Software Reliability Growth with Test Coverage

Yashwant K. Malaiya, Senior Member IEEE
Colorado State University, Fort Collins

Naixin Li
Microsoft, Redmond

James M. Bieman, Senior Member IEEE
Colorado State University, Fort Collins

Rick Karcich
Sun Microsystem, Broomfield

1 Key Words

Software reliability, software testing, test coverage, reliability-growth model, defect density.

2 Summary and Conclusions

Software test coverage measures quantify the degree of thoroughness of testing. Tools are now available that measure test coverage in terms of blocks, branches, c-uses, p-uses, etc. covered. In this paper, we model the relations among testing time, coverage and reliability. We present a logarithmic-exponential (LE) model that relates testing effort to test coverage (block, branch, c-use or p-use). The model is based on the hypothesis that the enumerable elements (like branches or blocks) for any coverage measure have different probabilities of being exercised; just like defects have different probabilities of being encountered. This model allows us to relate a test coverage measure directly with defect coverage. We have fitted the model to four 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 presented here 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 takes into account the fact that at 100% coverage for a given enumerable, all defects may not yet have been found.

3 Introduction

Acronyms

LE model the proposed logarithmic-exponential model
DSi (i=1,2,3,4) data-set i
RGM reliability growth model

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 [mus99].

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, provided the distribution used for testing is same as the operational profile. However, the focus of testing is on finding defects, and defects can be often found much faster by non-random methods [bei90]. 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 several factors:

- 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.

- Testing of rarely executed modules: Such modules include exception handling or error recovery routines. These modules rarely run [hec94], and are notoriously difficult to test. Yet, they are critical components of a system that must be highly reliable.

  Intuition 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 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 available tools now allow it to be done automatically.

  The effectiveness of testing in finding defects has been recently examined by several researchers. Dalal, Horgan and Keterring [dhk93] have examined the correlation between test coverage and the error removal rate. Vouk [vou92] has suggested that the relation between structural coverage and fault coverage is a variant of the Rayleigh distribution. Chen et al. [chm92, chm96] add structural coverage to traditional time-based software reliability models (SRMs) by excluding test cases that do not increase coverage. Assuming random testing, Piwowarski, Ohba and Caruso [poc93] analyze block coverage growth during function test, and derive an exponential model relating the number of tests to block coverage. Frankl and Weiss [fra93] have experimented with detection of defects in small programs. Hutchins et al. [hfgo94] 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.

  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 fault exposure ratio, we can predict reliability from test coverage measures.

Notation

Superscript 0 indicates defects and superscripts 1, 2, 3, 4 indicate specific enumerables

\( C^j(n) \) expected coverage of the enumerables of type \( j \)

\( C_{\text{knee}}^j \) The coverage level at which the knee occurs

\( \beta_i^0, \beta_i^1 \) the Logarithmic model parameters for enumerable \( i \)

\( a_i^1, a_i^2, a_i^3 \) parameters for proposed model in terms of enumerable \( i \) used in Equation 4

\( b_i^0, b_i^1 \) parameters used for Equation 3

\( K^i \) fault or enumerable exposure ratio

\( T_L \) Linear execution time

\( t_f \) Time when debugging stops

\( N_0^i \) The total number of enumerables of type \( i \)

\( \lambda \) failure intensity

\( A^i, B^i \) parameters used in equation Equation 5

4 Coverage of Enumerables

Test coverage in software is measured in terms of structural or data-flow units that have been exercised. Some of the common coverage measures are 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 uses (c-uses) that have been covered during testing. A c-use pair includes two 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) [rap85, ram85].

- P-use coverage: the fraction of the total number of predicate uses (p-uses) that have been covered during testing. A p-use pair includes two 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) [rap85, ram85].

To keep the following discussion general, we will use the term enumerable to indicate a unit covered by testing [mali94]. For defect coverage the enumerables are defects, for branch coverage, the enumerables are branches, 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 identify one of the five types in this way: \( 0 \): defects, \( 1 \): blocks, \( 2 \): branches, \( 3 \): c-uses, \( 4 \): p-uses. We assume 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, 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. Sometimes 85% branch coverage is considered to be the minimum acceptable value [gra92]. 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.

In general, test coverage increases when more test cases are applied as long as 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.

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 [rap85, cla89, bisc92].
5 A New Logarithmic-Exponential (LE) Coverage Model

In this paper, we use the Musa-Okumoto logarithmic growth model [mus87, far96, mus99, mkv92, mvs93]. We hypothesize that the defect coverage growth follows the logarithmic model:

\[
C^0(t) = \frac{1}{N^0} \beta_0^0 \ln(1 + \beta_1^0 t), \quad C^0(t) \leq 1
\]

where \( C^0(t) \) is the defect coverage at time \( t \) and \( N^0 \) is the total number of initial defects. Note that since the maximum value of coverage is one, this equation is applicable for coverage values less than or equal to one.

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

\[
C^i(t) = \frac{1}{N^i} \beta_0^i \ln(1 + \beta_1^i t), \quad C^i(t) \leq 1
\]

Both equations 1 and 2 can be considered to be 2-parameters models. Note that the maximum value of \( C^i(t) \) is 1. Once this value is reached during testing, it remains 1 with further testing. Equation 2 can be given in the general form by

\[
C^i(t) = b_0^i \ln(1 + b_1^i t), \quad C^i(t) \leq 1, \quad i = 1 \text{ to } 4
\]

Equation 2 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 \( C^i, i = 1 \text{ to } 4 \). Using Equation 2, we solve for \( t \),

\[
t = \frac{1}{\beta_1^i} [\exp(\frac{C^i N^i}{\beta_0^i}) - 1], \quad i = 1 \text{ to } 4
\]

Substituting \( t \) for \( C^0 \) in Equation 1,

\[
C^0(C^i) = \frac{\beta_0^0}{N_0^0} \ln[1 + \frac{\beta_1^0}{\beta_1^i}(\exp(\frac{C^i N^i}{\beta_0^i}) - 1)], \quad i = 1 \text{ to } 4
\]

Defining \( a_0^i = \frac{\beta_0^0}{N_0^0}, a_1^i = \frac{\beta_0^i}{\beta_1^0} \) and \( a_2^i = \frac{N_0^i}{\beta_0^i} \), we can write the above using three parameters as,
\[ C^0(C^i) = a_0^i \ln[1 + a_1^i \exp(a_2^i C^i) - 1] \quad i = 1 \text{ to } 4 \] (4)

Equation 4 gives a convenient three-parameter model for defect coverage in terms of a measurable test coverage metric. Equation 4 is applicable for only \( C^0 \leq 1 \).

Figure 1 plots the relationship of defect coverage versus test coverage, as given by Equation 4. 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. The location of the knee of the curve depends on the initial defect density [mal98].

As we can see from Figure 1, the curve can be approximated by a linear plot when coverage \( C^i \) exceeds a knee in the curve. This knee value is termed \( C^i_{knee} \).

We can see that Equation 4 will result in a linear expression when \( a_1^i \exp(a_2^i C^i) \gg 1 \) and when \( \exp(a_2^i C^i) \gg 1 \). Analysis of actual data in the next section suggests that \( a_1^i \ll 1 \) thus \( a_1^i \exp(a_2^i C^i) \gg 1 \) implies \( \exp(a_2^i C^i) \gg 1 \).

The knee at \( C^i_{knee} \) is influenced by the initial defect density [mal98]. A low initial defect density may mean that easy to find defects have already been found and removed in the past. Then one would start finding new defects only when test coverage is sufficiently high.

For \( C^i > C^i_{knee} \), a linear approximation for \( C^0 \) can be given as:

\[ C^0 \approx a_0^i \ln(a_1^i \exp(a_2^i C^i)) = -A^i + B^i C^i \quad C^i > C^i_{knee} \] (5)

where \( A^i \) and \( B^i \) are the parameters for the linear approximation.

Note that full test coverage of an enumerable does not imply full defect coverage. Full statement coverage may be reached before full branch coverage because of the subsumption hierarchy.
Defect Coverage may not be observed in small programs approximately linear here

Enumerable Test Coverage

Figure 1: Defect Coverage vs Test Coverage
6 Analysis of Data

We have fitted the proposed model, as given by Equations 2 and 4, using four data sets listed in Table 1. The first data set, DS1, is from a multiple-version automatic airplane landing system [lyu93]. It was collected using the ATAC tool developed at Bellcore. The twelve versions have a total of 30,694 lines of code. 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 an inertial navigation system [vou92]. As an example, the data set DS3 is reproduced in Table 2.

Table 1: Data Sets Used

<table>
<thead>
<tr>
<th>DataSet</th>
<th>KLOC</th>
<th>#Tests</th>
<th>Defects</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS1(lyu93)</td>
<td>30</td>
<td>21k</td>
<td>66</td>
<td>ATAC</td>
</tr>
<tr>
<td>DS2(vou92)</td>
<td>5</td>
<td>1196</td>
<td>9</td>
<td>BCG1</td>
</tr>
<tr>
<td>DS3(vou92)</td>
<td>5</td>
<td>796</td>
<td>9</td>
<td>BCG1</td>
</tr>
<tr>
<td>DS4(vou92)</td>
<td>5</td>
<td>796</td>
<td>7</td>
<td>BCG1</td>
</tr>
</tbody>
</table>

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

The results for data set DS1 are summarized in Table 3. 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 \( b_0 \) and \( b_1 \) and the least square error (LSE) are given in the rows below.

Table 4 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.

Figure 2 shows actual and computed values for fault coverage for data sets DS2, DS3 and DS4. The computed values have been obtained using branch coverage and Equation 4. Note that the knee occurs at different 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
Table 2: Coverage Data: DS3
NASA project: Sensor management in inertial management [Vouk]
(integration/acceptance test phase: 9 faults found with 796 tests)

<table>
<thead>
<tr>
<th>Cumulative</th>
<th>Number of Test Cases</th>
<th>%Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faults</td>
<td>blocks</td>
<td>branches</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>57.01</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>58.50</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>61.30</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>69.39</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>77.80</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>85.61</td>
</tr>
<tr>
<td>7</td>
<td>44</td>
<td>87.00</td>
</tr>
<tr>
<td>8</td>
<td>114</td>
<td>92.40</td>
</tr>
<tr>
<td>9</td>
<td>160</td>
<td>93.50</td>
</tr>
<tr>
<td>9</td>
<td>796</td>
<td>95.99</td>
</tr>
</tbody>
</table>

only 84% branch coverage, 90% fault coverage is obtained. Note that the plots in Figures 2 and 4 assume that that in each case, one fault is still undetected. In practice, estimating the number of number of remaining defects is a major challange that needs further investigation.

Table 5 presents the results for DS3 which involves 796 test cases. The values of the parameters obtained can be compared with the values for DS2 presented in Table 4.

The coverage growth of different enumerables are plotted in Figure 3.

Figure 4 plots actual and model defect coverage values against branch coverage for DS3. It

Table 3: Summary table for DS1
(total 21,000 tests applied)

<table>
<thead>
<tr>
<th>Blocks i=1</th>
<th>Decisions i=2</th>
<th>c-uses i=3</th>
<th>p-uses i=4</th>
<th>Defects i=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total enum.</td>
<td>6977</td>
<td>3524</td>
<td>8851</td>
<td>4910</td>
</tr>
<tr>
<td>Final cov.</td>
<td>91.8%</td>
<td>83.9%</td>
<td>91.7%</td>
<td>73.5%</td>
</tr>
<tr>
<td>$b_0$</td>
<td>0.031</td>
<td>0.049</td>
<td>0.036</td>
<td>0.041</td>
</tr>
<tr>
<td>$b_1$</td>
<td>2E+8</td>
<td>1234</td>
<td>3.4E+6</td>
<td>2439</td>
</tr>
<tr>
<td>LSE</td>
<td>5.7E-4</td>
<td>3.5E-5</td>
<td>5.8E-4</td>
<td>8.1E-5</td>
</tr>
</tbody>
</table>

10
Table 4: Summary table for DS2

<table>
<thead>
<tr>
<th></th>
<th>Blocks</th>
<th>Branches</th>
<th>c-uses</th>
<th>p-uses</th>
<th>Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>89%</td>
<td>84%</td>
<td>76%</td>
<td>61%</td>
<td>90%</td>
</tr>
<tr>
<td>i=2</td>
<td>0.032</td>
<td>0.060</td>
<td>0.034</td>
<td>0.039</td>
<td>0.166</td>
</tr>
<tr>
<td>i=3</td>
<td>2E+8</td>
<td>870</td>
<td>3E+7</td>
<td>2500</td>
<td>0.11</td>
</tr>
<tr>
<td>i=4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSE</td>
<td>0.02</td>
<td>6.2E-4</td>
<td>3.5E-3</td>
<td>4.9E-3</td>
<td>0.025</td>
</tr>
</tbody>
</table>

Table 5: Summary table for DS3 (796 test cases)

<table>
<thead>
<tr>
<th></th>
<th>Blocks</th>
<th>Branches</th>
<th>C-uses</th>
<th>P-uses</th>
<th>Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=1</td>
<td>96%</td>
<td>94%</td>
<td>92%</td>
<td>85%</td>
<td>90%</td>
</tr>
<tr>
<td>i=2</td>
<td>0.07</td>
<td>0.074</td>
<td>0.044</td>
<td>0.079</td>
<td>0.139</td>
</tr>
<tr>
<td>i=3</td>
<td>2725</td>
<td>870</td>
<td>6.6E6</td>
<td>86</td>
<td>2.03</td>
</tr>
<tr>
<td>i=4</td>
<td>0.015</td>
<td>0.01</td>
<td>0.008</td>
<td>0.002</td>
<td>0.038</td>
</tr>
<tr>
<td>i=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSE</td>
<td>0.139</td>
<td>0.139</td>
<td>0.14</td>
<td>0.189</td>
<td></td>
</tr>
<tr>
<td>a_i</td>
<td></td>
<td>7E-4</td>
<td>2.4E-3</td>
<td>9E-7</td>
<td>0.042</td>
</tr>
<tr>
<td>a_i</td>
<td>14.13</td>
<td>13.14</td>
<td>21.46</td>
<td>9.88</td>
<td></td>
</tr>
<tr>
<td>LSE</td>
<td>0.023</td>
<td>0.014</td>
<td>0.04</td>
<td>0.023</td>
<td></td>
</tr>
</tbody>
</table>

Table 6 summarizes the result for DS4. Figure 5 illustrates 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 extent p-use coverage, are both strongly correlated with block coverage. The correlation with c-use coverage is weaker.
Figure 2: Actual and fitted values of defect coverage for DS2, DS3 and DS4

Figure 3: Coverage Growth of Different Enumerables (DS3)
Figure 4: Fault Coverage & Relative Defect Density (DS3)

Table 6: Summary table for DS4 (796 test cases)

<table>
<thead>
<tr>
<th>Final Coverage</th>
<th>Blocks</th>
<th>Branches</th>
<th>C-uses</th>
<th>P-uses</th>
<th>Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>94%</td>
<td>93%</td>
<td>94%</td>
<td>87%</td>
<td>90%</td>
</tr>
<tr>
<td>$b_0$</td>
<td>0.063</td>
<td>0.072</td>
<td>0.051</td>
<td>0.077</td>
<td>0.116</td>
</tr>
<tr>
<td>$b_1$</td>
<td>9759</td>
<td>1400</td>
<td>4.4E5</td>
<td>214</td>
<td>3.78</td>
</tr>
<tr>
<td>LSE</td>
<td>0.013</td>
<td>0.017</td>
<td>0.012</td>
<td>0.011</td>
<td>0.01</td>
</tr>
<tr>
<td>$a_0$</td>
<td>0.116</td>
<td>0.116</td>
<td>0.11</td>
<td>0.116</td>
<td></td>
</tr>
<tr>
<td>$a_1$</td>
<td>6E-4</td>
<td>3.8E-3</td>
<td>1E-5</td>
<td>0.017</td>
<td></td>
</tr>
<tr>
<td>$a_2$</td>
<td>15.23</td>
<td>13.4</td>
<td>19.20</td>
<td>12.95</td>
<td></td>
</tr>
<tr>
<td>LSE</td>
<td>0.022</td>
<td>0.022</td>
<td>0.04</td>
<td>0.01</td>
<td></td>
</tr>
</tbody>
</table>
Figure 5: Plot of C2, C3 and C4 against C1 (DS4)
7 Defect density and reliability

Here we consider the failure intensity during the operational period. We assume that debugging stops at a time $t_f$ and no further changes in the program are done. After time $t_f$, the defects remaining are not removed. Thus the failure intensity $\lambda$ no longer depends on time. Since the failure intensity is proportional to the number of defects $[mvs93]$, we have,

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

where $K$ is the overall value of fault exposure ratio. Musa et al. have found that the value of $K$ ranges between $1 \times 10^{-7}$ to $7.5 \times 10^{-7}$ failures/fault for several data sets examined $[mus87]$. The value of $K$ does not depend on the program size, but can depend on defect distribution in the program and the testing approach $[mvs93]$.

During testing and debugging, the faults found are removed. If we assume that no new faults are introduced during this process, the total number of defects to be found by $t_f$ can be computed as:

$$N^0(t_f) = N^0_0 (1 - C^0(t_f))$$

It should be noted that in actual practice debugging may be imperfect $[ohb89]$. Substituting $C^0$ using Equation 4,

$$N^0(t_f) = N^0_0 (1 - a'_0 \ln [1 + a'_1 (\exp(a'_2 C^i(t_f)) - 1)])$$

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

$$\frac{1}{\lambda(t_f)} = \frac{T_L}{K N_0 (1 - a'_0 \ln [1 + a'_1 (\exp(a'_2 C^i(t_f)) - 1)])}$$

Equation 6 can also be used for the operational period with the appropriate value for the fault exposure ratio. Notice that $K$ will depend on the operational profile encountered during the operational period $[mus99]$. 
8 Future Work

Further experimental and theoretical research is needed to validate the model proposed in this paper. Analysis of additional data sets will provide further insight into the problem. Here we have evaluated the values of the parameters $a_1^i$, $a_2^i$, $d_2^i$ by curve fitting. It will be useful to be able to obtain initial estimates of the parameter values using empirical methods. That would involve interpretation of the parameters for the logarithmic model [mvs93, mald97]. Estimation of the number remaining defects is another problem that needs further investigation.

9 Acknowledgement

The work by Y.K. Malaiya and N. Li was partly supported by a BMDO funded project monitored by ONR. J. Bieman’s work was supported in part by NSF, the NASA Langley Research Center, the Colorado Advanced Software Institute, Storage Technology Inc. and Micro-Motion Inc.

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

References


[ram85] J. Ramsey and V.R. Basili, “Analyzing the Test Process Using Structural Cover-
age”, Proc. 8th Int. Conf. on Software Engineering, August 1985, pp. 306-312.

[rap85] S. Rapps and E.J. Weyuker, “Selecting Software Test Data Using Data Flow In-

[vou92] M.A. Vouk “Using Reliability Models During Testing With Non-operational Pro-
files,” Proc. 2nd Bellcore/Purdue workshop on issues in Software Reliability Esti-