarithmic-exponential (the proposed model)
DC	defect coverage
DSi	data-set i ($i = 1, 2, 3, 4$)
RGM	reliability growth model
TC	test coverage

NOTATION

0	superscript 0 indicates defects
i	superscripts 1, 2, 3, 4 indicate specific enumerables
a_0^i, a_1^i, a_2^i	parameters for proposed LE model in terms of enumerable i used in (4)
A^i, B^i	parameters used in (5)
b_0^i, b_1^i	parameters used for (3)
$C^0(t)$	DC at time t
$C^j(n)$	s-expected coverage of the enumerables of type j
C^j_{knee}	coverage level at which the knee occurs
K	overall value of fault-exposure ratio
K^i	fault or enumerable exposure ratio
N^0	total number of initial defects
N_0^i	total number of enumerables of type i
t_f	time when debugging stops
T_L	linear execution time
β_0^i, β_1^i	logarithmic-model parameters for enumerable i
λ	failure intensity.

I. 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, developers can optimally allocate resources to meet a deadline with the target reliability [21].

To quantify reliability during testing, the code is executed using inputs randomly selected following some distribution. Then, a reliability growth model can be used to predict the amount of effort required to satisfy product reliability requirements, if the distribution used for testing is the same as the operational profile. However, the focus of testing is on finding defects, and defects can be often found much faster by nonrandom methods [1]. Testing is directed toward inputs and program-components where errors are more likely. For example, testing can 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 nonrandom testing are clearly needed. The achieved reliability is affected by several factors:

Manuscript received September 13, 1997; revised October 22, 1999, July 18, 2001, and February 19, 2002. The work of Y. K. Malaiya and M. N. Li was supported in part by a BMDO funded project monitored by ONR. The work of J. M. Bieman was supported in part by NSF, the NASA Langley Research Center, the Colorado Advanced Software Institute, Storage Technology Inc., and Micro-Motion Inc.

Y. K. Malaiya and J. M. Bieman are with the Computer Science Department, Colorado State University, Fort Collins, CO 80525 USA (e-mail: Malaiya@cs.colostate.edu; Bieman@cs.colostate.edu.)

M. N. Li is with Rm 422682, Redmond, WA 98052 USA (e-mail: NaixinLi@microsoft.com).

R. Karcich is with Sun Microsystem, Louisville, CO 80028 USA (e-mail: Richard.Karcich@central.sun.com).

Digital Object Identifier 10.1109/TR.2002.804489

[1]The singular and plural of an acronym are always spelled the same.

0018-9529/02$17.00 © 2002 IEEE

• Testing strategy: TC can be based on the functional specification (black-box), or 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.

• Testing of rarely executed modules: Such modules include exception handling or error recovery routines. These modules rarely run [10], and are notoriously difficult to test. Yet, they are critical components of a system that must be highly reliable.

Intuition suggests that TC must be related to reliability. Yet, the connection between structure-based measurements (e.g., TC) and reliability is still not well understood.

There are several motivations for investigating the relation between TC and reliability. TC, rather than test effort, is a direct measure 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 might be less efficient in finding defects. Measuring TC is usually an intrusive approach; however available tools now allow it to be done automatically.

The effectiveness of testing in finding defects has been recently examined by several researchers. Reference [7] examines the correlation between TC and the error removal rate. Reference [3] suggests that the relation between structural coverage and fault coverage is a variant of the Rayleigh distribution. References [4], [5] add structural coverage to traditional time-based software reliability models by excluding test cases that do not increase coverage. Assuming random testing, [24] analyzes block-coverage growth during function test, and derives an exponential model relating the number of tests to block coverage. Reference [11] experiments with detection of defects in small programs. Reference [12]: a) studies detection effectiveness of test sets with various coverage values for realistic seeded faults; b) finds that a test set with higher coverage has higher per-test detection probability; c) shows that 100% coverage using a specific measure might not detect all the faults.

This paper explores the connection between TC and reliability by developing a model that relates TC to DC. With this model the defect density can be estimated. With knowledge of the fault exposure ratio, reliability can be predicted from TC measures.

II. Coverage of Enumerables

TC in software is measured in terms of structural or data-flow units that have been exercised. Some of the common coverage measures are:

• 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 c-uses that have been covered during testing. A c-use pair includes 2 points in the program, a point where the value of a variable is defined or modified, followed by a point where it is used for computation (without the variable being modified along the path) [14], [23].

• P-use coverage: The fraction of the total number of p-uses that have been covered during testing. A p-use pair includes 2 points in the program, a point where the value of a variable is defined or modified, followed by a point which is a destination of a branching statement where it is used as a predicate (without modifications to the variable along the path) [14], [23].

To keep this discussion general, the term *enumerable* indicates a unit covered by testing [17]. For DC, the enumerables are defects, for branch coverage, the enumerables are branches, *et al.*

Enumerable-type implies defects, blocks, branches, c-uses, or p-uses. The superscript i, $i = 0, 1, 2, 3, 4$ identifies 1 of the 5 types:

0: defects,
1: blocks,
2: branches,
3: c-uses,
4: p-uses.

It is assumed that no functional changes are being attempted; and thus no new code is being added to the software under test.

When an enumerable is exercised, one or more associated faults can be detected. "Counting the number of covered units" gives a measure of the extent of sampling. Sometimes 85% branch coverage is considered to be the minimum acceptable value [9]. The DC in software can be defined in an analogous manner: the fraction of actual defects initially present that would be detected by a given test set.

In general, TC increases when more test cases are applied, as long as the test cases are not repeated and complete TC has not already been achieved. A few enumerables might not be reachable in practice. Assume that the fraction of such enumerables is negligible.

It has been shown that if all paths in the program have been exercised, then all p-uses must have been covered. Similarly all p-use coverage implies all-branches coverage, and all-branches coverage implies all-instructions coverage. This is termed the *subsumption hierarchy* [2], [6], [14].

III. A New LE Coverage Model

This paper uses the Musa–Okumoto logarithmic growth model [8], [15], [18], [19], [21]. Assume that the DC growth follows the logarithmic model:

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

Because the maximum value of coverage is 1, (1) applies to coverage values ≤ 1.

Assume that coverage growth of enumerable i follows the logarithmic model ($i = 1, 2, 3, 4$):

$$C^i(t) = \frac{1}{N^i} \beta_0^i \cdot \log(1 + \beta_1^i \cdot t), \qquad C^i(t) \le 1. \tag{2}$$

Both (1) and (2) can be considered as 2-parameter models. The maximum value of $C^i(t)$ is 1. Once this value is reached during testing, it remains 1 with further testing. Equation (2), in the general form, is

$$C^i(t) = b_0^i \cdot \log(1 + b_1^i \cdot t), \qquad C^i(t) \le 1,\ i = 1, 2, 3, 4. \tag{3}$$

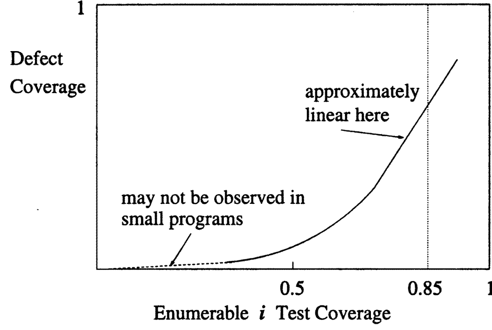

Fig. 1. DC versus TC.

Equation (2) relates C^i to the number of tests applied. It is used to obtain DC C^0 in terms of one of the C^i, $i = 1, 2, 3, 4$. Using (2) solve for t,

$$t = \frac{1}{\beta_1^i} \cdot \left[\exp\left(\frac{C^i \cdot N_0^i}{\beta_0^i}\right) - 1\right], \qquad i = 1, 2, 3, 4.$$

Substitute t for C^0 in (1):

$$C^0(C^i) = \frac{\beta_0^0}{N_0^0} \cdot \log\left[1 + \frac{\beta_1^0}{\beta_1^i} \cdot \left(\exp\left(\frac{C^i N_0^i}{\beta_0^i}\right)\right) - 1)\right], \quad i = 1, 2, 3, 4.$$

Let

$$a_0^i \equiv \frac{\beta_0^0}{N_0^0}, \qquad a_1^i \equiv \frac{\beta_1^0}{\beta_1^i}, \qquad a_2^i \equiv \frac{N_0^i}{\beta_0^i}.$$

Then write the previous equation using these 3 parameters as,

$$C^0(C^i) = a_0^i \cdot \log\left[1 + a_1^i \cdot (\exp(a_2^i \cdot C^i) - 1)\right] \quad i = 1, 2, 3, 4 \tag{4}$$

Equation (4) is a convenient 3-parameter model for DC in terms of a measurable TC metric. Equation (4) applies only for $C^0 \leq 1$.

Fig. 1 plots the relationship of DC versus TC, as given by (4). The overall curve is nonlinear, although the initial segment might not be observed in small programs because even a single test execution can provide close to 50% enumerable coverage. The location of the knee of the curve depends on the initial defect density [20]. Fig. 1 shows that the curve can be approximated by a linear plot when C^i exceeds C^i_{knee}.

Equation (4) results in a linear expression when $a_1^i \cdot \exp(a_2^i \cdot C^i) \gg 1$, and when $\exp(a_2^i \cdot C^i) \gg 1$. Analysis of actual data in Section IV suggests that $a_1^i \ll 1$. Thus $a_1^i \cdot \exp(a_2^i C^i) \gg 1$ implies: $\exp(a_2^i \cdot C^i) \gg 1$.

The knee at C^i_{knee} is influenced by the initial defect density [20]. A low initial defect density can mean that easy-to-find defects have already been found and removed. Thus one would begin finding new defects only when TC is sufficiently high.

For $C^i > C^i_{\text{knee}}$, a linear approximation for C^0 is:

$$C^0 \approx a_0^i \cdot \log\left(a_1^i \cdot \exp(a_2^i \cot C^i)\right) = -A^i + B^i \cdot C^i, \quad C^i > C^i_{\text{knee}}, \tag{5}$$

TABLE I
DATA-SETS USED

DataSet	KLOC	# of Tests	# of Defects	Tool
DS1 [13]	30	21k	66	ATAC
DS2 [3]	5	1196	9	BCG
DS3 [3]	5	796	9	BCG
DS4 [3]	5	796	7	BCG

KLOC ≡ kilo number of lines of code;
ATAC, BCG ≡ names of the tools used to collect the coverage data .

TABLE II
COVERAGE DATA: DS3

NASA project: Sensor management in inertial management (integration/acceptance test phase: 9 faults found with 796 tests)
CF ≡ Cumulative Faults

CF	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 III
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	0.0	0.184
b_1^i	2E+8	1234	3.4E+6	29	0.01
LSE	5.7E-4	3.5E-5	5.8E-4	8.1E-5	7.3E-7

A^i, B^i are the parameters for the linear approximation.

The full TC of an enumerable does not imply full DC. Full statement coverage can be reached before full branch-coverage because of the subsumption hierarchy.

IV. ANALYSIS OF DATA

The proposed model, given by (2) and (3) is fitted using the 4 data sets in Table I.

DS1 is from a 12-version automatic airplane landing system [13] software. It was collected using the ATAC tool developed at Bellcore. The 12 versions of the software have a total of 30 694 lines of code. The data used are 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, DS4 are from a NASA supported project implementing sensor management in an inertial navigation system [3]. As an example, DS3 is reproduced in Table II.

The results for data set DS1 are summarized in Table III. Row #1 gives the total number of enumerables for all versions. Row #2 gives the average coverage when 21 000 tests had been applied. The estimated values of b_0^i, b_1^i, and LSE are given in the remaining rows.

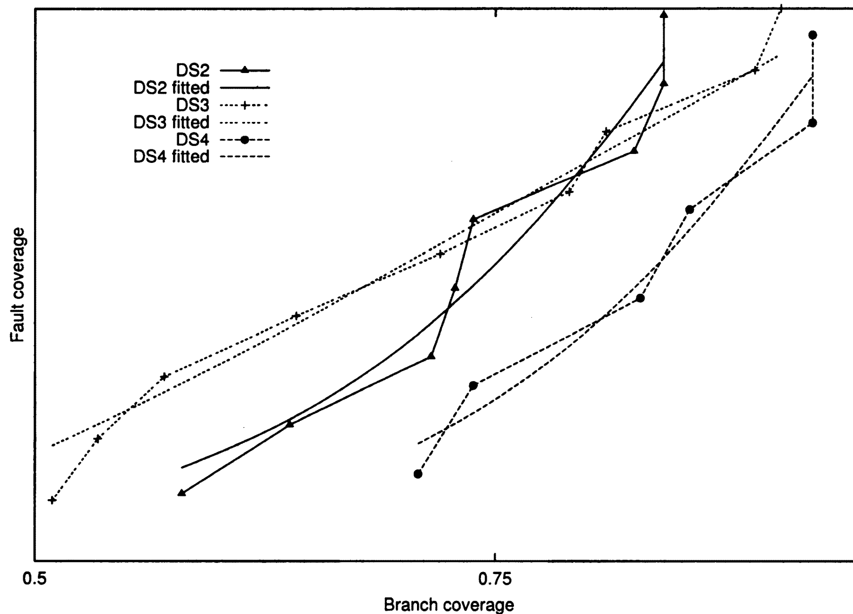

Fig. 2. Actual and fitted values of DC for DS2, DS3, DS4.

TABLE IV
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	

TABLE V
SUMMARY TABLE FOR DS3 (796 TEST CASES)

Final Coverage	Blocks 96%	Branches 94%	C-uses 92%	P-uses 85%	Defects 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	

Table IV summarizes the result for DS2; 9 faults were revealed by application of 1196 tests; 1 fault (*viz*, 10%) is assumed to be still undetected.

Fig. 2 shows actual and computed values for fault coverage for data sets DS2, DS3, DS4. The computed values were obtained using branch coverage and (4). The knee occurs at various branch coverage values. For Data Set DS2 (shown by a solid line), at 50% branch coverage the fault coverage is still quite low (about 10%); however with 84% branch coverage, 90% fault coverage is obtained. Figs. 2 and 4 assume that in each case, 1 fault is still undetected. In practice, estimating the number of remaining defects is a major challenge that needs further investigation.

Table V presents the results for DS3 which involves 796 test cases. The values of the parameters obtained can be compared with the values for DS2 in Table IV.

Fig. 3 plots the coverage growth of various enumerables.

Fig. 4 plots actual and model DC values against branch coverage for DS3. It shows how the relative defect density declines as branch coverage increases. The vertical line represents 85% branch coverage (for reference).

Table VI summarizes the results for DS4. Fig. 5 illustrates the correlation of other TC measures, C^2, C^3, C^4, with block coverage C^1. As anticipated, branch coverage, and to a lesser extent, p-use coverage, are both strongly correlated with block coverage. The correlation with c-use coverage is weaker.

V. DEFECT DENSITY AND RELIABILITY

Consider the failure intensity during the operational period. Assume that debugging stops at t_f, and no further changes in the program are made. After t_f, the defects remaining are not removed. Thus λ no longer depends on time. Because λ is proportional to the number of defects [19], then

$$\lambda(t_f) = \frac{K}{T_L} \cdot N^0(t_f).$$

Reference [19] shows that the value of K ranges from $1 \cdot 10^{-7}$ to $7.5 \cdot 10^{-7}$ failures/fault, for several data-sets examined. The value of K does not depend on the program size, but can depend on defect distribution in the program and the testing approach.

During testing and debugging, the faults found are removed. If no new faults are introduced during this process, then the total number of defects to be found by t_f is:

$$N^0(t_f) = N_0^0 \cdot \left(1 - C^0(t_f)\right).$$

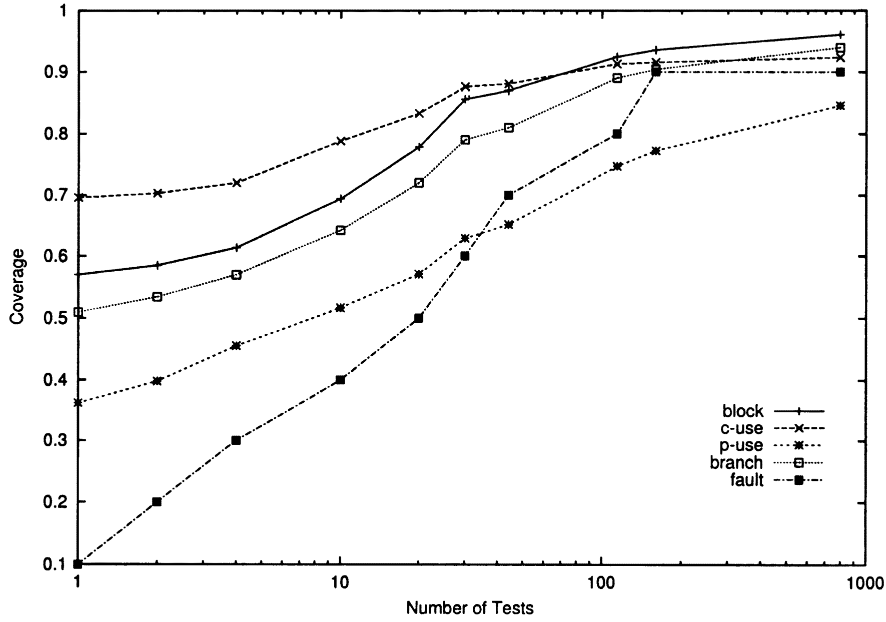

Fig. 3. Coverage growth of various enumerables (DS3).

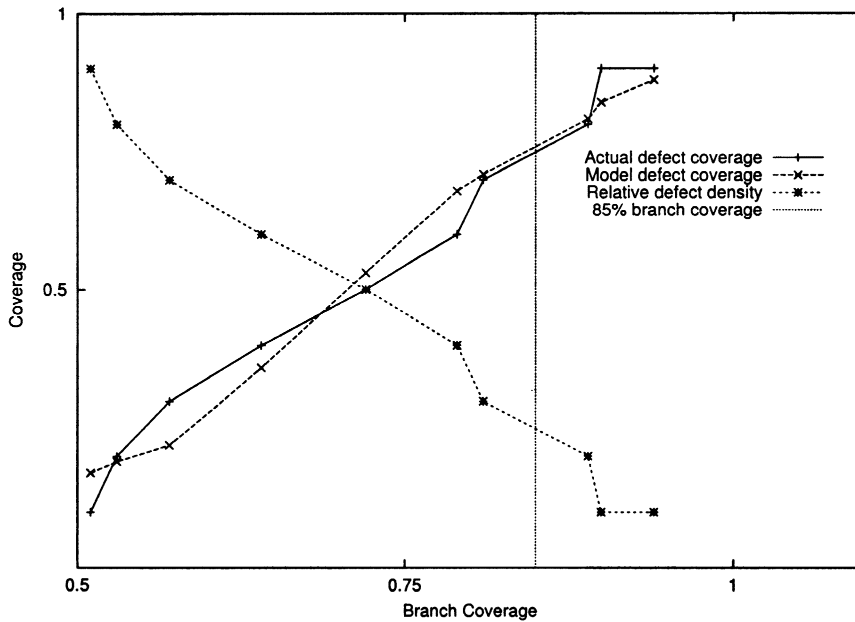

Fig. 4. Fault coverage and relative defect density (DS3).

In practice, debugging can be imperfect [22].

Substitute C^0 using (4):

$$N^0(t_f) = N_0^0 \cdot \left(1 - a_0^i \cdot \log\left[1 + a_1^i \cdot \left(\exp[a_2^i \cdot C^i(t_f)] - 1\right)\right]\right).$$

Hence, the mean duration between successive failures is:

$$\frac{1}{\lambda(t_f)} = \frac{T_L}{K} \cdot \frac{1}{N_0 \cdot \left(1 - a_0^i \cdot \log\left[1 + a_1^i \cdot \left(\exp(a_2^i \cdot C^i(t_f)) - 1\right)\right]\right)}. \quad (6)$$

Equation (6) can also be used for the operational period with the appropriate value for the fault-exposure ratio. K depends on the operational profile encountered during the operational period [21].

VI. Future Work

Further experimental and theoretical research is needed to validate the model in this paper. Analysis of additional data sets will provide further insight into the problem. This paper evaluates the values of a_1^i, a_2^i, a_2^i by curve fitting. It will be useful

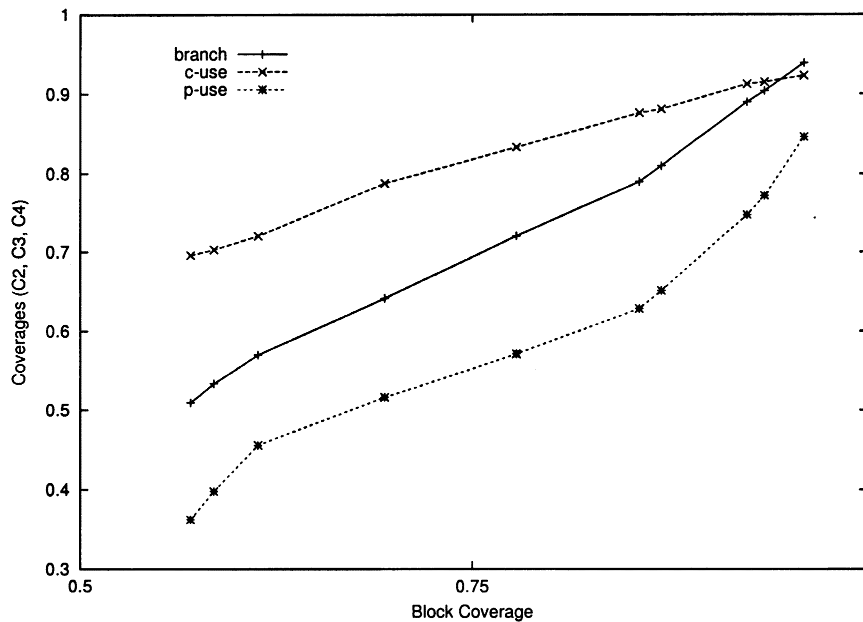

Fig. 5. Plot of C2, C3, C4 against C1 (DS4).

TABLE VI
SUMMARY TABLE FOR DS4 (796 TEST CASES)

Final Coverage	Blocks 94%	Branches 93%	c-Uses 94%	p-Uses 87%	Defects 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	

to obtain initial estimates of the parameter values using empirical methods. That would involve interpreting the parameters for the logarithmic model [16], [19]. Estimating the number of remaining defects is another problem that needs further investigation.

ACKNOWLEDGMENT

The authors would like to thank M. Vouk for providing some of the data sets, and A. Pasquini, B. Horgan, A. Mathur, and B. Skibbe for discussions on this subject.

REFERENCES

[1] B. Beizer, *Software Testing Techniques*: Van Nostrand Reinhold, 1990, pp. 74–75, 161–171.

[2] J. M. Bieman and J. L. Schultz, "An empirical evaluation (and specification) of the all-du-paths testing criterion," *Software Engineering J.*, pp. 43–51, Jan. 1992.

[3] M. A. Vouk, "Using reliability models during testing with nonoperational profiles," in *Proc. 2nd Bellcore/Purdue Workshop on Issues in Software Reliability Estimation*, Oct. 1992, pp. 103–111.

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

[5] M. H. Chen, M. R. Lyu, and W. E. Wong, "An empirical study of the correlation between code coverage and reliability estimation," in *Proc. Int. Software Metrics Symp.*, 1996, pp. 133–141.

[6] L. A. Clarke, A. Podgurski, D. J. Richardson, and S. J. Zeil, "A formal evaluation of the data flow path selection criteria," *IEEE Trans. Software Engineering*, pp. 1318–1332, Nov. 1989.

[7] S. R. Dalal, J. R. Horgan, and J. R. Kettenring, "Reliable software and communications: Software quality, reliability and safety," in *Proc. 15th Int. Conf. Software Engineering*, May 1993, pp. 425–435.

[8] W. Farr, "Software reliability modeling survey," in *Hdbk. of Software Reliability Engineering*, M. R. Lyu, Ed: McGraw-Hill, 1996, pp. 71–117.

[9] R. E. Grady, *Practical Software Metrics for Project Management and Process improvement*: Prentice-Hall, 1992, pp. 58–60.

[10] H. Hecht and P. Crane, "Rare conditions and their effect on software failures," in *Proc. Ann. Reliability & Maintainability Symp.*, 1994, pp. 334–337.

[11] P. Frankl and S. N. Weiss, "An experimental comparison of the effectiveness of branch testing and data flow testing," *IEEE Trans. Software Engineering*, vol. 19, no. 8, pp. 774–787, Aug. 1993.

[12] M. Hutchings, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of data-flow and control-flow based test data adequacy criteria," in *Int. Conf. Software Engineering*, 1994, pp. 191–200.

[13] M. R. Lyu, J. R. Horgan, and S. London, "A coverage analysis tool for the effectiveness of software testing," in *IEEE Int. Symp. Software Reliability Engineering*, 1993, pp. 25–34.

[14] S. Rapps and E. J. Weyuker, "Selecting software test data using data flow information," *IEEE Trans. Software Engineering*, pp. 367–375, Apr. 1985.

[15] Y. K. Malaiya, N. Karunanithi, and P. Verma, "Predictability of software reliability models," *IEEE Trans. Reliability*, vol. 41, pp. 539–546, Dec. 1992.

[16] Y. K. Malaiya and J. Denton, "What do the software reliability growth model parameters represent," in *Proc. IEEE Int. Symp. Software Reliability Engineering*, Nov. 1997.

[17] Y. K. Malaiya, N. Li, and J. Bieman *et al.*, "The relationship between test coverage and reliability," in *Proc. Int. Symp. Software Reliability Engineering*, Nov. 1994, pp. 186–195.

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

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

[20] Y. K. Malaiya and J. Denton, "Estimating the number of residual defects," in *Proc. IEEE High Assurance Systems Engineering Symp.*, 1998, pp. 98–105.

[21] J. Musa, *Software Reliability Engineering*: McGraw-Hill, 1999.

[22] M. Ohba and X. M. Chou, "Does imperfect debugging affect software reliability growth?," in *Proc. 11th Int. Conf. Software Engineering*, May 1989, pp. 237–244.

[23] J. Ramsey and V. R. Basili, "Analyzing the test process using structural coverage," in *Proc. 8th Int. Conf. Software Engineering*, Aug. 1985, pp. 306–312.

[24] P. Piwowarski, M. Ohba, and J. Caruso, "Coverage measurement experience during function test," in *Proc. 15th Int. Conf. Software Engineering*, May 1993, pp. 287–300.

Yashwant K. Malaiya is a Professor in the Computer Science Department at Colorado State University. During 1978–1982, he was with State University of New York, Binghamton. He received the B.Sc. and M.Sc. in physics from Sagar University, the M.Sc. Tech. in electronics from BITS Pilani, and the Ph.D. in electrical engineering from Utah State University. He has done research in fault-modeling, testing, reliability, and fault-tolerance. He has served as General Chairperson for IEEE International Work Defect-Based Testing 2000, IEEE Asian Test Symposium 1999, International Conf. VLSI Design 1993, IEEE International Symp. Software Reliability Engineering 1993, and International Symp. Microarchitecture 1991. He received the IEEE CS Golden Core Award in 1996 and the IEEE Third Millenium Medal in 2000.

Michael Naixin Li received his Ph.D. in computer science from Colorado State University. He has been with Microsoft since 1995, and is a senior engineer specializing in software testing.

James M. Bieman is a Professor in the Computer Science Department at Colorado State University. He joined Colorado State in 1989 after serving as a faculty member in the Computer Science Department at Iowa State University for 5 years. He received a B.S. (chemical engineering) from Wayne State University; M.P.P. (Public Policy) from the University of Michigan; and an M.S. and Ph.D. in computer science from the University of Louisiana, Lafayette. His research focus is software design, evaluation, and improvement. He is the Editor-in-Chief of the Software Quality Journal.

Rick Karcich is a Staff Engineer in the Network Storage Division of Sun Microsystems, with responsibility for testing highly fault-tolerant disk-storage subsystems. He is interested in measuring testing-effectiveness since serving as the first chairperson of the IEEE Subcommittee on Software Reliability.