Modeling Successive Software Up-gradations with Faults of Different Severity

P.K. Kapur¹, Adarsh Anand² and Ompal Singh³

Department of Operational Research, University of Delhi-110007
¹pkapur1@gmail.com, ²adarsh.anand86@gmail.com and ³ompalsingh@yahoo.com

ABSTRACT

The intense global competition in the dynamic environment has lead to up-gradations of software product in the market. The software developers are trying very hard to recover to project themselves as organizations that provide better value to their customers. One major way to increase the market presence is by offering new functionalities in the software periodically. To get the competitive edge it is critical to know the appropriate release of the software. The timing decision depends on whether companies are able to remove all the types of the faults lying inside the software. To capture the effect of faults due to existing software and those left in the software (due to their nature) at various points in time, we develop an up-gradiation multi release software reliability growth model. This model uniquely identifies the faults left in the software when it is in operational phase during the testing of the new code i.e. developed while adding new features to the existing software. We examine the case where there exists two types of faults in the software; simple and hard and during testing the simple faults are removed by exponential rate whereas hard faults are removed by logistic rate. Results are supplemented by a numerical example.

KEYWORDS

Multi Up-gradation, SRGM, Fault Severity.

1. INTRODUCTION

In today’s time, software is taking a dual role (Pressman 2001), on one side it acts as a product and, on the other hand, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is information transformer-producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), communication of information (networks), and the creation and control of other programs (software tools and environments). Software’s impact on our society and culture continues to be profound. As its importance grows, the software community continually attempts to develop technologies that will make it easier, faster, and less expensive to build high-quality computer programs. Because of this stiff competition, the software developers are trying very hard to survive in the market by adding some new technology to the existing software period wise. Technological breakthroughs are happening rapidly and these new innovations often take form of a new product. It has been seen that in the initial period of the software more efforts are put increasingly so that overall performance of the technology can be improved till attaining its natural performance limit. The concept of performance of a new technology generation over its life cycle has been explained by using well known s-shaped curve or sigmoid curve [2,3,9]. In general when software reaches a level when it attains it operational reliability level desired by the firm, a new version is introduced and the software gets upgraded. The term upgrade refers to the replacement of a product with a newer version of the same product.

Abundance of software reliability models have been developed in the history of this subject. Goel and Okumoto [1,2,3,12,14] proposed an SRGM, which describes the fault detection rate, as a non homogeneous Poisson process (NHPP) assuming that hazard rate is proportional to remaining fault number. The basic assumptions of their model were as follows:-

1. Software systems are subject to failure during execution caused by a fault remaining in the system.
2. Failure rate of the software is equally affected by the faults remaining in the software.
3. The number of faults detected at any time is proportional to the remaining number of faults in the software.
4. On a failure, repair effort starts and the fault is removed with certainty.
5. All faults are mutually independent from failure detection point of view.
6. The proportionality of fault detection/isolation/correction is constant.
7. The fault detection/ correction are modeled by non homogeneous poison process.
8. The number of faults in the beginning of the testing phase is finite.

Last two decades have given Software Reliability models which shows the relationship between the testing time and the corresponding number of faults removed. They are either Exponential or S-shaped or a mix of the two [3,8,9,10,14,15]. In this paper we have emphasized on the fact that the software includes different types of faults, and each fault requires different strategies and different amounts of testing effort to remove it. Obha [3] refined the Goel-Okumoto model by assuming that the fault detection/removal rate increases with
time and that there are two types of faults in the software. SRGM proposed by Bittanti et al. [1,2] and Kapur and Garg [8] has similar forms as that of Obha [3,14] but is developed under different set of assumptions but all are flexible in nature. 

The typical software failure curve experienced by traditional software reliability growth model can be depicted by the following Figure 1. Thus the traditional software reliability growth model fails to capture the error growth due to the software enhancements in user-end. In the useful-life phase as the software firm introduced new add-ons or features on the basis of the user needs, software will experience a drastic increase in failure rate each time an upgrade is made. The failure rate levels off gradually, partly because of the defects found and fixed after the upgrades. Figure 2 depicts the increase in failure rate due to the addition of new features in the software. Due to the feature updation, the complexity of software is likely to be increased as the functionality of software is enhanced. Even fixing bugs may induce more software failures by fetching other defects into software. But if the goal of the firm is to upgrade the software by enhancing its reliability then it is possible to incur a drop in software failure rate that can be done by redesigning or re-implementing some modules using better engineering approaches.

The paper is organized as follows. Sec 2 gives an outlining about the symbols used, Sec 3 discusses about the Software Reliability Modeling taking into account the modeling of multi upgradation for each release. In Sec 4 we have discussed about the data set and parameter estimation results. Finally, a small discussion about the results in Sec 5 and conclusion has been drawn in Sec 6, followed by References in Sec 7.

2. NOTATIONS

\[ m_i(t) : \text{ Expected number of faults removed in the time interval}\]
\[ (0,t) \text{ for } i^{th} \text{ release} \]
\[ a_i : \text{ Constant, representing the number of faults lying dormant in the software at the beginning of testing of each release} \]
\[ \beta, b : \text{Constants} \]
\[ \beta_{i-1} : \text{Logistic learning factor for release} \]
\[ (i = 2 \text{ to } 4) \]
\[ b_i : \text{Fault detection rate for Type } i \text{ in} \]
\[ 2,3 \text{ and } 4^{th} \text{ release } (i = 1 \text{ to } 2) \]
\[ \lambda_i : \text{fraction of simple faults of i}^{th} \text{ release; } i=2,3,4. \]
\[ (1-\lambda_i) : \text{fraction of hard faults of i}^{th} \text{ release; } i=2,3,4. \]
\[ \delta_{i-1} : \text{fraction of remaining simple faults of just previous release; } i=2,3,4 \]
\[ (1-\delta_{i-1}) : \text{fraction of remaining hard faults of just previous release; } i=2,3,4 \]
\[ \delta_i \text{ and } (1-\delta_i) : \text{the fraction of remaining faults from the first release.} \]
\[ F_{ij}(t) : \text{Probability distribution function for type } j \text{ fault of } i^{th} \text{ release; } \]
\[ i = 1, 2, 3, 4 \text{ and } j = 1, 2 \]

3. SOFTWARE RELIABILITY MODELING

Using the hazard rate approach in deriving the mean value function of cumulative number of faults removed, we have:

Let \( \{N(t), t \geq 0\} \) be a counting process representing the cumulative number of software failures by time \( t \). The \( N(t) \) process is shown to be a NHPP with a mean value function \( m(t) \). Mean value function represents the number of faults removed my time \( t \).
Modeling Successive Software Up-Gradations with Faults of Different Severity

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

And
\[ m(t) = \int_0^t \lambda(x) \, dx \]
\[ \frac{dm(t)}{dt} = f(t) (1 - F(t) (a - m(t))) \]

Solving the above equation using the initial condition \( m(0) = 0 \) we get
\[ m(t) = a F(t) \] (1)

Out of several SRGM developed in literature some depict exponential reliability growth whereas others show S-shaped growth, depending on the nature of growth phenomenon during testing. They are broadly classified in these two categories. If the growth is uniform, generally exponential models have been used and for non-uniform growth, S-shaped models have been developed. Besides, models which can capture variability in exponential and S-shaped curves have also been developed and they have been termed as flexible growth models. As S-shape in reliability can be ascribed to different reasons, therefore as many models exist in the literature at times leading to confusion in model selection from the plethora of models available.

Most of these models were proposed under the assumption that similar testing efforts and testing strategy is required for removing each of the faults. However this assumption may not be true in practice, different faults may require different amount of testing efforts and testing strategy for their removal from the system. In literature, to incorporate this phenomenon, faults are categorized as of different types and are analyzed separately. Yamada, Osaki and Narithisa [16] proposed a modified exponential SRGM assuming that there are two types of faults in the software and exponential failure curve. Pham [2] proposed a SRGM with multiple failure types. Later Kapur, Younes and Agarwala [10] have also developed such a model called the generalized Erlang SRGM by classifying the faults in the software system as simple, hard and complex faults. It is assumed that the time delay between the failure observation and its subsequent removal represent the severity of faults. Another model due to Kapur, Bardhan and Kumar [2000] describes the implicit categorization of faults based on the time of detection of fault. However an SRGM should explicitly define the errors of different severity as it is expected that any type of fault can be detected at any point of testing time. Therefore it is desired to study the testing and debugging process of each type of fault. It is this combined effect of all the above discussed modes that we have used in the paper and considered that simple faults are being removed quickly with a different rate and hard faults are being removed by a different rate. The mean value function of the SRGM is described by the joint effect of the type of faults present in the system. Such an approach can capture the variability in the reliability growth curve due to the errors of different severity depending on the testing environment. Therefore equation (1) can be actually written as a combination of two types of faults removed as:

\[ m_i(t) = \lambda_i a_i F_s(t) + (1 - \lambda_i) a_i F_h(t) \]

where; \( a_i = \lambda_i a_i + (1 - \lambda_i) a_i \)

\( F_s(t) \) and \( F_h(t) \) : Probability distribution function for simple and hard faults

\[ F_s(t) = (1 - e^{-bt}) \]
\[ F_h(t) = \left( \frac{1 - e^{-bt}}{1 + \beta e^{-bt}} \right) \]

3.1 MODELING MULTI-UPGRADATION FOR EACH RELEASE

3.1.1 Release 1

In the software reliability growth phase, the software testing process determines the nature of the failure data. Before releasing the software, it has to be made bug free. It is the duty of the testing team to make sure that before the release of the software in the market they remove maximum number of bugs from the software. Practically it is not possible to remove all the bugs in the software, therefore, when one software version is tested by the testing team, there are chances that they may detect a finite number of bugs in the code developed. These finite numbers of bugs are then removed by taking exponential rate for simple faults and logistic rate for hard faults. But the estimation results did not come out to be satisfactory. Infect the data analysis result was completely exponential in nature. Now, there can be many factors that affect software testing. These factors are unlikely to be kept stable during the entire process of software testing, with the result that the underlying statistics of the failure process is likely to experience major changes. The fault detection rate strongly depends on some parameters like skill of test team, program size, software testability, defect density and resource allocation. During a software testing process, there is a possibility that the underlying fault detection rate is changed at some time moment called Change Point. This would result in a software failure intensity function either increasing or decreasing monotonically. All the finite numbers of bugs are then removed by taking exponential rate for simple faults considering the effect of change point. Hence we use the mathematical equation for Release 1 as

\[ m_i(t) = a_i F_{1i}(t) \quad 0 < t < t_1 \]

(3)
\[ F_{11}(t) = \left(1 - e^{-b_2(t-\tau)}\right) \]

Where; \( \tau \) is the change point

and \( b_1, b_2 \) are the rates before and after \( \tau \)

3.1.2 Release 2

Technological changes and stiff competition forces the software developer to add certain new features to the software. Adding new functionality to the software leads to change in the code. These new specifications in the code lead to increase in the fault content. Now the testing team starts testing the upgraded system, besides this the testing team considers dependency and effect of adding new functionalities with existing system. While testing the newly formed code, there is always a possibility that the testing team may find some faults which were present in previously developed code. Also, the testing team realizes that not all the faults were of similar character, hence while upgrading the software, the testing team has now in mind that faults present in the software can be categorized into “simple” and “hard” faults according to their severity. Therefore, in this period the leftover faults of the first release i.e. \( a_1(1-F_{11}(t_i)) \) interact with the newly identified simple and hard detection rates respectively. As a result of these interactions a fraction of faults which were not removed during the testing of the first version of the product gets removed. In addition, faults are generated due to the enhancement of the features, a fraction of these faults are also removed during the testing with new detection rate i.e. \( F_{21}(t-t_1) \) for simple faults and \( F_{22}(t-t_1) \) for hard faults.

The change in the fault detection is due to change in time, change in the complexity due to new features, change in testing strategies etc. The resulting equation can be written as

\[ m_2(t) = \lambda_2 a_2 F_{21}(t-t_1) + \lambda_3 a_3 F_{22}(t-t_1) \]

\[ + \delta_2 a_1(1-F_{11}(t_i)).F_{21}(t-t_1) \]

\[ + (1-\delta_2) a_1(1-F_{11}(t_i)).F_{22}(t-t_1) \]

3.1.3 Release 3

In this period, the leftover simple faults i.e. faults \( (1-\lambda_2) a_2 (1-F_{21}(t_2-t_1)) \) and hard faults \( (1-\lambda_3) a_3 (1-F_{22}(t_2-t_1)) \) of the second version interact with new simple and hard detection/correction rate. A fraction \( \delta_2 \) of remaining simple faults interact with the new simple rate and the remaining fraction \( (1-\delta_2) \) of simple faults interact with the new hard rate, while the leftover of the hard faults interact with the new hard rate only because of faults severity. As a result of these interactions a fraction of faults gets removed. On similar basis as in second release, we find that faults are generated due to new add-ons, therefore a fraction of these faults are also removed during the testing with new detection rate i.e. \( F_{31}(t-t_2) \) for simple faults and \( F_{32}(t-t_2) \) for hard faults and the corresponding mathematical equation can be represented as follows:

\[ m_3(t) = \lambda_4 a_4 F_{31}(t-t_3) + \lambda_3 a_3 F_{32}(t-t_3) \]

\[ + \delta_3 a_1(F_{21}(t_3-t_2)).F_{31}(t-t_3) \]

\[ + (1-\delta_3) a_1(F_{21}(t_3-t_2)).F_{32}(t-t_3) \]

\[ + (1-\lambda_3) a_3(F_{22}(t_3-t_2)).F_{32}(t-t_3) \]

\[ ; t_3 < t < t_4 \]

4. MODEL VALIDATION, DATA SET AND DATA ANALYSIS

Figure (1-4) shows the estimated and the actual values of the number of faults removed for four releases. To check the validity of the proposed model and to describe the software reliability growth, it has been tested on tandem computer [6] four release data set. Also we have used non linear least square technique in SPSS software for estimation of parameters. Estimated value of parameters of each releases are given in Table 1. Table 2 shows the comparison criterion of the four software releases. Based on data available given in Table1, the performance analysis of proposed model is measured by the four common criteria that we define as below:

4.1 CRITERIA FOR COMPARISON

To give quantitative comparisons, some criteria were used to judge the performance of the proposed model. Here we let \( n \) represent the sample size of selected data set, \( m_i \) represent the actual number of faults by time \( t_i \) and \( m(t_i) \) represent the estimated number of faults by time \( t_i \) indicate lower value indicate less fitting error.

1. The Bias is defined as:

\[ Bias = \frac{1}{n} \sum_{i=1}^{n} \left( m(t_i) - m_i \right) \]
The difference between the observation and prediction of number of failures at any instant of time $i$ is known as PE$_i$(prediction error). The average of PEs is known as bias. Lower the value of Bias better is the goodness of fit.

2. The **Variation** is defined as:
   \[
   \text{Variation} = \sqrt{\frac{1}{N-1} \sum (PE_i - \text{Bias})^2}
   \]
   The average of the prediction errors is called the prediction Bias, and its standard deviation is often used as a measure of the variation in the predictions.

3. The **Root Mean Square Prediction Error (RMSPE)** is defined as:
   \[
   \text{RMSPE} = \sqrt{\text{Variance}^2 + \text{Bias}^2}
   \]
   RMSPE is a measure of the closeness with which the model predicts the observation.

4. The **Mean Square Error (MSE)*** is defined as:
   The difference between the expected values, $\hat{m}(t_i)$ and the observed data $y_i$ is measured by MSE as follows
   \[
   MSE = \frac{1}{k} \sum (\hat{m}(t) - y_i)^2
   \]
   Where $k$ is the number of observations. The lower MSE indicates less fitting error, thus better goodness of fit.

Figures (1-4): Release 1,2,3,4 respectively

---

**Table 1**

<table>
<thead>
<tr>
<th>Release 1</th>
<th>Release 2</th>
<th>Release 3</th>
<th>Release 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>103.2</td>
<td>a</td>
<td>118.2</td>
</tr>
<tr>
<td>$b_1$</td>
<td>0.104</td>
<td>$b_1$</td>
<td>0.184</td>
</tr>
<tr>
<td>$b_2$</td>
<td>0.134</td>
<td>$b_2$</td>
<td>0.368</td>
</tr>
<tr>
<td>$\lambda_1$</td>
<td>0.329</td>
<td>$\lambda_2$</td>
<td>0.262</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>17.5</td>
<td>$\beta_2$</td>
<td>78.4</td>
</tr>
<tr>
<td>$\delta_1$</td>
<td>0.956</td>
<td>$\delta_2$</td>
<td>0.515</td>
</tr>
</tbody>
</table>

**Table 2**

<table>
<thead>
<tr>
<th>Release 1</th>
<th>Release 2</th>
<th>Release 3</th>
<th>Release 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>M.S.E</td>
<td>2.658</td>
<td>3.209</td>
<td>0.532</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0.996</td>
<td>0.997</td>
<td>0.999</td>
</tr>
<tr>
<td>Bias</td>
<td>0.190</td>
<td>0.129</td>
<td>0.158</td>
</tr>
<tr>
<td>Variation</td>
<td>1.666</td>
<td>1.835</td>
<td>0.744</td>
</tr>
<tr>
<td>RMSPE</td>
<td>1.677</td>
<td>1.840</td>
<td>0.760</td>
</tr>
</tbody>
</table>

**RESULT**

It was verified from the data analysis of the Release 1 that it did not gave suitable results but when the same problem was worked with change point concept the values of parameters improved drastically. The value of $\lambda$ is decreasing periodically (0.329 for Release 2, 0.262 for Release 3 and 0.252 for Release 4) showing that very less simple faults are being detected in prior release and that the upgraded version contains more hard faults which justifies the S-shape of these Releases. It can be
noticed that Release 3 has the maximum S-character followed by Release 2 and Release 4, whereas Release 1 was completely exponential with change point in it. Hence, the problem has been solved taking into consideration the change point concept for the Release 1.

CONCLUSION
Software products in general face a fierce competition in the market and therefore have to come up with upgraded versions of the software. A software fault may not always be of the same category, some faults are easy to remove (simple) and some are hard which needs more time and effort to leave the system. This leads the testing team to take different strategies to remove the faults lying dormant in the software. But this is not possible for them to remove the faults completely. Hence, some faults (both simple and hard) of the previous generation manifest themselves in the next release. In this paper we have proposed a multi up gradation model to capture the faults added at the time of introduction of the new product in the market and simultaneously remove the different types of faults left in the previous release. The value of parameters is reasonably good and estimated using original data and the goodness of fit value (Variance, RMSPE, MSE and bias) mentioned in Table 1 have decreasing behavior. Therefore, the proposed model fits the data reasonably well. The software reliability multi upgradation model in this paper is based on the classification of faults into two categories; simple and hard. The model can be extended to removal of faults with more complexity in them.

REFERENCES