A Software Reliability Growth Model for Estimating Debugging and the Learning Indices

R. SUBBURAJ*,1, G. GOPAL2 and P. K. KAPUR3

1Department of Information Technology, SRM University, SRM Nagar, Kattankulathur (Chennai), Kancheepuram District, Tamil Nadu - 603203, Tamil Nadu, India
2Department of Statistics, University of Madras, Chennai 600 005, India
3Amity International Business School, Amity University, Noida, India.

(Received on Nov. 28, 2010, revised on June 17, 2011 and June 18, 2012)

Abstract: Most Software Reliability Growth Models (SRGMs) ignore either the learning phenomenon of the testing team or quality of debugging or both for the sake of simplicity. Some SRGMs consider imperfect debugging tightly integrated with the learning phenomenon. In practice, imperfect debugging and the learning phenomenon of the testing team are independent of each other since the former indicates quality of debugging process and the latter the quality of the testing process, usually carried out by different teams. Furthermore, sometimes the debugging team may find and correct some more faults in addition to the faults which caused the failures leading to what is known as efficient debugging. In this paper, a new mathematically derived SRGM is proposed, which also provides estimates of various quality metrics such as the number of faults in the software system, debugging and the learning indices. The proposed SRGM seems to describe the time to failure data of many software projects satisfactorily.

Keywords: Debugging index; efficient debugging; imperfect debugging; NHPP; quality metrics; the learning index

1. Introduction

Due to the very nature of the process, errors creep in during the software development process which could lead to faults and consequently causing failures during the execution of the software system with specified inputs. When failures occur, the associated software system is debugged to find out the fault(s) and correct them. One of the objectives of system testing in the Software Development Life Cycle (SDLC) is to correct as many faults as possible without introducing new faults in the process. The fault correction leads to growth of reliability. The reliability models, used to plan and execute system testing in the SDLC are popularly known as Software Reliability Growth Models (SRGMs) since as a result of testing, faults are corrected and thereby reliability grows. Quantitative information on the quality of the development process, testing process, debugging process and also clear criteria for exit from the testing phase can all be derived by using appropriate SRGMs during the software system testing phase of each project.

It is to be noted that due to the widely varying nature of software development projects in terms of application complexity, technology, competence and skill levels of employees, schedules, resources and budget, the quality of debugging may vary from project to project. To simplify the task of reliability analysis, the early SRGMs assume that fault
fixing is always perfect. Sometimes fault detection and correction may be erroneous. Ohba and Chou [1] observe that “it is hard to assume that no errors are introduced when an error is detected and removed”. According to Kapur and Garg [2], it is not always possible to find the exact cause of the failure and remove it due to insufficient knowledge about the software. Both Ohba and Chou [1] and Kapur and Garg [2] propose an extension to the Goel and Okumoto (G-O) model [3] to describe imperfect debugging phenomenon.

Pham et al. [4] assume that both the learning phenomenon of the testing team and imperfect debugging occur together at the same time always. However, the learning phenomenon of the testing team and imperfect debugging by the development team are independent of each other since the former reflects the skill of testing team and the latter the skill of the debugging team.

Zhang et al. [5] observe that when a failure occurs, a debugging effort is initiated immediately with probability \( p \). For each debugging effort, some new faults may be introduced into the software system with probability \( \beta \) less than \( p \). The fault detection rate function of this model, \( b(t) \) is a non-decreasing function with inflection S-shaped curve as in the case of the model proposed by Pham et al.[4].

A careful study of the features of all the above models reveals the following:

- An increase in total fault count in the software due to imperfect debugging is minimal and could be ignored for all practical purposes.
- Assuming a constant fault detection rate will result in a simpler model without significantly compromising on the estimation of the quality and reliability measures, the intended purpose of the SRGM.

Furthermore, the early models with imperfect debugging assumption proposed by Kapur and Garg [2] as well as Ohba and Chou [1] do not seem to quantify the learning phenomenon of the testing team. Imperfect debugging is independent of the learning phenomenon, whereas the PNZ model proposed by Pham et al. [4] and also the model proposed by Zhang et al. [5] seem to assume that the learning phenomenon and imperfect debugging are tightly integrated. The motivation for our research is to propose a flexible new model – the one that will address various types of debugging and the learning phenomenon of the testing team, independently as well as together with imperfect debugging depending on the actual situation.

According to Kapur and Garg [6] although most SRGMs assume that the fault detection phenomenon also describes the failure phenomenon, in reality it may not always be so. When a failure occurs, the debugging team may reason out the cause of the failure. They will have a close look at the code to find out the possible errors that might have caused the fault. In this process, sometimes they may also find and correct some more faults in addition to the fault that caused the failure. Subburaj, Gopal and Kapur [7] call this pattern as efficient debugging phenomenon since subsequent to one failure more than one fault are detected and corrected due to the initiative, resourcefulness, competence and efficiency of the debugging team. Due to this reason, the cumulative number of failures at infinite testing time may be less than the total number of faults detected, since some faults would have been detected without causing a failure. The objective of this research has therefore been further enhanced to mathematically derive a generalized NHPP SRGM that will be able to describe the software failure data adequately when any one of the three types of quality of debugging as discussed above are witnessed namely; imperfect debugging, perfect debugging and efficient debugging either in the presence or in the absence of the learning phenomenon of the testing team. Above all the proposed model will also quantify the learning and debugging indices observed in the project so as to help in further improvement of the processes.
This paper is organized as under. In Section 2, we derive the proposed SRGM and discuss about the methodology for estimation of its parameters for a given data set. In Section 3, we discuss about the goodness of fit tests carried out on the proposed model as well as comparison of performance of the proposed model with a similar model namely; Kapur and Garg [2] model. The vital quality metrics provided by the model are highlighted in Section 4. A better predictive ability of the proposed model has been established and presented in Section 5. Furthermore, a case study illustrating one application of the model is given in Section 6. Conclusions are given in the last section.

2 The proposed SRGM

2.1 Non-Homogenous Poisson Process Models

The Non-Homogenous Poisson Process (NHPP) models are widely used by the practitioners. G–O proposed a simple NHPP model assuming perfect debugging of faults and the absence of the learning phenomenon in the testing team. It is observed that the learning phenomenon of the testing team is present in some software projects, but may not be in all projects. Therefore, the SRGMs should be able to describe software failures data adequately irrespective of whether the learning phenomenon is present or not. In order to achieve this and enable the SRGM to be less sensitive to wide fluctuations in time between failures, Subburaj and Gopal [8] generalized the Goel and Okumoto [3] model.

2.2 NHPP Model with Modified shifted Weibull function ROCOF

It is to be noted that according to Farr [9], the mean value function \( \mu(t) \) of Poisson type models is given by the following equation:

\[
\mu(t) = aF(t)
\]

(1)

where \( a \): a constant and it represents the eventual number of faults (failures) detected in the system if it could have been observed over an infinite period of time.

\( F(t) \): cumulative distribution function (cdf) of the time to failure of individual faults.

With the perfect debugging assumption, it is assumed that each failure leads to correction of one fault perfectly in the software system under test. In this case, the failure intensity function \( \lambda(t) \) of NHPP model, which is a derivative of the mean value function \( \mu(t) \) can be written as follows:

\[
\lambda(t) = \left[(a - \mu(t))h(t)\right]
\]

(2)

where \( a \): the number of faults in the software system at the commencement of testing

\( \mu(t) \): mean value function at time \( t \) for failures

\( h(t) \): failure occurrence rate per fault i.e., hazard rate.

The above equation can be written as

\[
\lambda(t) = \frac{d\mu(t)}{dt} = (a - \mu(t))\frac{f(t)}{1 - F(t)}
\]

(3)

where \( f(t) \): probability density function (pdf) of the time to failure of individual faults

\( F(t) \): cumulative distribution function (cdf) of the time to failure of individual faults

The above equation based on perfect debugging assumption, conforms to one of the basic assumptions of NHPP models i.e., the failure intensity function at any time is proportional to the remaining number faults in the software system.

But in practice the fault correction may be imperfect in some projects leading to the number of failures to be greater than the number of faults at any time including at infinite time. It may be efficient in some projects i.e., more than one fault is corrected on the
average per failure. This leads to the situation when the number of failures is lesser than
the number of faults at any time including at infinite time. The quality of debugging in a
software project therefore can be classified in to the following:

- Imperfect debugging
- perfect debugging
- efficient debugging

Let us denote the index for quality of debugging or simply debugging index as \( c \). It will
be equal to 1 in the case of perfect debugging; will be less than 1 in the case of imperfect
debugging and greater than 1 in the case of efficient debugging. Thus we introduce an
index \( c \) to indicate the quality of debugging which may vary from project to project. When
we take in to account the quality of debugging with debugging index \( c \), the number of
faults actually corrected at time \( t \) will be equal to \( c \mu(t) \). Therefore the equation (3) gets
modified as given below:

\[
\lambda(t) = \frac{d \mu(t)}{d(t)} = \frac{(a - c \mu(t))f(t)}{1 - F(t)}
\]  

(4)

Then,

\[
\int \frac{d \mu(t)}{(a - c \mu(t))} = \int \frac{f(t)d(t)}{1 - F(t)}
\]

(5)

From the above, the generic equation for mean value function \( \mu(t) \) can be derived as follows:

\[
\mu(t) = \frac{a}{c} \left[ 1 - \left(1 - F(t)\right)^{c} \right]
\]

(6)

In the above equation, by substituting \( F(t) \) corresponding to the distribution function
chosen to represent ROCOF, we can derive a family of SRGMs since no single SRGM
may be suitable in all circumstances. In fact, the early imperfect debugging models
proposed by Ohba and Chou [1] and Kapur and Garg [2] were based on exponential
function ROCOF. But these models defined \( c \) as a probability \( p \) and hence restricted it to a
maximum of 1, thus not giving a provision for addressing efficient debugging. Kapur and
Garg [6] indeed proposed another model to address efficient debugging.

In order to describe the learning phenomenon and various types of debugging
independently as well as together, the authors choose the shifted Weibull function
ROCOF for the proposed SRGM because of the flexibility of Weibull distribution having
increasing, decreasing and constant hazard rates unlike the exponential function which can
describe constant hazard rate only. The natural choice could have been a two parameter
Weibull function ROCOF owing to its simplicity. However, Subburaj and Gopal [10]
analyze the necessity to include location parameter in the Weibull function when used to
represent ROCOF of generalized NHPP models. They conclude that since the first
software failure rarely occurs at time \( t = 0 \), the Goel generalized NHPP model based on
two parameter Weibull function ROCOF [11] sometimes does not seem to describe the
software failure data adequately due to the assumption of zero for the location parameter
[10]. Therefore, Subburaj and Gopal proposed a shifted Weibull function (with three
parameters) ROCOF to the generalized NHPP model addressing the learning phenomenon
of the testing team[10]. This modification to the Goel generalized NHPP model [11]
seems to have improved the performance of the generalized NHPP model consistently, as
confirmed by the goodness of fit statistic and predictive validity metrics, when applied to
failure data sets of 11 software projects with widely varying characteristics [10]. Hence
the authors in this paper propose a generalized NHPP model with modified shifted
A Software Reliability Growth Model for Estimating Debugging and the Learning Indices

Weibull function ROCOF to add debugging index to the Subburaj and Gopal SRGM [10] proposed earlier. The equation for \( F(t) \) of shifted Weibull function is given below:

\[
F(t) = 1 - \exp \left( -\left( \frac{t - \gamma}{\theta} \right)^\beta \right)
\]  

Substituting the above in equation (6) we get the mean value function for failures of generalized NHPP model with modified shifted Weibull function ROCOF as given in equation (8).

\[
\mu(t) = \frac{a}{c} \left[ 1 - \exp \left( -c \left( \frac{t - \gamma}{\theta} \right)^\beta \right) \right] ; \quad c>0, \ t\geq\gamma, \ \theta>0, \ \beta>0
\]  

The above can be written as

\[
\mu(t) = \frac{a}{c} \left[ 1 - \exp \left( -c \left( \frac{t - \gamma}{\theta} \right)^\beta \right) \right] ;
\]  

where, \( a \): the eventual number of faults that will be detected over an infinite amount of time

\( \theta \): scale parameter

\( \beta \): shape parameter

\( \gamma \): location parameter

\( c \): debugging index.

The shape parameter \( \beta \) provides an indication of overall testing efficiency in the project or in other words the skills of the testers in the project. Therefore, \( \beta \) may be considered as the learning index. The lower the value of \( \beta \) the higher are the test efficiency and so better are the skills of the testing team. The scale parameter \( \theta \) indicates the reciprocal of failure occurrence rate per fault or in other words reciprocal of rate of error detection per fault. Therefore, \( \theta \) may be considered as the test intensity index. The lower the value of \( \theta \), the higher will be the test intensity.

It is to be noted that the cumulative number of failures

\[
\mu(t) = 0 \quad \text{for} \ 0 < t \leq \gamma
\]

\[
= \frac{a}{c} \quad \text{for} \ t \rightarrow \infty
\]

In the case of perfect debugging both the number of faults and failures detected at infinite time will be equal to \( a \). If imperfect debugging is witnessed in the project, number of failures at infinite testing time will be greater than \( a \). If efficient debugging is witnessed in the project, number of failures at infinite testing time will be lesser than \( a \).

The failure intensity function, \( \lambda(t) \) is the rate of change of mean value function over time or the number of failures per unit time and hence it is the derivative of mean value function for failures with respect to time and is an instantaneous value [9]. It is given by:

\[
\lambda(t) = a \left( \frac{\beta}{\theta} \right) \left( \frac{t - \gamma}{\theta} \right)^{\beta-1} \left[ \exp \left( -c \left( \frac{t - \gamma}{\theta} \right)^\beta \right) \right]
\]  

2.3 Estimation of Parameters of the Model

The parameters of the model for a given failure dataset can be estimated by the method of least squares by using software tools. The NCSS Statistical Software, Utah, USA
(http://www.ncss.com) was used to estimate the values of the model parameters using this method. In the tool, curve fitting methodology was chosen to find an appropriate mathematical model which provides the relationship between a dependent variable \( y=\mu(t) \) and a single independent variable \( x=t \) by estimating the values of its parameters [12].

3. Performance Evaluation of the Proposed Model

The performance of an SRGM is assessed by its ability to fit the past failure data (goodness of fit) (GoF) and to predict time of occurrence of failures in the future satisfactorily (predictive validity) [13]. We discuss about goodness of fit measures in this section and predictive validity in Section 5. In order to check the GoF of the model, we chose the following failure data sets:

- Failure data sets, Stratus-1 and Stratus-2, each representing several hundred years of customer exposure on an operating system release with one million lines of code that supports fault tolerant hardware, given by Mullen [15] in the Appendix to his paper.

A number of metrics have been evolved over the years for finding Goodness of Fit (GoF) of a model for a given data set and some of them are given below:

- Coefficient of Determination \( R^2 \)
- Mean of Square Fitting Faults (MSF)

3.1 Coefficient of Determination \( R^2 \)

Coefficient of determination is a well-known measure of goodness of fit in a regression analysis and denoted as \( R^2 \). It represents the proportion of data that is closest to the line of best fit. The value of \( R^2 \) may vary from 0 to 1. If the model fits the data perfectly then \( R^2 = 1 \). If the model does not fit the data at all then \( R^2 = 0 \). The closer \( R^2 \) is to 1, the better is the fit.

3.2 Mean of Square Fitting Faults

Mean of Square Fitting Faults (MSF) [13] is given by:

\[
MSF = \left( \frac{1}{n} \right) \sum_{i=1}^{n} \left( x_i - E_i \right)^2
\]  

(11)

where \( n \) denotes the number of data points, \( x_i \) is the actual number of failures observed at time \( t_i \), and \( E_i \) is the number of failures estimated by a model to occur at the same instant of time \( t_i \). MSF is not a normalized measure of goodness of fit in contrast with \( R^2 \). MSF can only be used to compare the goodness of fit of different models for the same data set. If it is used to compare goodness of fit with different data sets it can be misleading. This is due to the fact that failure occurrence rate for each data set will be different. Even if the percentage error in estimation for different data sets is the same, MSF will be higher in case where failure occurrence rate is higher. Therefore \( R^2 \) is a better measure of goodness of fit across different models and data sets.

3.3 Performance Comparison with Classical Imperfect Debugging Model

We now compare the performance of the proposed model with Kapur and Garg [2] imperfect debugging model. We use \( R^2 \) and MSF to compare goodness of fit of the models. \( R^2 \) is used to compare goodness of fit across data sets. The MSF is only to
compare the goodness of fit of the two models for a given data set. The proposed model has a parameter $c$ indicating the debugging index whereas the K-G model has a parameter $p$ to indicate the probability of perfect debugging. Even when the project witnesses efficient debugging the $p$ value will be 1 since $p$ is a probability. However the proposed model will clearly distinguish efficient debugging with a $c$ value of more than 1. The performance comparison is given in Table 1.

**Table 1:** Comparison of Proposed Model with Kapur - Garg Imperfect Debugging Model

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Proposed Model $R^2$</th>
<th>MSF</th>
<th>$c$</th>
<th>K-G Model $R^2$</th>
<th>MSF</th>
<th>$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratus 1</td>
<td>0.99</td>
<td>45.8</td>
<td>0.7</td>
<td>0.97</td>
<td>195</td>
<td>0.75</td>
</tr>
<tr>
<td>Stratus 2</td>
<td>0.99</td>
<td>22.1</td>
<td>0.88</td>
<td>0.95</td>
<td>71.6</td>
<td>0.84</td>
</tr>
<tr>
<td>Musa P1</td>
<td>0.99</td>
<td>5.2</td>
<td>1</td>
<td>0.94</td>
<td>5.5</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>0.98</td>
<td>1.7</td>
<td>1</td>
<td>0.94</td>
<td>4644</td>
<td>1</td>
</tr>
<tr>
<td>P14 C</td>
<td>0.99</td>
<td>1.2</td>
<td>1.14</td>
<td>0.94</td>
<td>9.3</td>
<td>0.85</td>
</tr>
<tr>
<td>SS2</td>
<td>0.99</td>
<td>23.5</td>
<td>0.58</td>
<td>0.97</td>
<td>94.2</td>
<td>0.57</td>
</tr>
<tr>
<td>SS3</td>
<td>0.99</td>
<td>37</td>
<td>1.2</td>
<td>0.98</td>
<td>69</td>
<td>1</td>
</tr>
</tbody>
</table>

In Table 1, $R^2$ values are found to be very close to 1 for all the data sets for the proposed model, which confirms the flexibility of the model to fit adequately in all cases considered. From the above table it seems that the proposed model provides better goodness of fit for all data sets. Essentially, Kapur and Garg model [2] being an exponential model is not specifically meant to describe data sets with the learning phenomenon of the testing team, as revealed by relatively higher MSF figures. In fact, this is the cause for the Kapur and Garg model [2] realizing imperfect debugging phenomenon when it is not the case as in Musa’s [14] data set P14C where actually efficient debugging was observed. Now let us peruse the $p$ and $c$ values. In data set SS3 efficient debugging is witnessed, and hence $c$ value is 1.2 in the case of proposed model and the corresponding $p$ value is 1 in the case of K-G model. In case of data sets P1 and P2 both the models estimate perfect debugging. In case of three other data sets both models indicate imperfect debugging.

A comparison of the variation of failure intensity with time for both the above models in respect of Musa’s [14] data set SS2 is given in Figure 1.
As expected, the Kapur and Garg model [2] seems to miss the initially increasing and then decreasing pattern of failure intensity variation arising out of the learning phenomenon witnessed in the data set thus resulting in poor goodness of fit. However, the proposed model seems to capture the failure intensity variation accurately. Thus missing the learning phenomenon witnessed in a dataset seems to affect goodness of fit statistic of the K-G model.

4. Vital Quality Metrics

The very purpose of the model is to derive vital quality metrics. Table 2 contains a number of important metrics provided by the proposed model:

- $c$, the debugging index revealing the quality of debugging after a failure is observed during testing
- $\beta$, if greater than 1, indicates the existence of the learning phenomenon in the testing
- Remarks give a summary of the nature of the project

Table 2: Vital Quality Metrics Provided by the Model

<table>
<thead>
<tr>
<th>Data set</th>
<th>$\beta$</th>
<th>$a$</th>
<th>$c$</th>
<th>$N = a/c$</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratus 1</td>
<td>0.7</td>
<td>451</td>
<td>0.7</td>
<td>644</td>
<td>No learning, imperfect debugging</td>
</tr>
<tr>
<td>Stratus 2</td>
<td>0.59</td>
<td>301</td>
<td>0.88</td>
<td>342</td>
<td>No learning, imperfect debugging</td>
</tr>
<tr>
<td>P1</td>
<td>0.59</td>
<td>185</td>
<td>1</td>
<td>185</td>
<td>No learning, perfect debugging</td>
</tr>
<tr>
<td>P2</td>
<td>0.54</td>
<td>121</td>
<td>1</td>
<td>121</td>
<td>No learning, perfect debugging</td>
</tr>
<tr>
<td>P14 C</td>
<td>1.88</td>
<td>40</td>
<td>1.14</td>
<td>35</td>
<td>Learning, efficient debugging</td>
</tr>
<tr>
<td>SS2</td>
<td>1.5</td>
<td>196</td>
<td>0.58</td>
<td>338</td>
<td>Learning, imperfect debugging</td>
</tr>
<tr>
<td>SS3</td>
<td>1.27</td>
<td>375</td>
<td>1.2</td>
<td>312</td>
<td>Learning, efficient debugging</td>
</tr>
</tbody>
</table>

Note:

$a$: Total estimated number of faults in the software system before commencement of system testing

$N$: Total estimated number of failures

$c$: debugging index

The debugging index $c$ takes on different values ranging from 0.58 to 1.2. Musa [14] P14C, SS2 and SS3 witnessed the learning phenomenon of the testing team. It is interesting to note that one set of three projects witnessed the learning phenomenon of the testing team and another set of three projects witnessed imperfect debugging. The sets are not mutually exclusive. For instance, Musa’s project SS2 [14] witnessed both the phenomena. It is also noted that two projects each witnessed perfect debugging and efficient debugging. In Table 2 the quality of the development process can be assessed by the value of $\beta$, the number of faults estimated to lie in the software system before the commencement of testing. The value of $\beta$ indicates the quality of the testing process or the learning index. The debugging index $c$ indicates the quality of the debugging process. Thus, the model provides the vital few quality metrics needed by any software organization and is able to throw a lot of information as to how the development, testing and debugging went on in the project.

5. Predictive Validity

The predictive validity metrics are used to evaluate forecasting quality of the SRGMs. Goodness of Fit (GoF) test does not measure predictive validity of a model. It is possible to have an SRGM which fits the later behavior better, but not the earlier and such a model...
will give better GoF, but poor predictive validity [13]. The long-term predictive validity gives the relative error between estimated and observed values at the end of observation time window i.e., the final observation, while estimating model parameters for the data set at different points in time \((t_i)\). The model parameters were estimated at 1/6, 2/6, 3/6, 4/6 and 5/6 of the total observation time window. The long-term predictive validity for the proposed model for Musa’s data set P1 is given below in Table 3.

<table>
<thead>
<tr>
<th>Fraction of total observation time</th>
<th>RPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/6</td>
<td>-0.01</td>
</tr>
<tr>
<td>2/6</td>
<td>0.09</td>
</tr>
<tr>
<td>3/6</td>
<td>0.09</td>
</tr>
<tr>
<td>4/6</td>
<td>-0.07</td>
</tr>
<tr>
<td>5/6</td>
<td>-0.07</td>
</tr>
</tbody>
</table>

The above table confirms adequate predictive validity of the proposed model.

6. Case study – One Application of the Model

This case study provides one application of the model; namely software release time determination \(i.e.,\) when to stop testing. Subburaj and Gopal [10] highlight the importance of release time and its determination based on achieved failure intensity and total cost. We will apply this criterion to determine release time for Musa P1 data set [14] with a target failure intensity of 0.002 failures per second using the proposed model.

In the Musa P1 data set, in the first phase of testing 36 failures were observed. The model parameters were estimated with 36 failures. Then the optimum testing time \((T_{op})\) required for achieving the target failure intensity of 0.002 failures per second was found to be 19250 seconds. In 19250 seconds of testing time as per the data set Musa P1 [14], 78 failures were observed. With the updated failure data up to 78 failures, model parameters were estimated again and optimum testing time required for achieving the target failure intensity of 0.002 failures per second was found to be 31000 seconds. 91 failures were observed in 31000 seconds of testing time as per the data set [14]. With the updated failure data up to 91 failures, model parameters were estimated again and optimum testing time required for achieving the target failure intensity of 0.002 failures per second was found to be 31,250 seconds. As per the data sheet no failures occurred from 31,000 seconds to 31,250 seconds and thus the testing can be stopped after observing 91 failures, having reached the target failure intensity of 0.002 failures per second. The results are summarized in Table 4.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Time in seconds</th>
<th>No. of failures</th>
<th>(a)</th>
<th>(\theta)</th>
<th>(\beta)</th>
<th>(c)</th>
<th>(T_{op})</th>
<th>(R^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>5389</td>
<td>36</td>
<td>185</td>
<td>2.8</td>
<td>91289</td>
<td>0.563</td>
<td>1</td>
<td>19250</td>
</tr>
<tr>
<td>II</td>
<td>19250</td>
<td>78</td>
<td>185</td>
<td>2.8</td>
<td>44957</td>
<td>0.716</td>
<td>1</td>
<td>31000</td>
</tr>
<tr>
<td>III</td>
<td>31000</td>
<td>91</td>
<td>185</td>
<td>2.8</td>
<td>48442</td>
<td>0.710</td>
<td>1</td>
<td>31250</td>
</tr>
</tbody>
</table>
Thus the proposed model has been used to get a clear and unambiguous software release time based on the chosen criteria for a practical data set. Since $c$ is 1, the number of faults will be equal to the number of failures which is 185. Since we stopped testing when we observed 91 failures and since $p$ equals 1, the number of faults corrected is also 91. The number of faults still remaining in the software system when we exit the system testing is 94. Thus the model facilitates obtaining one more quality metrics i.e., faults still remaining in the software system at the release time.

7. Summary and Conclusions

The classical software reliability growth models assume perfect debugging. A few SRGMs were proposed later to address imperfect debugging phenomenon and recently few more models were proposed to address imperfect debugging tightly integrated with the learning phenomenon of the testing team. It is essential that an SRGM addresses both imperfect debugging and the learning phenomenon independently since in practice the phenomena are independent of each other. Furthermore, a software development organization may need to collect vital few quality metrics such as quality of testing/debugging and the number of faults in the software system at the beginning as well as at the end of system testing i.e., at the time of release of the software system for use. In order to facilitate collection of such additional quality metrics, the authors propose a generalized NHPP model with modified shifted Weibull function ROCOF. This model also enables estimation of reliability growth with testing, determining testing time needed to achieve the required failure intensity objective and the number of failures to be observed before the target failure intensity will be achieved as revealed by the case study given in the paper. In this paper a generalized approach to derive such models mathematically is also given using which a family of SRGMs can be derived in the future since no single SRGM may be adequate in all circumstances in a realistic environment.

References

A Software Reliability Growth Model for Estimating Debugging and the Learning Indices


R. Subburaj is currently Professor and Consultant in the Department of Information Technology, SRM University, Chennai. He played a key role in the Department of Electronics and Information Technology/STQC Directorate of the Government of India for nearly three decades. He Headed Electronics Test & Development Centre, Chennai, Centre For Reliability, Chennai and Electronics Regional Test Laboratory (West), Mumbai. He was authorized by Carnegie Mellon University/Software Engineering Institute, USA to teach their official Intro’ to Software CMM course. He was awarded IETE Lal C Verman Award for the year 2003 for his distinguished contributions in the field of Standardization, Quality Control and Precision measurements. He was awarded M.Tech. degree by Indian Institute of Technology (IIT), Delhi, India and awarded Ph.D. degree by University of Madras, Chennai, India. He was the Conference Chair of the second International Conference on Reliability and Safety Engineering – INCRESE 2006.

G. Gopal is Professor and Head of the Department of Statistics, University of Madras, Chennai. His current areas of research are Majorization, Reliability and survival analysis. He has to his credit about 25 research papers in leading national / International journals. He has guided so far 7 PhD scholars. He has also introduced Actuarial Science in the post graduate program in the University of Madras. The Actuarial Society of India has conferred him with the title Hon.Fellow of Actuarial Society of India (Hon.FASI) for promoting actuarial education in India. He obtained his Master’s degree and Ph.D. in Statistics from the University of Madras, Chennai, India.

P. K. Kapur is Professor & HOD Research at Amity International Business School, Amity University, Noida and former Head of the Department of Operational Research and former Dean of the Faculty of Mathematical Sciences, University of Delhi. He has been the president of Society for Reliability Engineering, Quality and Operations Management (Regd.) since 2000 and former president of Operational Research Society of India. He obtained his Ph.D. degree in Reliability Theory (Operational Research) from University of Delhi in 1977. He has published extensively in Indian journals and abroad in the areas of Marketing, Hardware Reliability, Optimization, Queuing Theory and Maintenance and Software Reliability (more than 200 papers). He is author of 2 renowned books on Software Reliability, Software Reliability Assessment with O.R. Applications, Springer UK (2011) and Contributions to Hardware and Software Reliability, World Scientific, Singapore (1999).