Flexible Discrete Software Reliability Growth Model for Distributed Environment  
Incorporating two types of Imperfect Debugging

Sunil Kumar Khatri  
Amity Institute of Information Technology  
Amity University, Noida, India  
e-mail: sunilkkhatri@gmail.com

P. K. Kapur  
Amity International Business School, Amity University, Noida, India  
e-mail: pkkapur1@gmail.com

Prashant Johri  
Department of Computer Science, Galgotias University, Greater Noida, India  
e-mail: johri.prashant@gmail.com

Abstract— In literature we have several software reliability growth models developed to monitor the reliability growth during the testing phase of the software development. These models typically use the calendar / execution time and hence are known as continuous time SRGM. However, very little seems to have been done in the literature to develop discrete SRGM. Discrete SRGM uses test cases in computer test runs as a unit of testing.

Debugging process is usually imperfect because during testing all software faults are not completely removed as they are difficult to locate or new faults might be introduced. In real software development environment, the number of failures observed need not be same as the number of errors removed. If the number of failures observed is more than the number of faults removed then we have the case of imperfect debugging. Due to the complexity of the software system and the incomplete understanding of the software requirements, specifications and structure, the testing team may not be able to remove the fault perfectly on detection of the failure and the original fault may remain or get replaced by another fault.

In this paper, we discuss a discrete software reliability growth model for distributed system considering imperfect debugging that faults are not always corrected/removed when they are detected and fault generation. The proposed model assumes that the software system consists of a finite number of reused and newly developed sub-systems. The reused sub-systems do not involve the effect of severity of the faults on the software reliability growth phenomenon because they stabilize over a period of time i.e. the growth is uniform whereas, the newly developed subsystem does involve. For newly developed component, it is assumed that removal process follows logistic growth curve due to the fact that learning of removal team grows as testing progresses. The fault removal phenomena for reused and newly developed sub-systems have been modeled separately and are summed to obtain the total fault removal phenomenon of the software system. The model has been validated on two software data sets and it is shown that the proposed model fairs comparatively better than the existing one.

Keywords- Probability Generating Function (PGF); Imperfect Debugging; Fault Removal Rate (FRR); Distributed Development Environment; Logistic removal rate

I. INTRODUCTION

Developing software system is generally a quite complex and time-consuming process. Moreover, the nature and complexity of software requirements have drastically changed in the last few decades and users all over the world have become much more demanding in terms of cost, schedule and quality. These three parameters, all being desirable, have an apparent contradiction at times, which can only be resolved by optimum design of software using well-established software engineering methodologies. The only way to verify and validate the software is by testing. The software testing involves running the software and checking for unexpected behavior of the software output.

A number of SRGM have been developed in the literature, under different sets of assumptions and testing environments. SRGM are generally classified into two groups. The first group contains models, which use the execution time (i.e., CPU time) or calendar time. Such models are called continuous time models [1,4,7,10,12]. The second group contains models, which use the test cases as a unit of fault removal period. Such models are called discrete time models, since the unit of software fault removal period is countable [2,9,11,17]. A test case can be a single computer test run executed in an hour, day, week or even month. Therefore, it includes the computer test run and length of time spent to visually inspect the software source code. A large number of models have been developed in the first group while fewer are there in the second group due to the difficulties in terms of mathematical complexity involved. The utility of discrete SRGM cannot be under estimated. As the software failure data sets are discrete, these models many times provide better fit than their continuous time counterparts. Therefore, in spite of difficulties in terms of mathematical complexity involved, discrete models are proposed regularly.

The concept of imperfect debugging was first introduced by Goel [3]. He introduced the probability of imperfect debugging in Jelinski and Moranda [6]. Kapur and Garg [7] introduced the imperfect debugging in Goel and Okumoto [4]. They assumed that the FRR per remaining faults is
reduced due to imperfect debugging. Thus the number of failures observed by time infinity is more than the initial fault content. Although these two models describe the imperfect debugging phenomenon yet the software reliability growth curve of these models is always exponential. Recently, Kapur et al. [10] proposed three discrete models taking into accounts imperfect debugging and fault generation phenomena.

Yamada et al. [19] have proposed an SRGM based upon NHPP for a distributed software system consisting of \( n \) reused and \( m \) newly developed software components assuming that cumulative number of detected faults is an exponential curve when a software system consists of several used software components; while on the other hand, the cumulative number of faults is described by an s-shaped growth curve when the newly developed software components are used. Kapur et al [7, 8] proposed an SRGM with three types of fault. For each type, the Fault Removal Rate per remaining faults is assumed to be time independent. The first type is modeled by an Exponential model of Goel and Okumoto [4]. The second type is modeled by Delayed S-shaped model of Yamada et al. [18]. The third type is modeled by three stages Erlang model proposed by Khoshgoftaar [16]. The total removal phenomenon is again modeled by the superposition of the three SRGM [12]. The fault is classified as simple if the delay between failure observation, fault isolation and fault removal is negligible. If there is delay, it is classified as a hard fault. If the removal of a fault after its isolation involves an even greater delay, it is classified as a complex fault. Later they extended their model to cater for more types of faults [8].

In this paper, we propose a flexible discrete software reliability growth model for distributed systems considering both types of imperfect debugging. The model is based on Non Homogenous Poisson Process (NHPP). The model has been validated on two datasets and it is shown that the proposed model fairs well in comparison to the discrete version of the model proposed by Goswami et al [5]. For estimation of parameters, SPSS is used. SPSS is a Statistical package for Social Sciences.

The paper is organized as follows: Section 2 discusses the flexible discrete model development under imperfect debugging and fault generation. Sections 3 gives the estimated results of the developed model to actual software reliability data sets collected from real software development projects. Conclusion of the paper is given in Section 4.

**NOTATIONS**

- \( a \) Total fault content
- \( a_i \) Initial fault content of type \( i \) reused component
- \( a_j \) Initial fault content of type \( j \) newly developed components with hard faults
- \( a_k \) Initial fault content of type \( k \) newly developed component with complex faults
- \( b_i \) Proportionality constant failure rate/fault isolation rate per fault of \( i^{th} \) reused component with simple faults.
- \( b_j \) Proportionality constant failure rate/fault isolation rate per fault of \( j^{th} \) newly developed component with hard faults.
- \( b_k \) Proportionality constant failure rate/fault isolation rate per fault of \( k^{th} \) newly developed component with complex faults.
- \( b_j(n) \) Fault removal rate per fault of \( j^{th} \) newly developed component and is assumed to follow logistic function
- \( b_k(n) \) Fault removal rate per fault of \( k^{th} \) newly developed component and is assumed to follow logistic function
- \( p_i \) The probability of fault removal on a failure (i.e., the probability of perfect debugging) of \( i^{th} \) reused components with simple faults.
- \( p_j \) The probability of fault removal on a failure (i.e., the probability of perfect debugging) of \( j^{th} \) newly developed components with hard faults.
- \( p_k \) The probability of fault removal on a failure (i.e., the probability of perfect debugging) of \( k^{th} \) newly developed components with complex faults.
- \( \alpha_i \) The rate at which the faults may be introduced during the debugging process per detected faults of \( i^{th} \) reused components with simple faults.

**BASIC ASSUMPTIONS**

The proposed model is based upon the following basic assumptions:

1. Failure observation / fault removal phenomenon is modeled by NHPP.
2. Software is subject to failures during execution caused by faults remaining in the software.
3. Software system consists of a finite number of reused and newly developed sub-systems.
4. FRR is a logistic learning function dependent on the number of test cases.
5. During the fault isolation / removal, new fault can introduce into the system.
6. The debugging process is imperfect in two ways- incomplete removals and error generation.
7. Software reliability growth in the reused sub-system is constant while in the newly developed sub-system is not.
8. Each time a failure is observed, an immediate (delayed) effort takes place to decide the cause of the failure in order to remove it. The delay between the failure observation and its subsequent removal is assumed to represent the complexity of faults. The more complex the fault, more the delay.
9. Failure rate of the software is equally affected by faults remaining in the software.
\( \alpha_j \) The rate at which the faults may be introduced during the debugging process per detected faults of \( j^{th} \) newly developed components with hard faults.

\( \alpha_k \) The rate at which the faults may be introduced during the debugging process per detected faults of \( k^{th} \) newly developed components with complex faults.

\( m_i(n) \) Mean number of faults removed from \( i^{th} \) reused component by \( n^{th} \) test case

\( m_f(n) \) Mean number of failures caused by \( f^{th} \) newly developed component by \( n^{th} \) test case

\( m_r(n) \) Mean number of faults removed from \( r^{th} \) newly developed component by \( n^{th} \) test case

\( m_f(n) \) Mean number of failures caused by \( f^{th} \) newly developed component by \( n^{th} \) test case

\( m_i(n) \) Mean number of failures caused by \( i^{th} \) newly developed component by \( n^{th} \) test case

\( m_f(n) \) Mean number of failures caused by \( f^{th} \) newly developed component by \( n^{th} \) test case

\( m_i(n) \) Mean number of failures caused by \( i^{th} \) newly developed component by \( n^{th} \) test case

\( r_1 \) Reused components having simple type of faults

\( r_2 \) Developed components having hard type of faults

\( r_3 \) Developed components having complex type of faults

\( b_1 \) Proportionality constant failure rate/fault isolation rate per fault of reused component with simple faults.

\( b_2 \) Proportionality constant failure rate/fault isolation rate per fault of newly developed component with hard faults.

\( b_3 \) Proportionality constant failure rate/fault isolation rate per fault of newly developed component with complex faults.

\( p_1 \) The probability of fault removal on a failure (i.e., probability of perfect debugging) of the reused component with simple faults.

\( p_2 \) The probability of fault removal on a failure (i.e., probability of perfect debugging) of the newly developed component with hard faults.

\( p_3 \) The probability of fault removal on a failure (i.e., probability of perfect debugging) of the newly developed component with complex faults.

\( \alpha_i \) Rate at which faults may be introduced during the debugging process per detected faults of the reused component with simple faults.

\( \alpha_k \) Rate at which faults may be introduced during the debugging process per detected faults of the newly developed component with hard faults.

\( \alpha_f \) Rate at which faults may be introduced during the debugging process per detected faults of the newly developed component with complex faults.

\( \delta \) Constant time difference interval.

\( \beta_{ij} \) Constant parameter in the logistic learning function of \( i^{th} \) newly developed component with hard faults.

\( \beta_{kl} \) Constant parameter in the logistic learning function of \( k^{th} \) newly developed component with complex faults.

\( \beta_{il} \) Constant parameter in the logistic learning function of newly developed component with hard faults.

\( \beta_{jl} \) Constant parameter in the logistic learning function of newly developed component with complex faults.

I. MODELING SOFTWARE RELIABILITY

Goswami et al. [5] have proposed an continuous SRGM based upon NHPP for a distributed software system consisting of ‘x’ reused software components, ‘y’ newly developed software components with hard faults and ‘z’ newly developed software components with complex faults and assuming that cumulative number of detected faults is an exponential curve when a software system consists of several reused software components; while on the other hand, the cumulative number of faults is described by an S-Shaped growth curve when the newly developed software components are used. Under this assumption, the discrete SRGM based on NHPP for a distributed development environment can be formulated as

\[
m(n) = \sum_{i=1}^{a_i} \left[ 1 - (1 - \delta_{\alpha_i})^{p_i(1-\delta_{\alpha_i})} \right] + \sum_{j=1}^{b_j} \left[ 1 - (1 + \delta_{\beta_j})^{n_j(1-\delta_{\beta_j})} \right] + \sum_{k=1}^{c_k} \left[ 1 - (1 + \delta_{\gamma_k})^{n_k(1-\delta_{\gamma_k})} \right]
\]

This is the discrete version of the model in Goswami et al [5].

Consider the simple case where software system consists of one component of each type of faults (simple, hard and complex), equation (1) can be rewritten as

\[
m(n) = \sum_{i=1}^{a_i} \left[ 1 - (1 - \delta_{\alpha_i})^{p_i(1-\delta_{\alpha_i})} \right] + \sum_{j=1}^{b_j} \left[ 1 - (1 + \delta_{\beta_j})^{n_j(1-\delta_{\beta_j})} \right] + \sum_{k=1}^{c_k} \left[ 1 - (1 + \delta_{\gamma_k})^{n_k(1-\delta_{\gamma_k})} \right]
\]

II. PROPOSED MODEL

A. Modeling The Fault Removal Of Reused Components

Faults in the reused components are simple faults, which can be removed instantly as soon as they are observed. Hence, under assumptions 1 to 9, fault removal of reused components is modeled as one-stage processes:
\[
\frac{m_{fr}(n+1) - m_{fr}(n)}{\delta} = p_j b_j(n+1)[a_i(n) - m_{fr}(n)]
\]

where, \(a_i(n) = a_i + \alpha_i \) \(m_{fr}(n)\) and \(b_j(n+1) = b_j\)

Solving equation (3) using P.G.F. under initial condition \(m_{fr}(n) = 0\), we get mean value function as

\[
m_{fr}(n) = \frac{a_i}{1- \alpha_i}[1 - (1 - \delta b_j)((1-\alpha_i)n)]
\]

B. Modeling the fault removal of Newly Developed Components

Software faults in the newly developed software component can be of different complexity. The faults can either be modeled as two stage or three stage process according to the lag for removal.

Newly Developed Components containing hard faults

It is assumed that the faults of some newly developed components consume more testing effort when compared with faults of reused component. This means that the testing team will have to spend more test cases to analyze the cause of the failure and therefore requires greater efforts to remove them. Hence the removal process for such faults is modeled as a two-stage process:

\[
\frac{m_{fr}(n+1) - m_{fr}(n)}{\delta} = b_j[a_i - m_{fr}(n)]
\]

where \(m_{fr}(n)\) is the expected number of failures in \((0, n]\) and

\[
b_j(n+1) = \frac{b_j}{1 + \beta_{ij}(1 - \delta b_j)^{n+1}}
\]

The first stage of the two-stage process is given by the equation (5). This stage describes the failure observation process. The second stage of the two-stage process given by equation (6) describes the delayed fault removal process.

Solving equations (5) and (6) using P.G.F. under initial conditions \(m_{fr}(n) = 0\) and \(m_{fr}(n) = 0\), we get the mean value function as

\[
m_{fr}(n) = \frac{a_i}{1 - \alpha_i} \left\{ \frac{1 - (1 + \delta b_j)(1-\alpha_i)n}{1 + \beta_{ij}(1 - \delta b_j)^n} \right\}
\]

The above model can be derived in one stage directly as follows:

\[
\frac{m_{fr}(n+1) - m_{fr}(n)}{\delta} = b_j(n)(a_i - m_{fr}(n))
\]

where

\[
b_j(n) = \frac{b_j(1 + \beta_{ij} + b_j n \delta) - b_j(1 + \beta_{ij}(1 - b_j \delta)^n)}{(1 + \beta_{ij}(1 - b_j \delta)^n)(1 + \beta_{ij} + b_j n \delta)}
\]

It is observed that

\[
\frac{b_j(1 + \beta_{ij} + b_j n \delta) - b_j(1 + \beta_{ij}(1 - b_j \delta)^n)}{(1 + \beta_{ij}(1 - b_j \delta)^n)(1 + \beta_{ij} + b_j n \delta)} \rightarrow b_j \text{ as } n \rightarrow \infty.
\]

This model was specifically developed to account for lag in the failure observation and its subsequent removal. This kind of derivation is peculiar to software reliability only [8].

Under assumptions 1 to 9, the equation describing the removal process is given as

\[
\frac{m_{fr}(n+1) - m_{fr}(n)}{\delta} = p_j b_j(n)[a_j(n) - m_{fr}(n)]
\]

where, \(b_j(n) = \frac{b_j(1 + \beta_{ij} + b_j n \delta) - b_j(1 + \beta_{ij}(1 - b_j \delta)^n)}{(1 + \beta_{ij}(1 - b_j \delta)^n)(1 + \beta_{ij} + b_j n \delta)}\),

and \(a_j(n) = a_j + \alpha_j m_{fr}(n)\)

Solving the equation (10) using P.G.F. under initial condition \(m_{fr}(0) = 0\), we get mean value function as

\[
m_{fr}(n) = \frac{a_j}{1 - \alpha_j} \left\{ \frac{1 + \beta_{ij} + \delta b_j n}{1 + \beta_{ij}(1 - \delta b_j)^n} \right\} (1-\alpha_i)(p_i(1-\alpha_i)n)
\]

Newly Developed Components containing complex faults

There can be components still having harder faults or complex faults. These faults can require more effort for removal after isolation. Hence they need to be modeled with greater lag between failure observation and removal. The third stage added below to the model serves the purpose.

\[
\frac{m_{kf}(n+1) - m_{kf}(n)}{\delta} = b_k(a - m_{kf}(n))
\]

\[
\frac{m_{k}(n+1) - m_{k}(n)}{\delta} = b_k(m_{kf}(n) - m_{ki}(n))
\]

\[
\frac{m_{kr}(n+1) - m_{kr}(n)}{\delta} = b_k(n+1)(m_{ki}(n) - m_{kr}(n))
\]

where

\[
b_k(n+1) = \frac{b_k}{1 + \beta_{ik}(1 - \delta b_i)^{n+1}}
\]

The first stage of the three-stage process is given by the equation (12). This stage describes the failure observation process. The second stage given by equation (13) describes the fault isolation process. The third stage given by equation (14) describes the fault removal process.

Solving equations (12), (13) and (14) using P.G.F. under the initial condition, \(m_{kf}(n) = 0\), \(m_{k}(n) = 0\) and \(m_{kr}(n) = 0\), we get:
Solving the differential equation (18) under initial condition written as

\[
m_{kr}(n) = \frac{1 - \left(1 + b_k n + \frac{b_k^2 n(n+1)}{2}(1-b_k)\right)}{1 + \beta_{2k}(1-\delta b_k)^{n+1}} \]

This model can be derived in one stage directly and can be written as

\[
m_{kr}(n+1) - m_{kr}(n) = b_k(n)[a_k(n) - m_{kr}(n)]
\]

where,

\[
b_k(n) = \frac{-b_k(1 + \beta_{2k} + b_k n \delta + \frac{b_k^2 n \delta^2(n+1)}{2})}{(1 + \beta_{2k}(1-\delta b_k)^n)(1 + \beta_{2k} + b_k n \delta + \frac{b_k^2 n \delta^2(n+1)}{2})}
\]

and \(a(n) = a_k\)

Under assumptions 1 to 9, the differential equation describing the removal process is given as

\[
m_{kr}(n+1) - m_{kr}(n) = p_k b_k(n)[a_k(n) - m_{kr}(n)]
\]

where,

\[
b_k(n) = \frac{-b_k(1 + \beta_{2k} + b_k n \delta + \frac{b_k^2 n \delta^2(n+1)}{2})}{(1 + \beta_{2k}(1-\delta b_k)^n)(1 + \beta_{2k} + b_k n \delta + \frac{b_k^2 n \delta^2(n+1)}{2})}
\]

and \(a_k(n) = a_k + \alpha_k m_{kr}(n)\)

Solving the differential equation (18) under initial condition \(m_{is}(n=0) = 0\), we get mean value function as

\[
m_{is}(n) = \frac{a_i}{1-\alpha_i} \left[1 - \left(1 + \beta_{2i} + \delta b_i n + \frac{b_i^2 \delta^2 n(n+1)}{2}(1-b_i)\right)^{\alpha_i}\right]^{p_i(1-\alpha_i)n}
\]

\[
m_{kr}(\infty) = \frac{a_k}{1-\alpha_k}\]

which implies that if testing is carried out for an infinite number of test cases more faults are removed as compared to the initial fault content because there are some errors added to the software due to error generation. It is important to note that imperfect repair of faults results in more number of failures than removals and has no contribution in increasing the fault content.

C. Modeling total fault removal phenomenon

The proposed model is the superposition of the NHPP of 'x' reused and 'y' & 'z' newly developed components. (4), (11) and (19) are mean value functions of respective NHPPs [7]. Thus, the mean value function of superposed NHPP is:

\[
m(n) = \sum_{i=1}^{x+y+z} \frac{a_i}{1-\alpha_i} [1 - \left(1 + \beta_{2i} + \delta b_i n + \frac{b_i^2 \delta^2 n(n+1)}{2}(1-b_i)\right)^{\alpha_i}]^{p_i(1-\alpha_i)n}
\]

where \(a = a_1 + a_2 + a_3\) and \(a_i = a_{ri}\)

III. MODEL VALIDATION

To assess the performance of the proposed SRGM incorporating imperfect debugging and fault generation, we have carried out the parameter estimation on two real software failure datasets.

Data set 1(DS-1)

The first data set (DS-1) had been collected during 38 weeks and 231 faults were detected during testing. This data is cited from Misra [13].

Data set 2(DS-2)

This data is cited from Wood [15]. The software was tested for 20 weeks during which 10000 computer hours were used and 100 faults were removed.

Using Non-linear Regression technique parameters of the proposed SRGM (equation 21) are estimated. The
estimated values of parameters of mean value function \( m(t) \) given by the (eq. 21) are worked out. The estimation results for DS-1 and DS – 2 are provided in Table-1 while the comparison criteria results for DS–1 and DS–2 are shown in Table-2. Figure 1 and 2 shows the Goodness-of-Fit curves for DS-1 and DS – 2.

In the estimation results, \( r_1, r_2 \) and \( r_3 \) are proportion of faults in each of the three components.

From the estimation results for both the data sets, we observe that \( b_i \) is less than \( b_j \) for \( i > j \). It shows the detection rate is less for the more complex faults. Also, \( \alpha_i \) is slightly greater than \( \alpha_j \) which indicates that fault generation rate is higher for more complex faults.

From Figure 1 and 2, it can be observed that the proposed SRGM fits both the data sets very well.

<table>
<thead>
<tr>
<th>TABLE 1: (PARAMETER ESTIMATION)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
</tr>
<tr>
<td>GOSWAMI ET AL (EQ. 2)</td>
</tr>
<tr>
<td>( a )</td>
</tr>
<tr>
<td>( b_1 )</td>
</tr>
<tr>
<td>( b_2 )</td>
</tr>
<tr>
<td>( b_3 )</td>
</tr>
<tr>
<td>( r_1 )</td>
</tr>
<tr>
<td>( r_2 )</td>
</tr>
<tr>
<td>( r_3 )</td>
</tr>
<tr>
<td>( p_1 )</td>
</tr>
<tr>
<td>( p_2 )</td>
</tr>
<tr>
<td>( p_3 )</td>
</tr>
<tr>
<td>( \alpha_i )</td>
</tr>
<tr>
<td>( \alpha_j )</td>
</tr>
<tr>
<td>( \alpha_k )</td>
</tr>
<tr>
<td>( \beta_i )</td>
</tr>
<tr>
<td>( \beta_j )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 2: FOR DS-1 (COMPARISON CRITERIA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Criteria</td>
</tr>
<tr>
<td>GOSWAMI ET AL (EQ. 2)</td>
</tr>
<tr>
<td>( R^2 )</td>
</tr>
<tr>
<td>MSE</td>
</tr>
<tr>
<td>Bias</td>
</tr>
<tr>
<td>Variation</td>
</tr>
<tr>
<td>RMSPE</td>
</tr>
</tbody>
</table>

IV. CONCLUSIONS

A flexible discrete SRGM for distributed environment with two types of imperfect debugging has been proposed. It has been considered that faults are not always corrected/removed when they are detected and fault generation. The results show that the proposed model provides fairly good goodness-of-fit for software failure occurrence / fault removal data due to their applicability.

ACKNOWLEDGMENT

Prof. Sunil Kumar Khatri and Prof. P K Kapur express their deep sense of gratitude to The Founder President of Amity Universe, Dr. Ashok K. Chauhan for his keen interest in promoting research in the Amity Universe and always been an inspiration for achieving greater heights.

REFERENCES


GOODNESS-OF-FIT CURVES

Figure 1. Goodness-of-fit curve for DS-1

Figure 2. Goodness-of-fit curve for DS-2