Two Dimensional Flexible Software Reliability Growth Model with Two Types of Imperfect Debugging

P.K. Kapur¹, Anu G. Aggarwal² and Abhishek Tandon³
Department of Operational Research, University of Delhi, Delhi-110007
¹pkkapur1@gmail.com, ²anuagg17@gmail.com and ³abhishek.tandon@rocketmail.com

ABSTRACT
Software reliability is the probability of failure free operation of software in a given period of time under specified conditions. Software testing can be defined as a process to detect faults in the totality and worth of developed computer software. Testing is very important in assuring the quality of the software by identifying faults in software, and possibly removing them. But testing of the software for a long time may not ensure a bug free software and high reliability. Optimum amount of code also needs to be covered to make sure that the software is of good quality. Testing time alone may not give the correct picture of the number of faults removed in the software. Therefore to capture the combined effect of testing time and testing coverage we propose a two dimensional software reliability growth model. We have used cobb-douglas production function to develop the two dimensional model incorporating the effect of testing time and testing coverage on the number of faults removed in the software system. The faults in the software may not be removed perfectly; this is mainly due to complexity of software or nature of testing team. This phenomenon is known as imperfect debugging. When the faults are not removed perfectly and leads to further generation of faults this process is known as error generation. In this paper, we develop an s shaped model with imperfect debugging and fault generation. The proposed model is validated on real data sets.

KEYWORDS
Software Reliability, Two dimensional, Non-Homogeneous Poisson Process, Testing Coverage (TC), Imperfect Debugging

1. INTRODUCTION
The concern for software reliability has grown over a period of time especially with the advent of real life systems such as satellite and shuttle control, telephone, internet and banking services. With the growing economy there has been a growing interest of companies to know about their competitors and have been spending a lot on strategic decision making. All these activities require complex software systems. It is important that these systems are thoroughly tested before implementation. There is a huge cost attached with fixing failures, safety concerns, and legal liabilities therefore organizations need to produce software that is reliable. There are several methodologies to develop software but questions that need to be addressed are how many times will the software fail and when, how to estimate testing coverage, when to stop testing, and when to release the software. Also, for a software product we need to predict/estimate the maintenance effort; for example, how long must the warranty period must be, once the software is released, how many defects can be expected at what severity levels, how many engineers are required to support the product, for how long, and so forth. The prospective disastrous failure of software and the consequential damage is a scenario that is of great concern in terms of scientific disaster preparedness. Proper vigilance for such an event to enable mitigation of adverse impacts to the greatest degree possible requires modeling of software reliability to capture the risk of software failure. Software failures may be due to errors, ambiguities, oversights or misinterpretation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems. These errors can hamper the performance of the software and can adversely affect its reliability. Therefore software reliability is of great concern for software developers.

Reliability techniques can be divided into two categories namely Trending and Predictive. Trending reliability tracks the failure data produced by the software system to develop a reliability operational profile of the system over a specified time. Whereas Predictive reliability assigns probabilities to the operational profile of a software system. In practice, reliability trending is more appropriate for software, whereas predictive reliability is more suitable for hardware. Trending reliability can be further classified into four categories: Error Seeding, Failure Rate, Curve Fitting, and Reliability Growth. Error Seeding: Estimates the number of errors in a program by using multistage sampling. Errors are divided into indigenous and induced (seeded) errors. The unknown number of indigenous errors is estimated from the number of induced errors and the ratio of errors obtained from debugging data. Failure Rate: Is used to study the program failure rate per fault at the failure intervals. As the number of remaining faults change, the failure rate of the program changes accordingly. Curve Fitting: Uses statistical regression analysis to study the relationship between software complexity and the number of faults in a program, as well as the number of changes, or failure rate.
Reliability Growth: Measures and predicts the improvement of reliability programs through the testing process. Reliability growth also represents the reliability or failure rate of a system as a function of time or the number of test cases. As reliability is of great concern for software products this field is catching attention of various researchers and practitioners. This has lead to a new concept Software Reliability Growth Modelling. An SRGM is defined as a tool that can be used to evaluate the software quantitatively, develop test status, schedule status, and monitor the changes in reliability performance. Software reliability assessment and prediction is important to evaluate the performance of software system. The reliability of the software is quantified in terms of the estimated number of faults remaining in the software system. During the testing phase, the emphasis is on reducing the remaining fault content hence increasing the software reliability. The SRGMs developed in literature are either dependent on testing time, testing effort or testing coverage. Testing time is the calendar time or the CPU time whereas testing effort takes into account the manpower time and the CPU time. We discuss some novel software reliability growth models dependent on time in Section 2.

Testing Coverage plays a very important role in predicting the software reliability. Testing Coverage is actually a structural testing technique in which the software performance is judged with respect to specification of the source code and the extent or the degree to which software is executed by the test cases [8, 16]. TC can help software developers to evaluate the quality of the tested software and determine how much additional effort is needed to improve the reliability of the software besides providing customers with a quantitative confidence criterion while planning to use a software product. Hence, safety critical system has a high coverage objective. The basic testing coverage measures are [8, 16]:

1. Statement Coverage: It is defined as the proportion of lines executed in the program. If we assume that the faults are uniformly distributed throughout the code, then percentage of executable statements covered shows the percentage of faults discovered.
2. Decision / Condition Coverage: This measure indicates whether Boolean expressions tested in control structures evaluated to both true and false.
3. Path Coverage: This measure shows the percentage of all possible paths existing in the code exercised by the test cases.
4. Function Coverage: This measure indicates the proportion of functions/ procedures influenced by the testing Software reliability growth models based on testing coverage have been developed by Inoue et al and Pham et al [8, 16] in literature based on various testing coverage functions. These testing coverage functions are based on time and are utilized to measure the coverage of the software. We discuss this in detail later in the paper.

The past 30 years have seen the formulation of a number of software reliability growth models to predict the reliability and error content of software systems. These models are concerned with forecasting future system operability from the failure data collected during the testing phase of a software product. A plethora of reliability models have appeared in the literature. Software reliability is important because of our dependency on computer software system in our daily life and to the fact that software system cannot be made error free. In the last two decades different methodologies and techniques have been developed and put in to practice in the hope of producing high quality, low cost software systems. Generally the software development process is composed of four phases: requirement phase, design phase, coding and testing. The testing phase aims to detect and remove the latent software errors in order to ensure, as far as possible, error free operation of software in a given time. In other words, the testing phase quantifies the quality of the software in terms of its reliability. Thus software reliability is dependent on the number of errors remaining in the software. Recently, Huang et al [4] discussed software reliability analysis and assessment using queuing models with multiple change-points. Kapur et al [14] discussed a flexible software reliability growth model using stochastic differential equations. Inoue et al [6] developed a two dimensional software reliability model extending the one unit time scale to a two unit time and coverage scale.

Traditionally, one-dimensional models have been proposed in the literature with respect to testing time or testing coverage, although not much has been done to capture the collective effect of the testing coverage and the testing time. Ishii and Dohi[8] proposed a two dimensional software reliability growth model and their application .In this paper we develop a two-dimensional model which shows the combined effect of testing time and testing coverage to remove the faults lying dormant in the software. We assume that the number of faults removed in the software by a fixed time is dependent on the total testing resources available to the testing team. This testing resource is a fusion of both testing time and testing coverage. We employ Cobb-Douglas production function [8] to demonstrate the effect of both testing time and testing coverage in removing the faults in the software. Inoue proposed a two dimensional software reliability growth model with testing coverage using the Cobb Douglas production function.

2. TIME DEPENDENT MODEL

The time dependent behavior of fault removal process is explained by a Software Reliability Growth Model (SRGM). In literature, several SRGMs have been proposed to measure the reliability during the testing phase. Most of these can be categorized under Non Homogeneous Poisson Process (NHPP) models. The assumption that governs these models is, ‘the software failure occurs at random times during testing caused by faults lying dormant in software.’ And, for modeling the software fault detection phenomenon, counting process \( \{ N(t); t \geq 0 \} \) is defined which represents the cumulative
Two Dimensional Flexible Software Reliability Growth Model with Two Types of Imperfect Debugging

number of software faults detected by testing time \( t \). The SRGM based on NHPP is formulated as:

\[
Pr\{N(t) = n\} = \frac{m(t)^n \cdot \exp(-m(t))}{n!}, \quad n = 0, 1, 2, \ldots
\]

where \( m(t) \) is the mean value function of the counting process \( N(t) \).

2.1 TESTING COVERAGE BASED MODELING

NOTATIONS

\( m(t) \): Expected number of faults identified in the time interval \((0, t]\).

\( c(t) \): Testing coverage as a function of time \( t \).

\( v \): Constant

\( N \): Constant, representing the number of faults lying dormant in the software at the beginning of testing.

The testing coverage based software reliability growth model can be formulated as follows:

\[
\frac{dm(t)}{dt} = c(t) \frac{t}{1-c(t)} \left(N-m(t)\right)
\]

Here \( c(t) \) defines the percentage of the coded statements that has been observed till time \( t \). So, \( 1 - c(t) \) defines the percentage of the coded statements which has not yet been covered till time \( t \). Then, the first order derivative of \( c(t) \), denoted by \( c'(t) \), represents the testing coverage rate.

Therefore, function \( \frac{c'(t)}{1-c(t)} \) can be taken as a measure of the fault detection rate [7]. In one dimensional SRGM with testing coverage we need to define coverage function \( c(t) \) although in a two dimensional modeling approach we need not define a coverage function and it can be estimated directly from the data.

3. S-SHAPED FLEXIBLE MODEL

In 1992 Kapur and Garg [6] developed an S-shaped model with an assumption that when we remove the faults in the software some additional faults in the software are removed without actually affecting the system. The revised Kapur garg model [14] is derived using a logistic rate as the detection rate to capture the effect of imperfect debugging and fault generation. This model was based on the assumption of Non-Homogeneous Poisson Process. The basic assumptions of the model are as follows:

1. Failure /fault removal phenomenon is modeled by NHPP.
2. Software is subject to failures during execution caused by faults remaining in the software.
3. Failure rate is equally affected by all the faults remaining in the software.
4. Fault detection / removal rate may change at any time moment.

NOTATIONS

\( b \): Rate at which a fault is detected/removed in the software

\( m \): Mean Number of faults detected/ Corrected corresponding to testing time \( t \)

\( \beta \): \( \tan^{-1} t \)

\( x \): Rate of error generation

\( p \): Probability of imperfect debugging

The differential equation of the representing the rate of change of cumulative number of faults detected in time \( t \) is given as

\[
m'(t) = \frac{b}{1+\beta \exp(-bt)}\left(N-m(t)\right)
\]

The mean value function of the number of faults detected in time \( t \) is given as:

\[
m(t) = \frac{N\left(1-\exp(-bt)\right)}{1+\beta \exp(-bt)}
\]

4. TWO -DIMENSIONAL MODEL

Lately, two dimensional software reliability models have been developed to assess the software quantitatively. The need for developing a two dimensional model is an ideal solution to the problem of software reliability at the hands of software engineers. In one dimensional analysis the object variable is dependent on one basic variable although the object takes on many different roles based upon its dependence on various other factors. Two dimensional models are used to capture the joint effect of testing time and testing coverage on the number of faults removed in the software. The traditional one dimensional model has been dependent upon the testing time, testing effort or testing coverage. However, if the reliability of a software is measured on the basis on the number of hours spent on testing the software or the percentage of software that has been covered then the results are not conclusive. To cater the need of high precision software reliability we require a software reliability growth model which caters not only the testing time but also the testing coverage of the software i.e. the percentage of code covered of the software. For this we develop a two dimensional software reliability growth model incorporating the joint effect of testing time and testing effort on the number of faults removed in the software. The two dimensional model developed in this paper is based on the Cobb Douglas production function. The Cobb–Douglas functional form of production functions is widely used to represent the relationship of an output to inputs. It was proposed by Knut Wicksell (1851–1926), and tested against statistical evidence by Charles Cobb and Paul Douglas in 1900–1928. The Cobb–Douglas function considered a simplified
view of the economy in which production output is determined by the amount of labor involved and the amount of capital invested. While there are many factors affecting economic performance, their model proved to be remarkably accurate.

The mathematical form of the production function is given as follows:

\[ Y = A L^\alpha K^{1-\gamma} \]

Where:

- \( Y \) = total production (the monetary value of all goods produced in a year),
- \( L \) = labor input,
- \( K \) = capital input,
- \( A \) = total factor productivity \( \gamma \) is elasticity of labor. This value is constants and determined by available technology. Fig 1 shows graphically how the total production is influenced due to change in the proportion of labor and capital.

![Cobb Douglas Utility Function](image)

Fig1. A two-input Cobb–Douglas production function

The assumptions made by Cobb and Douglas can be stated as follows:

1. If either labor or capital vanishes, then so will production.
2. The marginal productivity of labor is proportional to the amount of production per unit of labor.
3. The marginal productivity of capital is proportional to the amount of production per unit of capital.

The Cobb-Douglas function based on the above assumptions is very appealing. The basic characteristic of this function is linearly homogeneous with constant return to scale i.e. a proportion increase in all inputs leads to same proportion increase in output.

Software Testing involves operation of a system or application under controlled conditions and evaluating the results (e.g., ‘if the user is in interface A of the application while using hardware B, and does C, then D should happen’). The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn’t or things don’t happen when they should. It is oriented to ‘detection’. The testing team has many resources of testing to make sure that software hence formed is of quality. These include software testing man hours, CPU time, testing effort testing coverage etc.

\[ \tau \equiv s^\alpha u^{1-\alpha} \quad 0 \leq \alpha \leq 1 \]  

(1)

Where

- \( \tau \) : testing resources
- \( s \) : testing time
- \( u \) : testing coverage
- \( \alpha \) : Effect of testing time

Let \( \{N(s,u), s \geq 0, u \geq 0\} \) be a two-dimensional stochastic process representing the cumulative number of software failures by time \( s \) and testing coverage \( u \). A two-dimensional NHPP with a mean value function \( m(s,u) \) is formulated as:

\[ \Pr(N(s,u) = n) = \frac{(m(s,u))^n}{n!} \exp(-m(s,u)), \quad n = 0, 1, 2... \]

and

\[ m(s,u) = \int \int \lambda(\zeta, \xi)d\zeta d\xi \]

4.1 TWO DIMENSIONAL S-SHAPED MODEL

In this paper we develop a two dimension S-shaped model determining the combined effect of testing time and testing coverage. We define some additional notations as follows:

- \( m(s,u) \) : Mean Number of faults removed corresponding to Coverage \( u \) and time \( s \)

The differential equation of the representing the rate of change of cumulative number of faults detected w.r.t. to the total testing resources is given as:

\[ m(\tau) = \frac{b}{1 + \beta \exp(-b\tau)} \left( N - m(\tau) \right) \]

The mean value function of the number of faults detected with testing resources \( x \) using the initial condition \( x(0) = 0 \) is given as:

\[ m(\tau) = N \left( 1 - \exp(-b\tau) \right) \frac{1}{1 + \beta \exp(-b\tau)} \]  

(2)

Now we extend the testing resource of one dimensional S-shaped model to a two dimensional problem. Using the cobb-douglas production the corresponding mean value function is given as:

\[ m(\tau) = \frac{N \left( 1 - \exp(-b^s u^{1-\alpha}) \right)}{1 + \beta \exp(-b^s u^{1-\alpha})} \]

In the above two-dimensional mean value function if \( \alpha = 1 \) the above mean value function can be regarded as a
4.2 TWO –DIMENSIONAL S-SHAPED MODEL WITH IMPERFECT DEBUGGING

Most of the debugging process in real life is not perfect. During the fault removal process two possibilities can occur. It may happen that the fault, which was considered to be perfectly fixed, had been improperly repaired and resulted in same type of failure again. There is also a fine chance that some new faults might get introduced during the course of correcting. This situation is much dangerous than the former one, because in the first case the total fault content is not altered, whereas in latter, error generation resulted in increased fault content. The effects of both type of imperfect debugging during testing phase are incorporated in our proposed model.

The rate equation of flexible model with imperfect debugging and error generation can be written as follows:

\[ \frac{d}{dt} m(t) = \frac{bp}{1 + \beta \exp(-\beta t)} [N + xm(t) - m(t)] \]

We use logistic function to incorporate the effect of imperfect debugging and error generation.\[13\]

Solving using initial conditions \( N(0) = 0 \)

\[ N(t) = \frac{N}{1-x} \left\{ 1 - \left( \frac{(1+\beta)}{1+\beta \exp(-\beta t)} \right)^{p(1-x)} \right\} \exp(-\beta p(1-x)) \]  (3)

Using (1) and (3)

5. RELIABILITY EVALUATION

Software evaluation is a very significant phenomenon in quantitative software reliability assessment. The software reliability function signifies the probability that a software failure does not occur in time-interval \( (t, t+x) \) \( t \geq 0, x \geq 0 \) given that the testing team or the user operation has been going up to time \( t \). In two dimensional SRGM, we can assess software reliability in an operation phase where we assume that the testing coverage is not expanded. We can derive the probability that the software failure does not occur in time-interval \([s_\pi, s_\pi + \omega]\) \( s_\pi > 0, \omega > 0 \) given that testing has been going up to \( s_\pi \) and the value of testing coverage has been attained up to \( u_\pi \) by testing termination time \( s_\pi \) as:

\[ R(\omega|s_\pi,u_\pi) = \exp\left[-\left(m(s_\pi + \omega,u_\pi)/\kappa\right) - m(s_\pi,u_\pi)/\kappa\right] \]  (7)

Where \( \kappa \) indicates the set of parameter estimates of a two-dimensional SRGM

6. ESTIMATION OF PARAMETERS MODEL VALIDATION AND COMPARISON CRITERIA

<table>
<thead>
<tr>
<th></th>
<th>Proposed Model</th>
<th>One dimensional (KG)</th>
<th>One dimensional (GO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS-1</td>
<td>MSE</td>
<td>0.0543</td>
<td>0.682</td>
</tr>
<tr>
<td></td>
<td>Bias</td>
<td>0.0012</td>
<td>0.139</td>
</tr>
<tr>
<td></td>
<td>Adj R²</td>
<td>.989</td>
<td>.912</td>
</tr>
<tr>
<td></td>
<td>Variation</td>
<td>.175333</td>
<td>.9565</td>
</tr>
<tr>
<td></td>
<td>RMPSE</td>
<td>.175334</td>
<td>.9695</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>DS-2</td>
<td>MSE</td>
<td>0.0237</td>
<td>0.1658</td>
</tr>
<tr>
<td></td>
<td>Bias</td>
<td>-0.027</td>
<td>-0.2298</td>
</tr>
<tr>
<td></td>
<td>Adj R²</td>
<td>.994</td>
<td>.901</td>
</tr>
<tr>
<td></td>
<td>Variation</td>
<td>0.105</td>
<td>0.836</td>
</tr>
<tr>
<td></td>
<td>RMPSE</td>
<td>0.106</td>
<td>0.843</td>
</tr>
</tbody>
</table>

Table 1: Model Comparison

CONCLUSION

Software reliability engineering uses quantitative measurement to increase the efficiency of the testing effort. By developing operational profiles of the system’s use, SRE requires that trade-offs between time, cost, and quality be made explicitly for the project. There is still much room for improvement of quality assurance practices within the software development industry and software reliability engineering provides a set of practices that is certain to help. A key challenge to software reliability engineering however is its reliance on heavyweight documentation processes; particularly in the face of the agile development movement. In this paper we have developed a general approach in deriving more general models based on simple assumptions, constant with the basic software reliability growth modeling based on NHPP. The proposed models implant a broader theoretical framework which accounts for interaction between different dimensions of software reliability metrics. Incorporating the dynamics of testing time of the software and the testing coverage has allowed us the model to
be a two dimensional framework. The proposed models use the Cobb Douglas production function to capture the combined effect of testing time and testing coverage. The proposed models are validated on real data sets and analyses are done using goodness of fit criterion.

Table 2: Parameter Estimates

<table>
<thead>
<tr>
<th>Proposed Model</th>
<th>N</th>
<th>p</th>
<th>b</th>
<th>( \alpha )</th>
<th>x</th>
<th>Beta</th>
</tr>
</thead>
<tbody>
<tr>
<td>One dimensional(GO)</td>
<td>9.23</td>
<td>.045</td>
<td>.0003</td>
<td>-</td>
<td>.045</td>
<td>-</td>
</tr>
<tr>
<td>One dimensional(KG)</td>
<td>10.32</td>
<td>.045</td>
<td>.001</td>
<td>-</td>
<td>.045</td>
<td>6.02</td>
</tr>
<tr>
<td>Proposed Model</td>
<td>11.323</td>
<td>0.001</td>
<td>0.056</td>
<td>0.125</td>
<td>0.001</td>
<td>1.67</td>
</tr>
<tr>
<td>One dimensional(GO)</td>
<td>10.03</td>
<td>.015</td>
<td>.0003</td>
<td>-</td>
<td>.015</td>
<td>-</td>
</tr>
<tr>
<td>One dimensional(KG)</td>
<td>11</td>
<td>.015</td>
<td>.001</td>
<td>-</td>
<td>.015</td>
<td>2.343</td>
</tr>
</tbody>
</table>

Table 2: Parameter Estimates

**FUTURE SCOPE**

We have focused only on two dimensional framework in this work. However, it is known that a software reliability growth process in a testing-phase is influenced by the following several software reliability factors: the test execution-time, the testing-skill and so forth. In future we can explore the possibility of including multi dimensional software reliability growth modeling so as to take care the effect of not only testing coverage but also other testing factors like testing effort, testing time/number of test cases on the fault removal process simultaneously.

**REFERENCES**


[8]. Shinji Inoue and Shigeru Yamada “Two-Dimensional Software Reliability Assessment with Testing-Coverage” 2008 Second International Conference on Secure System Integration and Reliability Improvement July 14-July 17

[9]. Tomotaka Ishii and Tadashi Dohi “Two-Dimensional Software Reliability Models and Their Application” 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)


Two Dimensional Flexible Software Reliability Growth Model with Two Types of Imperfect Debugging