Imperfect Debugging Software Reliability Growth Model for Multiple Releases

Anu.G Aggarwal¹, P.K.Kapur² and A.S.Garmabaki³
Department of Operational Research, University of Delhi, Delhi-110007
¹anuagg17@gmail.com, ²pkkapur1@gmail.com and ³garmabaki@gmail.com

ABSTRACT
Software Reliability Growth Model (SRGM) have been used for a long time to access the quality of the software being developed. But in today’s software environment of global competition where each company is trying to prove itself better than its competitors, software company have to continually do up-gradation or add-ons in their software to survive in the market. Each succeeding up-gradation offers some innovative performance enhancement or some new functionality etc distinguishing itself from the past release, but at the same time the amount of risk involved in up-gradation/add-ons of software with regard to introducing new faults or increasing the number faults in the software is also formidable. This model uniquely identifies the faults left in the previous operational release during the testing of the release new code i.e. code developed after adding new features to the existing software. Due to complexity and incomplete understanding of the software, the testing team may not be able to remove/correct the fault perfectly on observation/detection of a failure and the original fault may remain resulting in the phenomenon known as imperfect debugging, or get replaced by another fault causing error generation. In this paper we proposed new model and new concept for multi release software development environment. The model developed is validated on real data sets for software which has been released in the market with four times new features.

KEYWORDS
Software reliability, Up-gradation, Fault, Imperfect Debugging, NHPP.

1. INTRODUCTION
Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. Software reliability modeling has gained lot of importance in recent years. Critical software in some fields like Internet infrastructure, aerospace, telecommunication, military defense and medical applications has led to a tremendous increase in the amount of work being carried out in this area. Over the past three decades, many software reliability models with different parameters, reflecting various testing characteristics, have been proposed [1,2,3,4,5,10,11,12,23,24], and researcher introduced some realistic issues such as imperfect debugging and learning phenomena, fault severity, testing coverage etc. for estimating the reliability growth of software products, [1,3,9,12,13 ,14 ,15,16,18,27 ].

Measuring, Control and monitoring the software quality before its launch in the market are the most difficult problems being face by the software industry, hence software reliability becomes an important measure to help software managers with quantifying system behaviors and with this parameter we can determine release times and testing resources allocations. The intense global competition in the dynamic environment has led to a technological substitution of software product in market. The reputed software developing companies like Microsoft, IBM, Adobe and Wipro etc. are trying very hard to provide better value to their customers. They are trying to make their market presence by two major ways, one is by considering the bugs from the existing system i.e. reported bugs from the operational phase of the existing system and second by Upgradations/add-ons or by adding some new functionality to the existing software system periodically. 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. In general when software reaches a level when it attains its operational reliability level desired by the company, new upgraded software is introduced in the market.

Upgrading a software application is a complex task. The upgraded and existing system may differ in the performance, interface and functionality etc. Although the developers upgrades the software in order to improve the software product, which also includes the possibility that the upgrade version will worsen, That’s why there is risk involved into upgrading the software system. While upgrading an existing software system, only selected components of the software system are changed while the other will remain same to function. This process leads to an increase in the fault contents and the testing team is always interested in knowing the bugs present in the software which will decide the utility of up-graded software. Safe up-gradation can improve the behavior of the system and can preserve market for company, however risky up-gradation can cause critical error in system. For example in October 2005, a glitch in a software upgrade caused trading on the Tokyo Stock Exchange to shut down for most of the day [4,5], in 1991 after changing three lines code in a signaling program which contained millions lines of code, the local telephone systems in California and the eastern seaboard came to stop[6]. Similar gaffes have occurred from important government systems.
The behavior of failure intensity for software with multiple releases is not same as software without up-gradation. The typical software failure curve experienced by traditional software reliability growth model can be depicted by the figure 1. 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, software firm introduces new add-ons or features on the basis of the user need. Software will experience an increase in failure rate, each time an upgrade is made. The failure rate decreases 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. 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 [18].

Consider the faults of all previous releases. This takes less time of the testing team in comparison to test the complete software together (i.e. all releases together). The testing team may not be able to remove the fault perfectly on the detection of the failure and the original fault may remain or replaced by another fault because of the incomplete understanding of the internal structure of the software. While the first phenomenon is known as imperfect fault removal, the second is called error generation. In case of imperfect fault removal, the fault content of the software will not change but in case of error generation, the fault content increases as the testing progresses and removal results in introduction of new faults while removing old ones. Also Faults are categorized with respect to time which they take for isolation and removal after their observation. Faults are classified as “simple fault” if the time between their observation and removal is negligible else If more efforts and time is required for the removal of the fault is classified as “hard fault” [27].

The rest of this paper is organized as follows. Section 2 describes assumptions and notations for the modeling of software reliability multiple release model. Section 2.3 briefly reviews literature on Software Reliability with imperfect debugging and fault severity. In section 3, we propose a new up-gradation model. Section 4 shows the experimental results through real data sets. We also analyze about parameter in each release. Finally, conclusions and future direction are given in Section 5, 6 respectively.

2. MODELING OF THE MULTI UP-GRADATION SOFTWARE RELIABILITY

2.1. ASSUMPTIONS

1. The fault detection/correction are modeled by non homogeneous poison process (NHPP).

2. Software faults are two types, namely, Type I and Type II which have different severity, as simple and hard fault, respectively.

3. The number of faults detected at any time is proportional to the remaining number of faults in the software.

4. Failure introduction rate is equally affected by faults remaining in the software.

5. The number of faults in the beginning of the testing phase is finite.

6. All faults are mutually independent from failure detection point of view.

7. During the fault removal process, whether the fault is removed successfully or not, two type of imperfect debugging may happened, probability of perfect debugging

\[ \rho \] and faults are generated rate \[ \alpha \].
2.2. NOTATIONS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m(t) )</td>
<td>Expected mean number of faults removed by time ( t ).</td>
</tr>
<tr>
<td>( f(t) )</td>
<td>Probability density function for fault removal process.</td>
</tr>
<tr>
<td>( F(t) )</td>
<td>Probability distribution function for fault removal process.</td>
</tr>
<tr>
<td>( t_i )</td>
<td>Time for ( i^{th} ) release (i=1 to 4).</td>
</tr>
<tr>
<td>( a(t) )</td>
<td>Time dependent fault content function.</td>
</tr>
<tr>
<td>( a_i )</td>
<td>Initial fault content for ( i^{th} ) release (i=1 to 4).</td>
</tr>
<tr>
<td>( a_i^* )</td>
<td>Initial fault content for ( i^{th} ) release in presence of imperfect debugging.</td>
</tr>
<tr>
<td>( a )</td>
<td>Total Initial fault content in the software.</td>
</tr>
<tr>
<td>( b(t) )</td>
<td>Time dependent fault detection rate function.</td>
</tr>
<tr>
<td>( \rho_i )</td>
<td>Probability of error removal on a failure for ( i^{th} ) release (i=1 to 4).</td>
</tr>
<tr>
<td>( \rho )</td>
<td>Probability of error removal on a failure.</td>
</tr>
<tr>
<td>( \alpha_i )</td>
<td>Fault introduction rate for ( i^{th} ) release (i=1 to 4).</td>
</tr>
<tr>
<td>( \beta_i )</td>
<td>Logistic learning factor for ( i^{th} ) release (i=1 to 4).</td>
</tr>
<tr>
<td>( b_i )</td>
<td>Fault detection rate for Type ( i ) in each release (i=1 to 2).</td>
</tr>
<tr>
<td>( \beta, b, \lambda_i )</td>
<td>Constant</td>
</tr>
</tbody>
</table>

2.3 REVIEW OF SOFTWARE RELIABILITY MODELS BASED ON FAULT SEVERITY AND IMPERFECT DEBUGGING

In past three decades, many software reliability models with different type of fault introduced by researcher. Obha [8] developed a hyper-exponential software reliability growth model for a software system having different modules. He assumed the fault detection rate for each module to be different. Yamada, Osaki and Narithisa [26] proposed a modified exponential SRGM assuming that software contains two types of faults with different detection rates. Kimura, Yamada and Osaki [24] proposed exponential S-shaped model to capture different types of faults present in the software. Hence, it is a fact that different types of faults exist in the software. Kapur, Younes and Agarwala [19] introduced a flexible model called the generalized Erlang SRGM by classifying the faults in the software system as simple, hard and complex with the assumption that the time delay between the failure observation and its removal which represent the severity of faults. Another model due to Kapur, Bardhan and Kumar [15, 27], describes the implicit categorization of faults based on the time of detection of fault. However, an SRGM should explicitly define the different types of faults as it is expected that any type of fault can be detected at any point of testing time. Therefore, it is desired to study testing and debugging process of each type of faults separately. Kapur, Bardhan and Shatnawi [15, 20]. In real practice, it is important to know that how many types of faults exist in the software at any time, so that different testing strategy and testing effort can be applied to remove those faults. The mean value function of the SRGM is described by the joint effect of the types of faults present in the system. Faults are classified as “simple fault” if the time between their observation and removal is negligible else if more efforts and time is required for the removal of the fault is classified as “hard fault”. Goel et.al [2, 3] first introduced the concept of imperfect debugging. He introduced the probability of imperfect debugging. Model due to Chou and Obha [8] is an error generation model as applied on Goel-Okumoto model and has also been named as Imperfect debugging model. Kapur and Garg [16, 27] introduced the imperfect fault removal in Goel and Okumoto [3, 16]. They assumed that the fault detection rate per remaining faults is reduced due to imperfect fault removal. These models assumed that testing team may not be able to remove the fault perfectly on the detection of the failure and the original fault may remain or replaced by another fault.

Let \( N(t) \) \( t \geq 0 \) be a counting process representing the cumulative number of software failures by time \( t \). The counting process \( N(t) \) is shown to be a NHPP with a mean value function \( m(t) \) which represents the number of faults removed by time \( t \).

Based on the NHPP assumption, it can be shown that \( N(t) \) has Poisson distribution with mean \( m(t) \), i.e.,

\[
Pr\{N(t) = k\} = \frac{[m(t)]^k}{k!} e^{-m(t)} .
\]

By definition, the mean value function of cumulative number of failures, \( m(t) \), can be expressed in terms of the failure intensity function of the software, i.e.,

\[
m(t) = \int_0^t \lambda(s)ds .
\]

Based on previous assumptions the differential equation describing the removal phenomenon incorporating both types of imperfect debugging can be given by:

\[
\frac{dm(t)}{dt} = \rho b(t) (a(t) - m(t)) .
\]

It is assumed that Type I faults are simple faults which can be detect and removed instantly as soon as they are observed. Hence Type I faults are modeled as one stage process:

\[
\frac{dm_i(t)}{dt} = b_i \rho_i (a_i(t) - m_i(t))
\]

Where

\[
a(t) = a_i + \alpha m_i(t)
\]

The one stage process as modeled in Eq. (1) describes the failure observation, fault isolation and fault removal processes. Solving the differential equation (1) under the boundary condition \( m_i(t) (t = 0) = 0 \). We get

\[
m_i(t) = a_i^* \left[1 - e^{-b_i \rho_i t} \right] = a_i^* F_i(t)
\]
For Type II faults, it is assumed that the testing team will have to spend more time to analyze the cause of the failure and therefore requires greater efforts to remove them when compared with Type-I faults. Hence, we consider Yamada model as hard fault which incorporated with imperfect debugging (removal process for each fault is modeled as a two-stage process).

\[ m_{2}(t) = a_{1,2}^* \left[ 1 - e^{-\lambda (1 + b_2 t) e^{-b_2 t} (1 - \alpha_2)} \right], \]  
\[ = a_{1,2}^* F_{1,2}(t) \]

Where
\[ a_{1,2}^* = (1 - \lambda) a_1 / (1 - \alpha_2) \]

Then total fault removed up to time \( t \) for first release are given as:
\[ m(t) = m_1(t) + m_2(t), \]
by relation (2) and (3) we get:
\[ m(t) = a_{1,1}^* \left[ 1 - e^{-\lambda (1 - \alpha_1) t} \right] + a_{1,2}^* \left[ 1 - e^{-\lambda (1 + b_2 t) e^{-b_2 t} (1 - \alpha_2)} \right] \]
\[ = a_{1,1}^* F_{1,1}(t) + a_{1,2}^* F_{1,2}(t) \]

As the above assumption for classifying, we use GO Model for simple fault \( F_{1,1}(t) \), and Yamada Model for hard fault, \( F_{1,2}(t) \) which incorporated with imperfect debugging.

Note that the first index in \( F_{i,j} \) shows the release \( i \) and second index show fault severity \( (j) \).

3.1. MODELING FOR RELEASE 1

In today’s software market, up-gradation or update section become important part in the well-known company and based of these part companies are trying to survive in market by adding new feature or new functionality. Foundation and Structure of software is represented in market in the first release of the software product. Hence company will have to pay more attention on it. In other hand based on performance and bug, Testing team at this release have to detect and remove fault as much as possible and minimize the possibility of occurrence of errors in future. In this model testing team not only doesn’t cease testing process after release but also must check and analyze the feedback of error in current release in the operational phase. This model classified fault into two types as simple and hard fault. Some of removed fault are simple i.e. \( a_{1,1}^* F_{1,1}(t) \) and some another faults are hard \( a_{1,2}^* F_{1,2}(t) \). It may be noted that we can’t remove all simple and hard faults and some of these fault remain in the code even after release software at time \( t = t_1 \). The mathematical equation for the numbers of faults removed is given as:
\[ m_1(t) = a_{1,1}^* F_{1,1}(t) + a_{1,2}^* F_{1,2}(t), \quad 0 \leq t < t_1 \]

Where
\[ F_{1,1}(t) = \left[ 1 - e^{-\lambda (1 - \alpha_1) t} \right], \]
\[ F_{1,2}(t) = \left[ 1 - e^{-\lambda (1 + b_2 t) e^{-b_2 t} (1 - \alpha_2)} \right] \]

And
\[ a_{1,1}^* = \lambda a_1 / (1 - \alpha_1) \text{ and } a_{1,2}^* = (1 - \lambda) a_1 / (1 - \alpha_2); \]

3.2. MODELING FOR RELEASE 2

After first release, the company has information about the reported bugs from the users, hence in order to attract more customers, a company adds some new functionality to the existing software system. Adding some new functionality to the software leads to change in the code. These new specifications in the code lead to increase in fault content. Now the testing team starts testing the upgraded system. At this stage model make difference between the location of the fault being removed i.e. whether the fault belongs to previous subroutines (release) or it is related to new functionality added to the software. Also model makes difference based on severity of faults as simple and hard fault. May be simple (hard) fault from previous release are removed at new release. This model assumed that simple fault interacts with new portion of detected faults as simple and hard fault interacts with new portion of hard detected faults. In this period when there are two versions of the software, \( a_{1,1}^* \left( 1 - F_{1,1}(t_1) \right) \) is the leftover simple fault content and also left over hard fault content is \( a_{1,2}^* \left( 1 - F_{1,2}(t_1) \right) \) for the first release, which interacts with new portion of detected faults i.e. \( F_{2,1}(t - t_1), F_{2,2}(t - t_1) \), respectively. In addition a fraction of faults generated due to enhancement of the features are removed with new rate. The mathematical equation of these finite numbers of faults removed can be given by:
\[ m_2(t) = [a_{2,1}^* + a_{1,1}^* \left( 1 - F_{1,1}(t_1) \right)] F_{2,1}(t - t_1) + [a_{2,2}^* + a_{1,2}^* \left( 1 - F_{1,2}(t_1) \right)] F_{2,2}(t - t_1); \quad t_1 \leq t < t_2 \]
Imperfect Debugging Software Reliability Growth Model for Multiple Releases

Where \( F_{2,i}(t-t_i), F_{2,2}(t-t_i) \) are imperfect GO and Yamada model and
\[
a_{2,1}^* = \lambda.a_2/(1-\alpha_i) \quad \text{and} \quad a_{2,2}^* = (1-\lambda).a_2/(1-\alpha_i); \]

3.3 MODELING FOR RELEASE 3
Similarly for release 3, we consider faults generated in third release and remaining number of faults from the second release. Simple fault from second release can remove as simple or hard fault in third release and hard fault only interacts with new portion of hard detected faults. Mathematical equation can be represented as follows:
\[
m_3(t) = [\lambda_3.a_{3,1}^* + \gamma_{2,1}^*\lambda_2.(1-F_{2,1}(t_2-t_1)].F_{3,1}(t-t_2) +
[(1-\lambda_3).a_{3,2}^* + \gamma_{2,2}^*\gamma_{2,2}^*(1-\lambda_2).(1-F_{2,2}(t_2-t_1)].F_{3,2}(t-t_2)
\]
\[t_2 \leq t < t_3\]
Where \( F_{3,1}(t-t_2) \) and \( F_{3,2}(t-t_2) \) is new portion of detected fault as simple and hard for third release given by:
\[
F_{2,1}(t) = \left[1-e^{-b_{1,i}t} \right],
F_{2,2}(t) = \left[1 - ((1+b_{2,i})e^{-b_{2,i}t}) \right] \]
\[(10)\]
where
\[
a_{3,1}^* = \lambda.a_3/(1-\alpha_i) \quad \text{and} \quad a_{3,2}^* = (1-\lambda).a_1/(1-\alpha_i); \]

3.4 MODELING MULTI UP-GRADATION FOR RELEASE 4
And similarly for release 4, the corresponding mathematical expression can be given by:
\[
m_4(t) = [a_{4,1}^* + a_{3,1}^*.(1-F_{3,1}(t_3-t_2)].F_{4,1}(t-t_3) +
[a_{4,2}^* + a_{3,2}^*.(1-F_{3,2}(t_3-t_2)].F_{4,2}(t-t_3) \quad ; t_3 \leq t < t_4\]
Where \( F_{4,1}(t-t_3) \) and \( F_{4,2}(t-t_3) \) can be defined as done in previous steps, where
\[
a_{4,1}^* = \lambda.a_4/(1-\alpha_i) \quad \text{and} \quad a_{4,2}^* = (1-\lambda).a_4/(1-\alpha_i); \]

4. ESTIMATION OF PARAMETER
In our study we used Method of Least Squares which is one of the most popular methods for estimation of parameters. Also for calculation of parameter estimation we have applied statistical package for social science “SPSS” software. In this model except release 1 we must calculate the leftover simple and hard faults for the previous release. Once the value of leftover fault is known, the parameter of multi up-gradation for software can be estimated. The testing time data are given in four columns which is divided based on time of the release.

4.1. MODEL VALIDATION

To check the validity of the proposed model and to describe the software reliability growth, it has been tested on Tandem data set which has been released in the market with new features four times.

4.2. DATA SET AND DATA ANALYSIS
To check the validity of the proposed model and to describe the software reliability growth, it has been tested on tandem computer [4] 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. Figure (3-6) shows the estimated and the actual values of the number of faults removed for four releases. Based on data available given in Table 2, the performance analysis of proposed model is measured by the four common criteria that we define as below:

4.3. CRITERIA FOR COMPARISONS
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 \( \hat{m}(t_i) \) represent the estimated number of faults by time \( t_i \). in all mentioned criteria the lower value indicate less fitting error.

1. The Bias is defined as:

\[
\text{Bias} = \frac{\sum_{i=1}^{k} (\hat{m}(t_i) - m_i)}{k}
\]

The difference between the observation and prediction of number of failures at any instant of time \( i \) is known as PE, (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:

\[
\text{Variation} = \sqrt{ \left( \frac{1}{N-1} \right) \sum (PE_i - Bias)^2}
\]

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:

\[
\text{RMSPE} = \sqrt{\text{Variation}^2 + \text{Bias}^2}
\]

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:

$$MSE = \frac{\sum_{i=1}^{k} (\hat{m}(t_i) - y_i)^2}{k}$$

Where $k$ is the number of observations. The lower MSE indicates less fitting error, thus better goodness of fit.

### 4.4. RESULTS OF ESTIMATION

In this section we analyze the parameter-estimation of each release separately. At first, we discuss about nature of fault on the bases of fault severity. In first release we get high value of $\lambda$ ($\lambda = 0.7663$). It means that more portion of fault detected at this release are simple fault. This value completely matched with the nature of exponential behavior of simple fault at first release. In second release this value decreases and in third release we get $\lambda = 0.0195$ (which can interpreted as more proportion of hard faults in release 2 and 3). Also this value is completely reasonable because of S-shapedness of the original data. It might be seen from actual data, release 3 is more S-shaped followed by release 2, further followed by S-Shapedness of release 4 but with lower rate and this behavior matched with behavior of parameter $\lambda$ in each release. In first release we estimate $a_1 = 100$ from 100 fault content in first release and model good estimated this parameter. In second release $a_1 = 118$ from 120 fault removed, parameter $a_2$ at this release is not under estimate, because we are applying the concept of imperfect debugging on the model and parameter $a_2$ related to new fault added to software by new functionality, and also we removed 5 old fault from previous release means that overall fault removed is 123. In release 2 we find imperfect debugging with rate $\rho_2 = 0.5535$ and $\alpha_2 = 0.2264$ for other three releases we find that imperfect debugging is less.
CONCLUSION
The software reliability multiple releases model in this paper is based on concept of imperfect debugging and classifies fault into two categories as simple and hard fault. Here we assumed that the overall fault removal of the new release in each category depends on the faults generated in that release and on the leftover faults of just previous release. ( for each release. The value of parameters is reasonably good and estimated using original data and the goodness of fit values (variance, RMSPE, MSE and Bias) mentioned in Table 1 have decreasing behavior. Therefore, the proposed model fits the data reasonably well.

FUTURE SCOPE
In future we propose to use different distribution functions for modeling different releases of the software incorporating three kind of fault as simple and hard and complex. Also to analyze the behavior and to monitor the change in reliability we wish to incorporate the concept of Laplace trend test on new model It will also help in getting more information about timing for next release.

REFERENCES
[14]. K. Kapur, A.S. garmabaki, J. Singh, software reliability multi upgradation model with imperfect debugging, international congress on productivity, quality, reliability, optimization and modeling 08-feb-2011
[15]. P.K. Kapur, D.N. Goswami, A. Bardhan c, Ompal Singh Flexible software reliability growth model with testing effort dependent learning process.


