Applying Code Coverage Approach to an Infinite Failure Software Reliability Model

Adalberto Nobiatu Crespo
CenPRA/MCT – Campinas - SP – Brazil
adalberto.crespo@cti.gov.br

Mario Jino
FEEC/Unicamp – Campinas - SP – Brazil
jino@dca.fee.unicamp.br

Alberto Pasquini
DeepBlue - Rome - Italy
alberto.pasquini@dblue.it

José Carlos Maldonado
ICMC/USP - São Carlos - SP - Brazil
jcmaldon@icmc.usp.br

Resumo – Uma abordagem para a modelagem de confiabilidade de software baseada em cobertura de código é usada para derivar o Modelo de Falhas Infinitas de confiabilidade de software Baseado em Cobertura de Código – MFIBC. Nosso objetivo foi verificar a robustez da abordagem sob diferentes suposições. O MFIBC foi avaliado com dados de teste de uma aplicação real, fazendo uso dos seguintes critérios estruturais de teste: todos-os-nós, todos-os-ramos, e potenciais-usos – uma família de critérios baseados em fluxo de dados. O MFIBC mostrou-se ser tão bom quanto o Modelo Geométrico – GEO, o modelo tradicional baseado em tempo que melhor se ajustou aos dados. Resultados da análise também mostram que o MFIBC é tão bom quanto o MBBC – Modelo Binomial de confiabilidade de software Baseado em Cobertura – um modelo derivado anteriormente usando a abordagem de cobertura de código, indicando que esta é efetiva sob diferentes suposições de modelagem.

Palavras chave: modelos de confiabilidade de software, critérios estruturais de teste, cobertura de teste.

Abstract – An approach to software reliability modeling based on code coverage is used to derive the Infinite Failure Software Reliability Model Based on Code Coverage – IFMBC. Our aim was to verify the soundness of the approach under different assumptions. The IFMBC was assessed with test data from a real application, making use of the following structural testing criteria: all-nodes, all-edges, and potential-uses – a data-flow based family of criteria. The IFMBC was shown to be as good as the Geometric Model – GEO, found to be the best traditional time-based model that fits the data. Results from the analysis also show that the IFMBC is as good as the BMBC – Binomial software reliability Model Based on Coverage – a model previously derived using the code coverage approach, indicating it to be effective under different modeling assumptions.

Keywords: software reliability models, structural testing criteria, test coverage.

I. INTRODUCTION

A. Context

Reliability is defined as the probability that the software won't fail over an interval of time, in a given environment [1]. If true operational reliability of a software product can be estimated during its production testing, it is a measure very relevant in making a decision on the release of software. Reliability has been extensively considered in the analysis of software quality and its study has attracted a significant attention from the scientific community. However, estimating the operational reliability of software, particularly while it is being developed, is a very challenging task. In practice, reliability represents quality from the user's point of view; it is known that it is practically impossible to achieve 100% reliability, even for programs which are not very complex. Software reliability may be used as a reference by software development organizations; the level of the measured reliability of a product would be a criterion for its approval. In critical applications tests are extensively performed not only to remove faults from a product but also to determine its reliability level.

Concern about software reliability has been around for a long time [2]. Some of the first studies and models of software reliability were proposed by Jelinski and Moranda [3] and Shooman [4]. A good historical perspective and survey is presented by Lyu [5].

B. Motivation for Using the Code Coverage Approach

Mathematical models of software reliability have a probabilistic nature and attempt to specify the probability of occurrence of software failures. The ultimate goal of the models is to quantify the software reliability as precisely as possible.

Models are used to measure reliability, to analyze failure data, to make inferences about the future behavior of the software, and for decision making during the testing and debugging processes. Several authors [6], [7], [8], [9], [10] and [11] have pointed out limitations of reliability models and reasons for their insufficient predictive ability. The usual procedure for software reliability modeling does not make use of information on the coverage of required
elements of none of the well-known “white box” testing criteria, despite the occurrence of failures being always connected to the exercise of a program’s elements during its execution. Overestimation of reliability occurs in measurements of time domain based reliability models when a test strategy (criterion) continues to be used after it has reached its limit [12]. As the number of unexercised required elements decreases, the time interval between failures increases. After the limit of the test criterion is reached, this effect is even more pronounced with a considerable increase in the time between failures. Reliability estimates produced by time domain based models grow without defect removal. Experiments show that different test strategies yield different estimates of reliability; that is, the testing strategy affects the performance of those models [13]. Another restriction to the use of time as a control variable is that there is no guarantee that the test effort is adequate if at least the main functions of the software are not exercised.

Several experiments and studies have been conducted to investigate the relationship between code coverage and software reliability [14], [15], [16], [17], [18], [10], [19], [20], [11], [21], [22], [23], [24], [25], [26], [27], [28], [29]. An early experiment using control flow and data flow based testing was conducted by Frate, Mathur and Pasquini [30]. Their results provide evidence of a relationship between software reliability and coverage of elements required by the testing criteria. Another experiment, by Crespo, Jino, Pasquini, and Maldonado [31], investigates the relationships between software reliability and coverage of elements required by the following testing criteria: all-nodes, all-arc, and the data flow based family of criteria – potential-uses, all-potential-uses, all-potential-uses/du, and all-potential-du-paths. Our approach has been used to derive the BMBC – Binomial software reliability Model Based on Coverage [32], [33] – whose results provided motivation to investigate the effectiveness of the code coverage approach under different modeling assumptions.

The approach assumes that execution of a test datum corresponds to a unit of software execution. Cai [34] adopts a similar assumption in an approach for software reliability modeling in discrete-time domain. The information on the coverage is used directly in the process of software reliability modeling.

Malaya et al. [17] make the same assumption to create a model establishing a relationship between code coverage and number of test data. Their model explains the coverage as a function of test data. Chen [12] also makes that assumption when using information on coverage to define a compression factor to correct the overestimation of reliability by the traditional reliability models.

C. Organization of the Paper

Section 2 presents basic concepts of testing criteria and of coverage of elements required by a testing criterion. Section 3 contains the development of the infinite failures category model based on coverage and the measures of reliability. Section 4 describes the experiment carried out to obtain the data used to evaluate the model, using real application software; the results of application of the model on these data are also shown. Section 5 presents a brief description of related work. Section 6 presents our conclusions and suggestions for future work.

II. TESTING CRITERIA AND CODE COVERAGE

A. A General View of Testing Criteria

In the structural test input data are selected to exercise the elements of the implementation of the software, that is, test requirements are derived from information associated to the control and data structures of the procedural design.

Structural testing methods or criteria may be classified as: Control Flow Based Testing – test requirements are based on the constructs of the control flow of a program code. Examples of control flow testing criteria are: statement testing (or node testing), branch testing (or arc testing), condition testing, and multiple-condition testing.

Data Flow Based Testing – test requirements are based on the types of occurrence of variables in a program code, basically, definitions and uses of variables. Examples of data flow based testing criteria are: all-uses, all-c-uses and all-p-uses [35]; all-potential-uses (PU), all-potential-uses/du (PUDU), all-potential-du-paths (PDU) [36].

Each of those testing criteria can be used either to evaluate or to select test data sets.

B. Coverage of Elements Required by Testing Criteria

Test coverage is always associated to a testing criterion \( C_j \). A coverage value, \( c_j \), expresses the percentage of already exercised required elements with respect to the total number of elements required by the testing criterion \( C_j \). A coverage of 100% (\( c_j = 1 \)) indicates that all the elements required by the criterion \( C_j \) were exercised by execution of a test data set. However, most of the real programs do have infeasible elements (an infeasible element can never be exercised, that is, there are no input data that make an infeasible element to be exercised), meaning that 100% coverage is usually not attainable [37].

Functional testing is one of the most common ways of software evaluation. A certain amount of input data is selected, the software is executed on these data, and a check is made whether the produced outputs match the expected results. The internal structure of the software is not considered in this process. The confidence level on the product reliability increases when a large number of input data executed by the software yields correct results.

Functional testing, although still necessary for validation, is not enough to assure that the product has been adequately tested, as it does not check the constructs of a software implementation; it does not provide information on what is or is not being exercised in the code.

While functional testing focuses on covering externally visible functions, interfaces and parameters, code coverage-
based testing measures the extent a test data set exercises the several regions and elements of all internal structures of the code. That is, the coverage is a metric used to assess the adequacy of a test data set concerning the appropriate degree of code exercising. If there is a growth in coverage, that is, a new test datum exercises an area or element of the code still not exercised, the likelihood increases of revealing new faults.

Therefore, coverage information provides a concrete basis for the use of the testing activity to measure appropriately software reliability.

III. AN INFINITE FAILURE MODEL BASED ON COVERAGE – IFMBC

A. Foundations

The failure process of the software for an infinite failure model is characterized by the behavior of the mean value function, \( \mu(t) \), of the process. According to Musa and Okumoto’s classification scheme [1], \( \lim_{t \to \infty} u(t) = \infty \) for this category of software reliability models. It means that the software is never completely free of faults. This could be caused by additional faults being introduced in the software by the defect correction process.

Time is replaced by the variable number of test data with the assumption that the execution of a test datum is equivalent to a unit of time of execution of the software. That is, the measurement unit of software testing is the test datum. Thus, the number of failures characterizes the failure rate of software by test datum or by test data set.

However, a number of test data executed by the software results in a measured coverage percentage of the software code. Moreover, the coverage percentage depends on the testing criterion used for evaluation of the test data. Hence, the failure rate of the software is related to the coverage of the testing criterion achieved by execution of the test data.

It should also be remarked that in the initial stages of testing the failure rate is high and the test coverage is low as few test data have been executed; in the final stages of testing, the failure rate is low and test coverage is usually high. That is, the complement of the measured test coverage \( (1 - c_i) \) can be used as a variable directly related to the failure rate.

Hence, the basic assumption of the proposed infinite failure model is that the failure rate of the software is directly proportional to the complement of the measured coverage achieved by execution of the test data.

For the same set of test data the software reliability estimated from measured code coverage must be the same independently of the testing criterion used to measure the code coverage. Distinct criteria yield different values of code coverage for the same test data. Therefore, normalized coverage will be used instead of the measured coverage with the purpose of estimating the software reliability independently of testing criteria. The estimated software reliability and the normalized coverage for the same test data are made the same across all testing criteria to estimate the parameters of the proposed software reliability model for each testing criterion.

B. Development of the Model

The infinite failure model is based on the following assumptions:
1. There is an infinite number of faults in the software;
2. All faults do not have the same probability of being detected;
3. Faults are detected independently;
4. The software is tested under conditions similar to those of its operational profile; and
5. The test data are executed and the coverage of the elements required by the selection criterion used in test data evaluation is calculated for each failure occurrence.

Assumption 1 means that the defect removal process isn’t perfect, i.e., the software is never free of faults. Assumption 2 means that the faults have different probabilities of being detected, i.e., as faults are removed the probability of detection changes. By Assumption 3 the joint probability density function of the maximum likelihood method can be determined by multiplying the density functions of each of the random variables. Assumption 4 assures that the model estimates obtained with data collected in the test environment are valid in the operational environment of the software. Assumption 5 is a characteristic of the testing procedure and indicates that the coverage should be measured at each failure occurrence. From this assumption, the failure rate of the software is constant between failures and decreases for each defect removed, as a function of the coverage reached in the code. We adopt the following functional form for the failure rate:

\[
Z(n) = D(\phi_i)^i
\]

D is the initial failure rate and \( \phi_i \) \( (0 \leq \phi_i \leq 1) \) is the complement of the coverage reached after removal of the \( i \)th defect, detected after applying \( n_i \) test data, for \( n_i < n \leq n_{i+1} \). The letter \( i \) stands for the order of the removed fault, i.e., \( i = 0, 1, 2, 3... \)

Notice that this model is an extension to the Geometric model [1], where \( \phi_i \) is made constant (\( \phi \)) for all intervals between failures. Notice also that the assumptions and the functional form of the failure rate are different from those of the model previously developed with the same approach – BMBBC [33].

The value of \( \phi_i \) depends on the strength of the criterion used for selection of the data used in the test. The strength of a criterion is associated to the level of difficulty in achieving the coverage of its required elements. It is a qualitative measure that establishes a transitive relationship among criteria; stronger criteria demand greater testing effort than that required by weaker criteria to have their required elements covered. Since the coverage reached by the test data is associated to the criterion, its value is related...
to the strength of the selection criterion used in the test. When the maximum measured coverage is reached the selection criterion used in the test reaches its saturation point; from this point on, it is not possible to assess the quality of an additional test datum, using this criterion.

Figure 1 illustrates the software testing process.

![Figure 1: Software testing process](image)

Observe that:

\[ n_0 = 0; \quad n_1 = x_0; \quad n_2 = n_1 + x_1; \quad \ldots; \quad n_{i+1} = n_i + x_i \quad \text{and} \quad \phi_0 = 1. \]

The failure rate function has the following interpretation:

Consider the weighting factors of a selection criterion considered weak and a selection criterion considered strong. The test data generated to satisfy a stronger selection criterion has a greater chance than those of a weaker one of revealing defects in the software as they exercise a larger number of distinct paths of the program. This means that, when a stronger criterion is adopted, the chance of revealing a larger number of defects increases. That is, the failure rate will decrease as the coverage increases probably more slowly for a stronger criterion than for a weaker criterion. Hence, the weighting factor of the criterion (the criterion strength) influences the scale of the failure rate of the software.

Therefore, the normalized coverage will be used instead of the observed coverage to standardize the estimates generated by the model, independently of any criterion. From now on, \( \phi_i \) is used as the normalized complement of coverage, that is, \( \phi_i = g(1 - c_i) \) where \( c_i \) represents the coverage observed in the test, after the detection of the \( i \)-th failure and \( g(\cdot) \) is an increasing function of the coverage.

In this model, the complement of coverage is normalized by adopting the linear transformation \( \hat{\phi}_i = \alpha_0 + \alpha_1 (1 - c_i) \). Other transformations could be used according to the assumptions and/or to the data obtained from testing. The parameters \( \alpha_0 \) and \( \alpha_1 \) can be estimated by the maximum likelihood method, according to Assumption 3. Observe that the complement of coverage as well as the normalized complement of coverage are decreasing functions of test data, subjected to the constraint that \( 0 \leq \alpha_0 + \alpha_1 (1 - c_i) \leq 1 \), that is, \( 0 \leq \phi_i \leq 1 \).

The adopted failure rate remains constant between consecutive failures. Thus, the random variable \( X_i \), “number of test data between failures”, has a probability distribution function that can be approximated to an exponential distribution with average:

\[ \mathbb{E}(X_i) = \frac{1}{D(\phi_i)^i} \]

Hence, the number of data test between failures is represented by an exponential distribution whose average changes for each removed defect.

For the infinite category software reliability model, Musa [1] suggests the following approximations:

\[ \lambda(n) \approx Z(n) = D(\phi_i)^i \]

for \( n > n_i \). That is, the mean value function of the random variable that represents the accumulated failures, \( \mu(n) \), is approximately the number of failures in the test, \( i \); the failure intensity function, \( \lambda(n) \), is approximately the software rate failure, \( Z(n) \).

From these approximations it can be shown [38] that the failure intensity function has the following form:

\[ \lambda(n) = \frac{D}{D \theta_i \exp(\theta_i)n + 1} \]  \hspace{1cm} (2)

where \( \phi_i = \exp(-\theta_i) \).

C. Measures of Reliability

The quantitative measures of software reliability based on an infinite category failure model of reliability growth are defined next.

a) Reliability Function of the Software

Musa [1] shows that for an infinite category failure model the reliability function may be obtained as:

\[ R(n) = \exp\left[-\int_0^n Z(n) \, dn\right] \]

From Equation 1, it follows that:

\[ R(n) = \exp\left[-D(\phi_i)^i \int_0^n dx\right] \]

Therefore, software reliability is defined as:

\[ R(n) = \exp\left[-D(\phi_i)^i n\right], \quad \text{for} \quad n > n_i \]

b) Degree of Software Purification

It is not possible to estimate the number of remaining faults in the software at a certain point of test for the infinite category failure models. However, the degree of purification, \( P \), for the software can be estimated. \( P \) provides an estimate for the quality of software testing ( \( 0 \leq P \leq 1 \) ).

The degree of purification for software is obtained as:

\[ P = \frac{Z(n_0)}{Z(n_i)} \]

Thus, using Equation 1 the degree of purification is given by:

\[ P = \frac{D - D(\phi_i)^i}{D} = 1 - (\phi_i)^i \]  \hspace{1cm} (3)

c) Mean number of Test Data To next Failure - MTCTF

As shown before, the mean number of test data to next failure – MTCTF is given by:
D. Estimation of the Parameters of the Model

The parameters of the proposed model can be estimated by the maximum likelihood method. From Equation 1, the probability density function of random variable X has the following form [38]:

\[ f(x_i) = D(\phi_i)^{\alpha_0 + \alpha_1(1-c_i)} \]

Using the relationship between observed coverage and normalized complement of coverage ( \( \phi_i = \alpha_0 + \alpha_1(1-c_i) \) ), \( f(x_i) \) can be expressed as:

\[ f(x_i) = D[\alpha_0 + \alpha_1(1-c_i)]^{\alpha_0 + \alpha_1(1-c_i)} \]

From Assumption 3 – independence of faults, the likelihood function after k faults removed is defined as:

\[ L(X_0, X_1, ..., X_k; D, \alpha_0, \alpha_1) = f(x_0)f(x_1)...f(x_k) = D^k \prod_{i=0}^{k-1} [\alpha_0 + \alpha_1(1-c_i)]^{\alpha_0 + \alpha_1(1-c_i)} x_i \]

\[ (4) \]

Taking the logarithm of Equation 4:

\[ \ln L = k \ln(D) + \sum_{i=0}^{k-1} \ln[\alpha_0 + \alpha_1(1-c_i)] - D \sum_{i=0}^{k-1} [\alpha_0 + \alpha_1(1-c_i)] x_i \]

Making \( \frac{\partial \ln L}{\partial D} = 0 \), \( \frac{\partial \ln L}{\partial \alpha_0} = 0 \) and \( \frac{\partial \ln L}{\partial \alpha_1} = 0 \), we obtain the following non-linear system of equations with respect to parameters D, \( \alpha_0 \) and \( \alpha_1 \):

\[ \begin{aligned}
D &= \frac{1}{\prod_{i=0}^{k-1} [\alpha_0 + \alpha_1(1-c_i)] x_i} \\
\sum_{i=0}^{k-1} \alpha_0 + \alpha_1(1-c_i) &= D \sum_{i=0}^{k-1} [\alpha_0 + \alpha_1(1-c_i)] x_i \\
\sum_{i=0}^{k-1} (1-c_i) &= D \sum_{i=0}^{k-1} [\alpha_0 + \alpha_1(1-c_i)] (1-c_i) x_i \\
\end{aligned} \]

(5)

To obtain the estimates of D, \( \alpha_0 \) and \( \alpha_1 \), it is necessary to solve the non-linear system (5). The values of D, \( \alpha_0 \) and \( \alpha_1 \) that satisfy simultaneously the system (5) are the maximum likelihood estimates of the parameters D, \( \alpha_0 \) and \( \alpha_1 \). The solution can be found through numerical procedures. An alternative way of obtaining the estimates of the parameters is to use optimizing routines to maximize the likelihood function (4), simultaneously as a function of D, \( \alpha_0 \) and \( \alpha_1 \).
**Measured Reliability**

Nelson’s method [41] was used to estimate the values of reliability, to establish the relationship of software reliability to other variables.

For each occurrence of a failure and after the removal of the corresponding fault, the software is executed with thousands of randomly generated test data; test data were generated using the same operational profile of the application of the model. The ratio is calculated between “n_e”, the number of executions with failure, and “n”, the total number of executions of the software. This ratio is an estimate of the probability of occurrence of a failure in the software. The reliability of the software is then estimated as:

\[ R = 1 - \frac{n_e}{n} \]

For each fault removed from the software, the method is applied again to recalculate the reliability. Thus, an estimate of the behavior of the reliability growth of the software is obtained as a function of fault removal. Notice that reliability is always calculated after the removal of a fault.

Table 1 shows the measured reliability after removal of each fault obtained from the total number of executions “n” and the number of executions with failure “n_e”. It also shows removed faults; test data accumulated up to removal of faults; and gives a general picture of the coverage achieved for each of the testing criteria.

<table>
<thead>
<tr>
<th>Removed Faults</th>
<th>Test Data Between Failures - x</th>
<th>Accumulated Test Data - n</th>
<th>COVERAGE OF TESTING CRITERIA</th>
<th>Number of Executions of the Software - n</th>
<th>Number of Executions with Failure - n_e</th>
<th>Measured Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NODES</td>
<td>ARCS</td>
<td>PU</td>
<td>PUDU</td>
<td>PDU</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4937</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.3068</td>
<td>0.2130</td>
<td>0.1671</td>
<td>1518</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0.3377</td>
<td>0.2234</td>
<td>0.1701</td>
<td>1591</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0.3579</td>
<td>0.2358</td>
<td>0.1800</td>
<td>1622</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0.3893</td>
<td>0.2486</td>
<td>0.1876</td>
<td>1671</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>0.3993</td>
<td>0.2540</td>
<td>0.1952</td>
<td>1736</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6</td>
<td>0.4013</td>
<td>0.2601</td>
<td>0.1969</td>
<td>1791</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
<td>0.4174</td>
<td>0.2739</td>
<td>0.1991</td>
<td>1852</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>8</td>
<td>0.4213</td>
<td>0.2811</td>
<td>0.2068</td>
<td>1998</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>9</td>
<td>0.4457</td>
<td>0.2884</td>
<td>0.2128</td>
<td>2034</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>11</td>
<td>0.4674</td>
<td>0.3016</td>
<td>0.2317</td>
<td>2199</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>12</td>
<td>0.4740</td>
<td>0.3241</td>
<td>0.2403</td>
<td>2218</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>13</td>
<td>0.4796</td>
<td>0.3287</td>
<td>0.2435</td>
<td>2246</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>14</td>
<td>0.4817</td>
<td>0.3311</td>
<td>0.2456</td>
<td>2268</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>15</td>
<td>0.4849</td>
<td>0.3333</td>
<td>0.2479</td>
<td>2292</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>17</td>
<td>0.4901</td>
<td>0.3398</td>
<td>0.2491</td>
<td>2316</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>18</td>
<td>0.4977</td>
<td>0.3413</td>
<td>0.2521</td>
<td>2369</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>23</td>
<td>0.5013</td>
<td>0.3499</td>
<td>0.2599</td>
<td>2415</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>24</td>
<td>0.5089</td>
<td>0.3505</td>
<td>0.2627</td>
<td>2454</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>25</td>
<td>0.5123</td>
<td>0.3679</td>
<td>0.2699</td>
<td>2567</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>26</td>
<td>0.5269</td>
<td>0.3719</td>
<td>0.2719</td>
<td>2692</td>
</tr>
<tr>
<td>21</td>
<td>6</td>
<td>32</td>
<td>0.5384</td>
<td>0.3836</td>
<td>0.2857</td>
<td>2854</td>
</tr>
<tr>
<td>22</td>
<td>39</td>
<td>71</td>
<td>0.5747</td>
<td>0.4233</td>
<td>0.3183</td>
<td>2953</td>
</tr>
<tr>
<td>23</td>
<td>20</td>
<td>91</td>
<td>0.6404</td>
<td>0.4775</td>
<td>0.3268</td>
<td>3035</td>
</tr>
<tr>
<td>24</td>
<td>35</td>
<td>126</td>
<td>0.6808</td>
<td>0.5159</td>
<td>0.3577</td>
<td>3294</td>
</tr>
<tr>
<td>25</td>
<td>60</td>
<td>186</td>
<td>0.6836</td>
<td>0.5198</td>
<td>0.4099</td>
<td>3645</td>
</tr>
<tr>
<td>26</td>
<td>253</td>
<td>439</td>
<td>0.7158</td>
<td>0.5807</td>
<td>0.4116</td>
<td>4113</td>
</tr>
<tr>
<td>27</td>
<td>400</td>
<td>839</td>
<td>0.7158</td>
<td>0.5807</td>
<td>0.4542</td>
<td>4116</td>
</tr>
<tr>
<td>28</td>
<td>401</td>
<td>1240</td>
<td>0.7178</td>
<td>0.5847</td>
<td>0.4621</td>
<td>4156</td>
</tr>
</tbody>
</table>

Legend: PU – all-potential-uses; PUDU – all-potential-uses/du; PDU – all-potential-du-paths
B. Application of the Infinite Failures software reliability Model Based on Coverage – IFMBC

Data from Table 1 were used to analyze traditional time-based software reliability models and the IFMBC.

In a previous study [32], [33] the SMERFS tool [42] was used to select the best among the traditional reliability models implemented in it. SMERFS was used to do a preliminary filtering of the models. Selected models were analyzed using the goodness-of-fit criteria, the Distance of Kolmogorov and the $\chi^2$ statistics. The Geometric Model – GEO, was found to be the one that best fits the data; it is the traditional time-based model chosen for comparison to the proposed model.

The reliability estimates generated by the proposed model – IFMBC - are graphically and statistically compared to those generated by the Geometric model and to the measured reliability values from Table 1.

C. Results of the Infinite Failures Category Model Based on Coverage - IFMBC

The parameters of the model were estimated using an optimization routine of the software Matlab, applied to the likelihood function of Equation 4.

Table 2 shows the results of the estimation of the parameters of the model by using the data from Table 1, for each of the testing criteria.

Table 3 shows the measured reliability and reliability estimated by IFMBC as a function of test data and of removed faults, for the testing criteria: Nodes, Arcs, PU, PUDU, and PDU.

Figure 2 illustrates graphically the behavior of the measured reliability and of the reliability estimated by the IFMBC, for each of the testing criteria. Observe that the values of the reliability estimated by IFMBC for the different testing criteria are practically equal for the same data, as they should be. The values of reliability estimated by the model are approximately the same for all of the testing criteria. Visually, the values of the measured reliability and of the estimated reliability are very close indicating that the model is well fitted to the data.

Figure 3 illustrates graphically the measured reliability and the reliability estimated by the IFMBC and by the Geometric Model – GEO; we use the reliability estimated for criterion all-nodes. IFMBC has an adjustment closer than GEO to the measured reliability. Table 4 consolidates the results from the Kolmogorov Smirnov’s test applied to the data of the measured reliability and estimated reliability of the IFMBC for the criterion all-nodes, and to the data of the measured reliability and estimated reliability of the GEO. In a sample of size 28 with a significance level of $\alpha = 0.01$, the bilateral hypothesis $H_0$ of equality between the measured reliability and the reliability estimated by the model is accepted for a value of statistical KD below 13 [43]. From Table 4, the hypothesis of equality is accepted for both models. However, for the maximum distances (Dmax), the smaller happens for the IFMBC, showing that its results are closer to the measured reliability than those of the geometric model.

IFMBC can also be applied to help making the decision on stopping testing activities. After each new defect is detected and removed, the model can be used to estimate the reliability level, which can then be the basis for the decision to stop or to continue testing.

For example, Table 5 illustrates the results when the IFMBC is applied to the data of Table 1 up to the 25th removed fault, for the criterion all-nodes. The estimated software reliability is $R_e = 0.9750$, and the degree of purification is $P = 0.9973$ (Equation 3). At this point, based on the estimated reliability or degree of purification, one can decide either to stop or to continue the test (in case it is desired to reach a higher level of reliability).

From Table 1, with the removal of the 25th defect, the measured reliability $R_m$ reaches 0.9907, a value close to that of $R_e$. IFMBC underestimates the value of software reliability, an important property for a software reliability model. Most of the traditional models overestimate the reliability of the software, a much-criticized characteristic. The mean number of test data to next failure – MTCFT, was estimated to be 50 by the model; from Table 1, the 26th failure happened with the 253rd test data, again an underestimation of the reliability.

D. Threats to Validity

There are threats to the validity of the results of our experiment. First of all, it is a one-program experiment in spite of being a real application software. Another threat is the definition of a single operational profile; in practice, software may be used in several ways, defined by several operational profiles.

The faults reinserted in the software, although being real ones, may not be independent of each other, contrarily to the assumption of the model. Another threat concerns the assumption that all test data are equivalent when we make the conversion of time to test data in the model; different test data may exercise different parts of the program and demand different amounts of time to be executed by the program.
Figure 2. Measured Reliability and Reliability Estimated by the IFMBC, for each Testing Criteria.

Table 3: Estimated and Measured Reliability as a Function of Test Data and Removed Faults for Testing Criteria

<table>
<thead>
<tr>
<th>Removed Faults</th>
<th>Accumulated Test Data</th>
<th>Reliability Estimated by IFMBC</th>
<th>Measured Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Arcs</td>
<td>PU</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.0623</td>
<td>0.0575</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.0684</td>
<td>0.0695</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.0807</td>
<td>0.0849</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.0977</td>
<td>0.1052</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0.1275</td>
<td>0.1314</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0.1551</td>
<td>0.1588</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0.1807</td>
<td>0.1910</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0.2253</td>
<td>0.2363</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>0.2613</td>
<td>0.2802</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.3346</td>
<td>0.3283</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>0.4141</td>
<td>0.3906</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>0.4690</td>
<td>0.4750</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>0.5224</td>
<td>0.5270</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>0.5673</td>
<td>0.5730</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>0.6127</td>
<td>0.6166</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>0.6601</td>
<td>0.6666</td>
</tr>
<tr>
<td>17</td>
<td>23</td>
<td>0.7094</td>
<td>0.7033</td>
</tr>
<tr>
<td>18</td>
<td>24</td>
<td>0.7472</td>
<td>0.7506</td>
</tr>
<tr>
<td>19</td>
<td>25</td>
<td>0.7888</td>
<td>0.7795</td>
</tr>
<tr>
<td>20</td>
<td>26</td>
<td>0.8188</td>
<td>0.8319</td>
</tr>
<tr>
<td>21</td>
<td>32</td>
<td>0.8617</td>
<td>0.8582</td>
</tr>
<tr>
<td>22</td>
<td>71</td>
<td>0.8933</td>
<td>0.8900</td>
</tr>
<tr>
<td>23</td>
<td>91</td>
<td>0.9394</td>
<td>0.9384</td>
</tr>
<tr>
<td>24</td>
<td>126</td>
<td>0.9778</td>
<td>0.9728</td>
</tr>
<tr>
<td>25</td>
<td>186</td>
<td>0.9897</td>
<td>0.9865</td>
</tr>
<tr>
<td>26</td>
<td>439</td>
<td>0.9922</td>
<td>0.9897</td>
</tr>
<tr>
<td>27</td>
<td>839</td>
<td>0.9962</td>
<td>0.9966</td>
</tr>
<tr>
<td>28</td>
<td>1240</td>
<td>0.9971</td>
<td>0.9974</td>
</tr>
</tbody>
</table>

Legend: PU – all-potential-uses; PUDU – all-potential-uses/du; PDU – all-potential-du-paths

Table 4: Bilateral Test of Kolmogorov-Smirnov

(α = 0.01)

<table>
<thead>
<tr>
<th>Statistics</th>
<th>IFMBC</th>
<th>GEO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dmax</td>
<td>0.21059207</td>
<td>0.30139797</td>
</tr>
<tr>
<td>KD Statistics</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Decision</td>
<td>Accept Hₐ</td>
<td>Accept Hₐ</td>
</tr>
</tbody>
</table>

Distance of Kolmogorov-Smirnov Between Measured Reliability and Reliability Estimated by the Models

223
There are several studies dealing with the relationship between software reliability and software testing code coverage. Malaiya et al. [17] and Malaiya et al. [26] model the relations among testing time, code coverage, and software reliability. They present an LE (logarithmic-exponential) model that relates testing effort to test coverage (block, branch, computation-use, or predicate-use).

Chen et al. [11], [24] state that: “Existing software reliability growth models often over-estimate the reliability of a given program. Empirical studies suggest that the overestimations exist because the models do not account for the nature of the testing.”. They present a technique intended to solve this problem, using both time and code coverage measures for the prediction of software failures in operation.

Frate et al. [30] conducted experiments to investigate the correlation between code coverage and software reliability. Reliability was estimated to be the probability of no failure over the given input domain defined by an operational profile. It was observed that an increase in reliability is accompanied by an increase in at least one code coverage measure.

Krishnamurthy and Mathur [22] conducted a pilot experiment to investigate the accuracy of reliability estimates obtained from code coverage. The code coverage was measured using random testing. The results indicate that the degree of correlation between coverage and reliability has an inverse relationship with the fault density, and a direct relationship with program size.

None of the above studies use directly the information on code coverage to model software reliability as it is done here and in our previous studies [32], [33].

VI. CONCLUSIONS AND FUTURE WORK

The approach of using the information on code coverage for modeling software reliability was used to derive a new model of software reliability.

The Infinite Failure Category Model Based on Coverage – IFMBC, is a model based on the coverage of elements required by structural testing criteria, where information on the coverage is used as a parameter in the formulation of the model.

Data on software failures from a real application were applied in the model. Failure data were obtained of the application of the testing criteria: all-nodes, all-arcs, all-potential-uses, all-potential-uses/du, and all-potential-du-paths. For the particular application software, the results have shown a clear superiority of the model IFMBC – Infinite Failure Category Model Based on Coverage over the Geometric model.

Results from application of IFMBC are as good as those obtained with the BMBC, which was also compared to the GEO – the traditional time-based model that best fits the data [33]; the overestimation effect of the traditional software reliability models does not happen for both IFMBC and BMBC.

The IFMBC is a model derived under the same approach of the BMBC [32], [33], using data from the same experiment. These models provide evidence that this is a very promising research direction in software reliability.

Additional experiments are being conducted to confirm the obtained results; in particular, experiments with different operational profiles of the same software are being carried out.

Other types of information related to software reliability should be investigated and other reliability models should be proposed and tested. Models should be evaluated with failure data from software with diverse characteristics; other testing criteria should also be considered.

REFERENCES


Table 5. Estimated and Measured Reliability and MTCTF

<table>
<thead>
<tr>
<th></th>
<th>Estimated by IFMBC - R_e</th>
<th>Measured - R_m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>0.9750</td>
<td>0.9907</td>
</tr>
<tr>
<td>MTCTF</td>
<td>50</td>
<td>253</td>
</tr>
<tr>
<td>Degree of Purification</td>
<td>0.9973</td>
<td>--</td>
</tr>
</tbody>
</table>

Legend: MTCTF- Mean Number of Test Data to Next Failure

Figure 3. Measured Reliability and Reliability Estimated by IFMBC and Geometric Model


