Chapter 2

Multi Up-Gradation Software Reliability Model with Fault Severity and Imperfect Debugging

Since computers are being used increasingly to monitor and control both safety critical and civilian systems, there is a great demand for high-quality software products. Reliability is a primary concern for both software developers and software users.

With dynamic markets and evolving business models, organizations need to stay agile to maintain and improve their competitive edge. Software companies 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 previous 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. Software reliability assessment is increasingly important in developing and testing new software products. Before newly developed software is released to the user, it is extensively tested for errors that may have been introduced during development.

This Chapter is based on the following research papers entitled:


Many software reliability growth models (SRGMs) have been proposed over past three decades that estimate the reliability of a software system as it undergoes changes through the removal of failure causing fault (Pham, 2006, Kapur and Garg, 1992, Kapur et al., 1995, Yamada et al., 1984, Singpurwalla and S.P Wilson, 1999, Xie, 1991, Zhang, 2006). But unfortunately most of the models didn’t consider anything about the increase in failure rate once an up-gradation is made on the software. The innovation-related factors like development process, software testing and debugging process and team structure have significant impact on a firm’s future growth potential. In the last few decades, it has been observed that the software development process (i.e. new product development, technology alliance etc.) has evolved rapidly due to the intensified market competition, shrinking budget, expanding system requirements and accelerating rate of software enhancement.

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 (Kapur et al., 2011b, Pham and Zhang, 2003). 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(K.Khataneh and Mustafa, 2009). Similar gaffes have occurred from important government systems to freeware on the internet (Kapur et al., 2011i, Kapur et al., 2008, Williams, 2005. Sometimes Upgrades can worsen a product and user may prefer an older version.
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 SRGM can be depicted by the Figure 1.2 given in chapter 1. The traditional SRGMs 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 1.2 (given in chapter 1) 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.

Recently Kapur et al. (2011i, 2011k), Kapur et al. (2010d) developed a multi up-gradation reliability model, considering that cumulative faults in each release depend on all previous releases and also assumes that fault is removed with certainty. But the proposed model is based on the assumption that the overall fault removal of the new release depends on the reported faults from the just previous release of the software and on the faults generated due to adding some new functionalities (add-ons/upgradations) to the existing software system. Therefore, it’s not necessary to 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).

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. While the first phenomenon is known as imperfect fault removal, the second is called error generation (Pham, 2006, Shyur, 2003). 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” (Singh et al., 2008, Kapur et al., 1999).

This chapter is divided into three sections. In Section 2.1, a multi up-gradation software reliability model is developed under imperfect debugging environment for faults of a different severity. Section 2.2 is discusses about change of nature of fault during successive release of software. In the proposed model, we consider that undetected simple faults of old code are removed as simple fault in new release or it is also may happen that simple faults of old code are removed as hard fault during new release. But the hard faults of old code are assumed not to change its nature during testing of next release and it is removed as hard only. Section 2.3 develops a unified framework using hazard rate function to develop various multi up-gradation model using different failure distributions under imperfect debugging phenomenon. Parameter estimation, model validation have been done for all the models discussed in the chapter using real data sets cited in literature. The models obtained are accurate and fit the data reasonably well.

2.1 Modeling Multi Up-gradation Software Reliability with Imperfect Debugging

Since the human factor is involved in debugging the software faults, we cannot deny the possibility of imperfection in testing and debugging. In practice, the testing efficiency is usually imperfect. Sometimes the debugging team may not be able to remove the fault perfectly on the detection of failure and the original fault may remain (known as imperfect fault removal) or replaced by another fault (known as fault generation). In this section, we have proposed a model considering the effect of fault severity in imperfect debugging environment on software reliability and removal process for multiple releases of the software.
2.1.1 Assumptions:

1. The fault detection/correction process are modeled by non- homogeneous poison process (NHPP).
2. Software faults are of 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. Probability of perfect debugging and error generation may happen during the fault removal process with the rate of $\rho$ and $\alpha$ , respectively.

2.1.2. Notations

- $m(t)$ Expected number of faults removed by time $t$.
- $f(t)$ Probability density function for fault removal process.
- $F_{i,j}(t)$ Probability distribution function for removal process for fault of severity $j$ and release $i$, $i=1..4$, $j=1..2$.
- $t_i$ Time for $i^{th}$ release, $i=1..4$.
- $a(t)$ Time dependent fault content function.
- $a_i$ Initial fault content for $i^{th}$ release, $i=1..4$.
- $a_i^*$ Initial fault content for $i^{th}$ release in presence of imperfect debugging.
- $a$ Total Initial fault content in the software.
- $b(t)$ Time dependent fault detection rate function.
- $\rho_i$ Probability of removal of simple fault for $i^{th}$ release, $i=1..4$.
- $\rho_2$ Probability of removal of hard fault for $i^{th}$ release, $i=1..4$.
- $\rho$ Probability of fault removal upon it’s defects.
- $\alpha_i$ Simple fault introduction rate for $i^{th}$ release, $i=1..4$.
\( \alpha_i \)  
Hard fault introduction rate for \( i^{th} \) release, \( i = 1..4 \).

\( \beta_i \)  
Logistic learning factor for \( i^{th} \) release, \( i = 1..4 \).

\( b_j \)  
Fault detection rate for different severity, \( j = 1..2 \).

\( \beta, b, \lambda, \lambda \)  
Constant.

### 2.1.3. Software Reliability Models Based on Fault Severity and Imperfect Debugging

In the simplest model, the function \( a(t) \) is always assumed to be constant. A constant \( a(t) \) stands for the assumption that no new errors are introduced during the debugging process. In a general model, the functions \( a(t) \) is function of time and, for practical purposes, is increasing with time. An increasing \( a(t) \) shows that the total number of errors (including those already detected) increases with time because new faults are introduced during the debugging process. It is further assumed that only two types of faults exist in software as Type I, Type II (simple and hard).

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

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

(2.1)

It is assumed that Type I faults are simple faults which can be detected and removed instantly as soon as they are observed. Hence Type I faults are modeled as one stage process(Goel and Okumoto, 1979):

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

(2.2)

Where \( a(t) = a_i + \alpha_i m_i(t) \)  
(2.3)

The one stage process as modeled in Eq.(2.2) describes the failure observation, fault isolation and fault removal processes with no time lag.
Chapter 2  
Multi Up-Gradation Software Reliability Model With Fault Severity and Imperfect Debugging

Solving the differential equation (2.2) under the boundary condition \( m_1(t = 0) = 0 \). We get

\[
m_1(t) = \frac{a_1}{(1 - \alpha_1)} \left[1 - e^{-\beta_1 t^{(1-\alpha_1)}}\right] = a_{1,1} F_{1,1}(t)
\]  
(2.4)

Where

\[
a_{1,2} = \frac{a_1}{(1 - \alpha_2)}
\]  
(2.5)

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 Yamada et al. (1983) model is considered as hard fault which incorporated with imperfect debugging which discussed in section (1.9.1.2) (removal process for such faults is modeled as a two-stage process).

\[
m_2(t) = \frac{a_1}{(1 - \alpha_2)} \left[1 - ((1 + b_2 t)e^{-b_2 t})^{\frac{1}{\beta_2}}\right]^{\frac{1}{\beta_2}} = a_{1,2} F_{1,2}(t)
\]  
(2.6)

Here \( a_{1,2} = \frac{a_1}{(1 - \alpha_2)} \)

Then total number of fault removed up to time \( t \) for first release are given as:

\[
m(t) = m_1(t) + m_2(t)
\]

by relation (2.4) and (2.6) we get:

\[
m(t) = \frac{\lambda a_1}{(1 - \alpha_1)} \left[1 - e^{-\beta_1 t^{(1-\alpha_1)}}\right] + \frac{(1 - \lambda) a_1}{(1 - \alpha_2)} \left[1 - ((1 + b_2 t)e^{-b_2 t})^{\frac{1}{\beta_2}}\right]^{\frac{1}{\beta_2}} = a_{1,1}^* F_{1,1}(t) + a_{1,2}^* F_{1,2}(t)
\]  
(2.7)

Where \( a_{1,1}^* = \lambda a_1 / (1 - \alpha_1) \) and \( a_{1,2}^* = (1 - \lambda) a_1 / (1 - \alpha_2) \).
Here, we use GO Model for simple fault $F_{i,1}(t)$, and Yamada Model for hard fault, $F_{i,2}(t)$ under imperfect debugging phenomenon.

2.1.4. Modeling Fault Removal Process for Multiple Software Releases

2.1.4.1. Modeling for Release 1

This model is based on the assumption that software consist of faults classified into two types as simple and hard fault under imperfect debugging. Let $a^*_{i,1} \cdot F_{i,1}(t)$ be number of simple faults and $a^*_{i,2} \cdot F_{i,2}(t)$ be the number of hard faults are removed from software system. 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 faults removal process is given as:

$$m_i(t) = a^*_{i,1} \cdot F_{i,1}(t) + a^*_{i,2} \cdot F_{i,2}(t), \quad 0 \leq t \leq t_1$$

(2.8)

Where

$$F_{i,1}(t) = \left[ 1 - e^{-b_1 \cdot p_i \cdot (1-\alpha_1) \cdot t} \right],$$
$$F_{i,2}(t) = \left[ 1 - ((1 + b_2 \cdot t) \cdot e^{-b_2 \cdot t})^{b_2 \cdot (1-\alpha_2)} \right]$$

(2.9)

$$a^*_{i,1} = \lambda \cdot a^*_i / (1 - \alpha_1) \text{ and } a^*_{i,2} = (1 - \lambda) \cdot a^*_i / (1 - \alpha_2);$$

2.1.4.2. 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 the 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. In addition, the proposed model differentiates between simple and hard faults on the base of their severity. This model assumed that
simple fault interacts with new portion of detected faults as simple and also hard fault interacts with new portion of hard detected faults (fault nature is remained same in fault detection/correction in new release). 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, fraction of faults generated due to enhancement of the features are removed with new rate. The mathematical equation for the numbers of faults removed may be given by:

\[
m_2(t) = [a_{2,1}^* + a_{1,1}^* \left( 1 - F_{1,1} (t_1) \right)] F_{2,1} (t - t_1) + [a_{2,2}^* + a_{1,2}^* \left( 1 - F_{1,2} (t_1) \right)] F_{2,2} (t - t_1) \quad t_1 < t \leq t_2
\]

(2.10)

Where \( F_{2,1} (t - t_1), F_{2,2} (t - t_1) \) correspond to GO and Yamada model under imperfect debugging and

\[
a_{2,1}^* = \lambda a_2^* / (1 - \alpha_1) \quad \text{and} \quad a_{2,2}^* = (1 - \lambda) a_2^* / (1 - \alpha_2)
\]

(2.11)

2.1.4.3. Modeling for Release 3

Similarly for release 3, we consider faults generated in third release and remaining number of faults from the second release. Fault from second release may be removed as simple or hard fault during testing of third release and simple (hard) fault from second release may interacts only with simple (hard) fault during fault removal process. The mathematical equation can be represented as follows:

\[
m_3(t) = [a_{3,1}^* + a_{2,1}^* \left( 1 - F_{2,1} (t_2 - t_1) \right)] F_{3,1} (t - t_2) + [a_{3,2}^* + a_{2,2}^* \left( 1 - F_{2,2} (t_2 - t_1) \right)] F_{3,2} (t - t_2) \quad t_2 < t \leq t_3
\]

(2.12)

Where \( F_{3,1} (t - t_2) \) and \( F_{3,2} (t - t_2) \) correspond to simple and hard fault for third release and they are given by:
2.1.4.4. Modeling for Release 4

As explained earlier, the corresponding mathematical expression for release 4 may be given by:

\[ m_4(t) = [a_{4,1}^* + a_{4,1}'(1 - F_{3,1}(t_3 - t_2))].F_{4,1}(t - t_3) + \\
[ a_{4,2}^* + a_{4,2}'(1 - F_{3,2}(t_3 - t_2))].F_{4,2}(t - t_3) \; ; \; t_3 < t \leq t_4 \]

(2.14)

Where \( F_{4,1}(t - t_3) \) and \( F_{4,2}(t - t_3) \) can be defined as done in previous steps, where

\[ a_{4,1}^* = \lambda.a_4^*/(1 - \alpha_1) \; \text{and} \; a_{4,2}^* = (1 - \lambda).a_4^*/(1 - \alpha_2) ; \]

2.1.5. Data Set and Model Validation

To check the validity of the proposed model and to describe the software reliability growth, it has been tested on tandem computer (Wood, 1996, Pham, 2006, Kapur et al., 2011b) four release data set. We have used non linear least square technique in SPSS software for estimation of parameters. Goodness of fit criteria namely Mean squared error (MSE); Coefficient of multiple determinations \( R^2 \); Bias, Root Mean Square Prediction Error (RMSPE), Variation are used to compare and analyze the fitness of the model.

2.1.6. Parameter Estimation and Goodness of Fit

Estimated value for parameters of each releases are given in Table 2.1(a). Table 2.1(b) shows the goodness of fit criteria of the four software releases. Figure (2.1) to (2.4) shows the estimated and the actual values of the number of faults removed for four releases.
Table 2.1(a): Parameter estimates

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Releases</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>100</td>
<td>118</td>
<td>64</td>
<td>45</td>
</tr>
<tr>
<td>$b_1$</td>
<td>0.4038</td>
<td>0.1881</td>
<td>0.01212</td>
<td>0.008768</td>
</tr>
<tr>
<td>$b_2$</td>
<td>0.2181</td>
<td>0.2943</td>
<td>0.3342</td>
<td>0.21863</td>
</tr>
<tr>
<td>$p_1$</td>
<td>0.2987</td>
<td>0.8379</td>
<td>0.51587</td>
<td>0.71812</td>
</tr>
<tr>
<td>$p_2$</td>
<td>0.9515</td>
<td>0.5535</td>
<td>0.78183</td>
<td>0.94077</td>
</tr>
<tr>
<td>$\alpha_1$</td>
<td>0.1475</td>
<td>0.2264</td>
<td>0.04295</td>
<td>0.001</td>
</tr>
<tr>
<td>$\alpha_2$</td>
<td>0.2351</td>
<td>0.01</td>
<td>0.001</td>
<td>0.06317</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0.7663</td>
<td>0.1849</td>
<td>0.0195</td>
<td>0.22093</td>
</tr>
</tbody>
</table>

Table 2.1(b): Goodness of fit criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Releases</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0.984</td>
<td>0.992</td>
<td>0.984</td>
<td>0.995</td>
</tr>
<tr>
<td>Bias</td>
<td>0.2293</td>
<td>0.09653</td>
<td>-0.1198</td>
<td>0.0601</td>
</tr>
<tr>
<td>MSE</td>
<td>14.7372</td>
<td>11.1621</td>
<td>6.9564</td>
<td>0.8847</td>
</tr>
<tr>
<td>RMSPE</td>
<td>3.9382</td>
<td>3.4324</td>
<td>2.7545</td>
<td>0.9663</td>
</tr>
<tr>
<td>Variation</td>
<td>3.9316</td>
<td>3.431</td>
<td>2.7519</td>
<td>0.96443</td>
</tr>
</tbody>
</table>

Also figure (2.1) to (2.4) shows the estimated and the actual values of the number of faults removed for four releases of software.
**Figure 2.1:** Goodness of fit of release 1.

**Figure 2.2:** Goodness of fit of release 2.
2.1.7. Data Analysis

At first, we discuss about nature of fault (fault severity). For first release, we have high value of $\lambda$ ($\lambda = 0.7663$). It implies that more proportion of fault detected at this release are simple faults. This value completely matches 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 may be interpreted as more proportion of hard faults in release 2 and 3). Also this value is completely reasonable because of S-shapedness of original data. It might be seen from actual data, release 3 is more S-shaped followed by release 2, further follows by S-shapedness of release 4 but with lower rate and this behavior is well exhibited by the estimated values of parameter $\lambda$ in each release. In first release, we estimate $a_1 = 100$ For second release the estimated value of $a_2$ is 118 while actual number of fault removed are 120, It may be noted that parameter $a_2$ at this release is not an under estimate, because we are applying the concept of imperfect debugging on the model. In release 2 we find imperfect debugging with probability $\rho_2 = 0.5535$ and error generation with $\alpha_2 = 0.2264$. For other three releases we find that imperfect debugging is less.

It can also been seen from Table 2.1(b) that the value of Adj-$R^2$ is highest for the fourth release and also the value of MSE, Variation and RMSPE are lowest. This concludes that the model fits the fourth release failure data set best.

2.2. Multi Up-Gradation SRGM with Varying Nature of Faults

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. Faults can be categorized with respect to time which they take for isolation and removal after their observation. Faults are classified as “simple fault” when time between their observation and removal is negligible and “hard fault” if more efforts and time is required for the removal process(Kapur et al., 1999).

In previous section, we developed multi up-gradation SRGM under imperfect debugging phenomenon for fault of different severity. Here in this section, we relax the imperfect debugging assumption and discuss about change of nature of fault within the releases which have effect on fault detection/correction process.
2.2.1. Assumptions:

The basic assumptions of the model are as follows:

1. The fault detection/correction processes are modeled by Non-Homogeneous Poison Process (NHPP).
2. The number of faults detected at any time is proportional to the remaining number of faults in the software.
3. The faults existing in the software are of two types and each type of fault is modeled by a different growth curve.
4. The number of faults in the beginning of the testing phase is finite.
5. All faults are mutually independent from failure detection point of view.
6. During the FRP, no new fault is introduced into the software, i.e. debugging process is perfect.

2.2.2. Notations

\[ m(t) \quad : \quad \text{Expected number of faults removed by time } t. \]
\[ F_{i,j}(t) \quad : \quad \text{Probability distribution function for removal process for fault of severity } j \text{ and release } i, \quad i = 1..4, \quad j = 1..2. \]
\[ t_i \quad : \quad \text{Time for } i^{th} \text{ release } i = 1..4. \]
\[ a_i \quad : \quad \text{Initial fault content for } i^{th} \text{ release } i = 1..4. \]
\[ a \quad : \quad \text{Initial fault content in the software for single release of software.} \]
\[ b(t) \quad : \quad \text{Time dependent fault detection rate function.} \]
\[ \beta_i \quad : \quad \text{Logistic learning factor for } i^{th} \text{ release } i = 1..4. \]
\[ b_j \quad : \quad \text{Fault detection rate for Type } i \text{ in each release } j = 1..2. \]
\[ \beta, b, \lambda \quad : \quad \text{Constant.} \]
\[ \lambda_i \quad : \quad \text{Proportion of simple fault in the release } i. \]
\[ (1 - \lambda_i) \quad : \quad \text{Proportion of hard fault in the release } i. \]
2.2.3. Change of Nature of Fault in Multi Release

It is quite possible that faults change its nature in successive release mainly because of two reasons:

1. Complexity due to feature intensification
2. Change in design and testing team.

Whenever some add-ons are done in the software i.e. we add some extra features into it, the chances of code becoming complex increases. Therefore, the embodying of certain features might result in changing the behavior of the faults lying in the system. Furthermore, there might be the chances that the project manager changes programmer team for newer release, therefore, due to lack of fully fledged knowledge about the development of the earlier code the new team might not be accustomed with the older code and in such a situation might face some difficulty to detect or correct the bugs reported from previous release.

In this model we consider that simple faults of old code is removed as simple fault in new release or it is also may happen that simple faults of old code is removed as hard fault during new release. But the hard faults of old code is assumed not to change its nature during testing of next release and they are removed as hard fault only.

![Figure 2.5: Pattern of change of nature of fault in multi release](image-url)
2.2.4. Software Reliability Models Based on Fault Severity

In literature, a growth curve has been proposed to represent the removal process of different type of faults. Here in section, we assumed that removal of simple faults in nature follows exponential curve. For other faults, which are more severe in nature, logistic learning has been incorporated during removal phenomenon and these faults are depicted by different types of S-shaped curves. It is assumed that only two types of faults exist in software as Type I, Type II (simple and hard).

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

\[
\frac{dm_1(t)}{dt} = b_1(a_1 - m_1(t))
\]  
(2.15)

Eq. (2.15) describes the failure observation, fault isolation and fault removal processes as one stage process.

Solving the differential equation (2.15) under the boundary condition \(m_1(t = 0) = 0\),

We get:

\[
m_1(t) = a_1 \left[1 - e^{-b_1 t}\right] = a_1 F_{t,1}(t)
\]  
(2.16)

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 the removal process for such faults is modeled as a two-stage process:

\[
\frac{dm_{2,f}(t)}{dt} = b_2(a_2 - m_{2,f}(t))
\]  
(2.17)

\[
\frac{dm_2(t)}{dt} = b_2(t)(m_{2,f}(t) - m_2(t))
\]  
(2.18)

Where \(b_2(t) = \frac{b_2}{1 + \beta e^{-b_2 t}}\)  
(2.19)
The first stage of the two-stage process is given by Eq. (2.17). This stage describes the failure observation process. The second stage of the two-stage process given by Eq. (2.18) describes the delayed fault removal process. During this stage the fault removal rate is assumed to be time dependent. The reason for this assumption is to incorporate the effect of learning on the removal personnel. With each fault removal insight is gained into the nature of faults present and function described in Eq. (2.19) is called logistic function. It may be noted that \( b_2(t) \) increase monotonically with time \( t \) and tend to constant \( b_2 \) as \( t \to \infty \).

Solving, the above differential equations under the boundary condition, \( m_2(t=0) = 0 \) and \( m_2(t=0) = 0 \), we get

\[
m_2(t) = a_2 \cdot \frac{1 - ((1+b_2t)e^{-b_2t})}{1 + \beta_2 e^{-b_2t}} = a_2 \cdot F_{1,2}(t) \tag{2.20}
\]

Also assuming that \( a = a_1 + a_2 \) and \( a_1 = \lambda a \), \( a_2 = (1-\lambda) a \).

Then total fault removed up to time \( t \) for first release are given as:

\[
m(t) = m_1(t) + m_2(t) \text{, by relation (2.16) and (2.20), we get:}
\]

\[
m(t) = a_1 \cdot \left[1 - e^{-b_1t}\right] + a_2 \cdot \frac{1 - ((1+b_2t)e^{-b_2t})}{1 + \beta_2 e^{-b_2t}}
\]

\[
= \lambda a \cdot F_{1,1}(t) + (1 - \lambda) a \cdot F_{1,2}(t) \tag{2.21}
\]

2.2.5. Multi Up-Gradation Model Development

2.2.5.1. Modeling for Release 1

Let the First Release of software be done at \( t = t_1 \). Note that this model classifies fault into two types as simple and hard fault, some of removed fault are simple i.e. \( \lambda_1 a \cdot F_{1,1}(t) \) and some another faults are hard \( (1-\lambda_1) a \cdot F_{2,1}(t) \). Also, we can’t
remove all faults as simple or hard and some faults remain in the code when software is released. The mathematical equation of number of faults removed is given by:

\[ m_i(t) = \lambda_i a_i F_{1,1}(t) + (1 - \lambda_i) a_i F_{1,2}(t), \quad 0 \leq t \leq t_1 \]  

Where

\[ F_{1,1}(t) = \left[ 1 - e^{-b_i (t-t_i)} \right] \]

\[ F_{1,2}(t) = \left[ \frac{1-(1+b_2(t-t_i))e^{-b_2(t-t_i)}}{1 + \beta_2 e^{-b_2(t-t_i)}} \right] \]  

2.2.5.2. Modeling for Release 2

During testing of second release, company has information about bugs reported from the users of first release in operational phase. We assume that these bugs may be categorized as simple or hard faults. During testing, it is further possible that simple faults of old code are removed as simple fault in new release or it may also happen that simple faults of old code are removed as hard fault during new release. But the hard faults of old code are assumed not to change their nature during testing of release 2 and they are removed as hard faults only. In other words, when first up-gradation is made, \( \gamma_1 \lambda_i a_i \left(1 - F_{1,1}(t_1) \right) \) represent the leftover simple fault content of the first release which interacts with new portion of simple detected faults i.e. \( F_{2,1}(t-t_i) \) and \( (1 - \gamma_1) \lambda_i a_i \left(1 - F_{1,1}(t_1) \right) \) is the leftover simple faults of the first version which are detected as hard faults and removed with CDF of \( F_{2,2}(t-t_i) \). Here \( (1 - \lambda_i) a_i \left(1 - F_{2,1}(t_1) \right) \) represent leftover hard fault content of the first release which interacts with new portion of hard detected faults i.e. \( F_{2,2}(t-t_i) \). In addition a fraction of simple and hard faults generated due to enhancement of the features are removed with new rate are \( \lambda_2 a_2 F_{2,1}(t-t_i) \) and \( (1 - \lambda_i) a_i F_{2,2}(t-t_i) \), respectively.

It may be noted that some of simple & hard faults will remain in the code when the software is released at time \( t = t_2 \).
\[ m_2(t) = [\lambda_2 a_2 + \gamma_2 \lambda_4 a_4 (1 - F_{1,1}(t_1))].F_{2,1}(t - t_1) + [(1 - \lambda_2).a_2 + (1 - \gamma_2) \lambda_4 a_4 (1 - F_{1,1}(t_1))] \\
+ (1 - \lambda_4) a_4 (1 - F_{1,2}(t_1))].F_{2,2}(t - t_1), \quad t_1 < t \leq t_2 \]  \hspace{1cm} (2.24)

Where

\[ F_{2,1}(t - t_1) = \left[ 1 - e^{-b_1(t-t_1)} \right] \quad \text{and} \quad F_{2,2}(t - t_1) = \left[ \frac{1 - ((1 + b_2(t-t_1))e^{-b_2(t-t_1)})}{1 + \beta e^{-b_2(t-t_1)}} \right] \]

2.2.5.3. Modeling for Release 3 and 4

Similarly, for release 3, we consider faults generated in third release and remaining faults from the second release. Simple faults from second release may be removed as simple or hard fault in third release and hard fault only interacts with new portion of hard detected fault. Mathematical equation may be represented as follows:

\[ m_3(t) = [\lambda_3 a_3 + \gamma_3 \lambda_5 a_5 (1 - F_{2,1}(t_2 - t_1))].F_{3,1}(t - t_2) + [(1 - \lambda_3).a_3 + (1 - \gamma_3) \lambda_5 a_5 (1 - F_{2,1}(t_2 - t_1))] \\
+ (1 - \lambda_5) a_5 (1 - F_{2,2}(t_2 - t_1))].F_{3,2}(t - t_2), \quad t_2 < t \leq t_3 \]  \hspace{1cm} (2.25)

Similarly for release 4, the corresponding mathematical expression may be given by:

\[ m_4(t) = [\lambda_4 a_4 + \gamma_4 \lambda_6 a_6 (1 - F_{3,1}(t_3 - t_2))].F_{4,1}(t - t_3) + [(1 - \lambda_4).a_4 + (1 - \gamma_4) \lambda_6 a_6 (1 - F_{3,1}(t_3 - t_2))] \\
+ (1 - \lambda_6) a_6 (1 - F_{3,2}(t_3 - t_2))].F_{4,2}(t - t_3), \quad t_3 < t \leq t_4 \]  \hspace{1cm} (2.26)

Where \( F_{3,1}(t - t_2), F_{3,2}(t - t_2), F_{4,3}(t - t_3) \) and \( F_{4,2}(t - t_3) \) are defined as done in previous subsections.

2.2.6. Data Set and Model Validation

To check the validity of the proposed model and to describe the software reliability growth, it has been tested on tandem computer (Kapur et al., 2011b, Pham, 2006, Wood, 1996) as described in section 2.1.5. goodness of fit criteria namely, Mean squared error (MSE); Coefficient of multiple determinations \( R^2 \); Bias, Root Mean
Square Prediction Error (RMSPE), Variation are used to compare and analyze the fitness of the model.

### 2.2.7. Parameter Estimation and Goodness of Fit

Estimated value of parameters of each releases are given in Table 2.2(a). Furthermore, Table 2.2(b) shows the goodness of fit of the four software releases.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Releases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$a$</td>
<td>109</td>
</tr>
<tr>
<td>$b_1$</td>
<td>0.26871</td>
</tr>
<tr>
<td>$b_2$</td>
<td>0.23154</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0.34924</td>
</tr>
<tr>
<td>$\beta$</td>
<td>2.17826</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>-----</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Releases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0.992</td>
</tr>
<tr>
<td>Bias</td>
<td>0.7941</td>
</tr>
<tr>
<td>MSE</td>
<td>6.8788</td>
</tr>
<tr>
<td>Variation</td>
<td>2.5645</td>
</tr>
<tr>
<td>RMSPE</td>
<td>2.6847</td>
</tr>
</tbody>
</table>
Figures (2.6) to (2.9) show the estimated and the actual values of the number of faults removed for four releases. From the curves, it may be calculated that model fits data reasonably well.

**Figure 2.6:** Goodness of fit of release 1.

**Figure 2.7:** Goodness of fit of release 2.
Here we discuss and analyze the value of parameters for each release. As described in section 2.1.7, the S-shapedness of curve in this model is justified by the value of
parameter $\beta_i$ in Table 1. Also the value of $\lambda$ in the third release is very small because at this release we detect highest value as learning factor (which can interpreted as hard fault) which make more percentage of hard fault in this stage.

The parameter $\gamma_3$ related to the portion of left over fault in third release is very small, means that all left over undetected faults of second release are hard in nature. As pointed in the section 2.1.7, the actual number of fault removed for second release is 120 while the estimated value of $a_2$ by this model is 118. Here we wish to emphasize that parameter $a_2$ at this release is not an under-estimate. It can be explained by asserting that $a_2$ is related to the new code added to software by new functionality, and the remaining 3 faults belong to old code of previous release making overall number of fault removed 120. The proposed model give very good fit as exhibited by the values of various goodness of fit criteria.

2.3. Unified Framework of Multi Up-gradation Under Imperfect Debugging

In this section, we propose a generalized framework for deriving several multi up-gradation SRGMs under Imperfect Debugging environment. The presented framework is capable of handling any general distribution function and is thus an important step towards the unification of multi up-gradation software reliability. The unification scheme eases the task of model selection. From this approach, we can not only obtain existing multi up-gradation SRGMs which was discussed in previous section but also can develop new models. It may noted that by assuming $\rho=1$ and $\alpha = 0$ model converted to multi up-gradation in a perfect debugging environment (Kapur et al., 2011c, Kapur et al., 2011a)

2.3.1 Assumptions:

The basic assumptions of the model are as follows:

1. The fault detection/correction processes are modeled by NHPP.
2. The number of faults detected at any time is proportional to the remaining number of faults in the software.

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

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

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

6. When a software failure occurs, an instantaneous repair effort starts and both type of imperfect debugging may occurs.

2.3.2. Notations

\[m(t)\] : Expected number of faults removed by time \(t\).

\[f(t)\] : Probability density function for FRP.

\[F(t)\] : Probability distribution function for FRP.

\[\Phi(t, \mu, \sigma)\] : Normal distribution function.

\[t_{i-1}\] : Time for \(i^{th}\) release \(i = 1..n\).

\[a(t)\] : Time dependent fault content function.

\[a_i\] : Initial fault content for \(i^{th}\) release \(i = 1..n\).

\[a_i^*\] : Initial fault content for \(i^{th}\) release in presence of imperfect debugging, \(i = 1..n\).

\[a\] : Total Initial fault content in the absence of imperfect debugging.

\[b(t)\] : Time dependent fault detection rate function.

\[\rho\] : Probability of fault removal upon its defect.

\[\rho_i\] : Probability of fault removal upon its defect for \(i^{th}\) release, \(i = 1..n\).

\[\alpha_i\] : Fault introduction rate for \(i^{th}\) release, \(i = 1..n\).

\[\beta_i\] : Logistic learning factor for \(i^{th}\) release, \(i = 1..n\).

\[b_i\] : Fault detection rate for \(i^{th}\) release, \(i = 1..n\).

\[\beta, b\] : Constant.
2.3.3. Unification Modeling Based On Hazard Rate with Imperfect Debugging

FRP can be model by the help of differential equation. The differential equation which explains the rate of change of cumulative number of fault removed under the effect of imperfect debugging phenomenon by using hazard rate function is given by:

\[
\frac{dm(t)}{dt} = \frac{f(t)}{1-F(t)} \cdot \rho \cdot [a(t) - m(t)]
\]

(2.27)

Where \( a(t) = a + \alpha m(t) \) (2.28)

and \( s(t) = \frac{f(t)}{1-F(t)} \) is hazard rate function.

Solving the differential equation (2.27) under initial condition \( m(0) = 0 \), we get mean value function as:

\[
m(t) = \frac{a}{1-\alpha} \left[ 1- \left(1 - F(t) \right)^{\rho(1-\alpha)} \right] = \alpha^* F^*(t)
\]

(2.29)

2.3.4. General Framework for Multi Up-gradation Model

Our modeling framework with both type of imperfect debugging is developed based on a unified framework proposed by literatures (Kapur et al., 2011a). Mathematical model related to each release are given in subsequent section, separately.

2.3.4.1. Modeling for Release 1

Let the First Release of software be done at \( t = t_0 = 0 \). As there is no other previous release, the testing phase of release 1 may represented by classical SRGM given by:

\[
m_1(t) = \frac{a_1}{1-\alpha_1} \left[ 1 - \left(1 - F_1(t) \right)^{\rho(1-\alpha_1)} \right] = a_{1*} F_{1*}^*(t) \quad 0 \leq t \leq t_1
\]

(2.30)

Where
\[ a_1^* = \frac{a_1}{1 - \alpha_1} \text{ and } F_1^*(t) = \left[ 1 - \left(1 - F_1(t)\right)^{\rho_1(1 - \alpha_1)} \right]. \]

### 2.3.4.2. Modeling for Release 2

As described in section 2.1.4.2, when there are two versions of the software, 
\[ a_1^*,(1 - F_1^*(t_1)) \] is the leftover fault content of the first version. During testing of second release these faults interacts with new portion of detected faults, i.e. 
\[ F_2^*(t - t_1) \]. In addition a fraction of faults generated due to enhancement of the features are removed with new rate i.e. 
\[ F_2^*(t - t_1) \]. The mathematical equation for the number of faults removed may be given by:

\[ m_2(t) = (a_2^* + a_1^*(1 - F_1^*(t_1)))F_2^*(t - t_1), \quad t_1 < t \leq t_2 \]  

(2.31)

Where

\[ a_2^* = \frac{a_2}{1 - \alpha_2} \quad \text{and} \quad F_2^*(t) = \left[ 1 - \left(1 - F_2(t)\right)^{\rho_2(1 - \alpha_2)} \right]. \]

### 2.3.4.3. Modeling for \( i^{th} \) Release

Similarly for release \( i \), we consider faults generated in \( i^{th} \) release and remaining number of faults from the \((i-1)^{th}\) release and the corresponding mathematical equation can be represented as follows:

\[ m_i(t) = (a_i^* + a_{i-1}^*(1 - F_{i-1}^*(t_{i-1} - t_{i-2})))F_i^*(t - t_{i-1}), \quad t_{i-1} < t \leq t_i \]  

(2.32)

Where \[ a_i^* = \frac{a_i}{1 - \alpha_i} \quad \text{and} \quad F_i^*(t - t_{i-1}) \] corresponded to FRP of new portion of detected fault for \( i^{th} \) release which given by:

\[ F_i^*(t - t_{i-1}) = \left[ 1 - \left(1 - F_i(t - t_{i-1})\right)^{\rho_i(1 - \alpha_i)} \right] \]
2.3.5 Derivation of New and Existing Models

In this section, we will derive two MUSRGMs under two type of imperfect debugging. By using unified, we are able to drive various multi release SRGM under the effect of imperfect debugging. Initially we used GO-model (Goel and Okumoto, 1979) and Yamada model(Yamada et al., 1983) in the model structure, but These models provide poor goodness of fit on software failure data. It must have occurred due to inflexibility nature of GO and Yamada model. It may be noted that GO model and Yamada model have exponential and s-shape nature, respectively. These models could be useful when failure data set has same behavior in all releases, Later we worked on Kapur (Kapur and Garg, 1992) as model (SRGM1) and Normal distribution function (SRGM2) and the results are given in the following section.

2.3.5.1 Multi Up-Gradation Model based on K-G-Model. SRGM 1

Kapur-Garg model is one of most popular model in software reliability area and can fit on many types of failure data because of flexible nature which related to learning parameter $\beta$. For deriving new model in first release, let consider that

$$s(t) = \frac{b_1}{1 + \beta_1 e^{-b_1 t}}$$

and use $a(t)$ mentioned in Eq.(2.28). Under initial condition $m(0) = 0$, we get mean value function as:

$$m(t) = \frac{a_1}{1 - \alpha_1 \left[ 1 - \left( \frac{(1 + \beta_1 e^{-b_1 t})}{1 + \beta_1 e^{-b_1 t}} \right)^{\alpha_1 (1-\alpha_1)} \right]}$$

(2.33)

$$= a_1^* F^*_1(t), \quad 0 \leq t \leq t_1$$

By using Eq. (2.31) and (2.32), the mathematical expression for release 2 to release $n$ may be given by:
Chapter 2

Multi Up-Gradation Software Reliability Model With Fault Severity and Imperfect Debugging

We proposed SRGM2 on the base of the Eq. (2.30), (2.31) and (2.32) by incorporating two type of imperfect debugging on normal distribution function. The normal distribution function is flexible model that can empirically fit many types of failure data(Rausand and Hoyland, 2004). The cumulative normal distribution function is given by:

\[
F_i(t) = \left[ 1 - \left( \frac{(1 + \beta_i) e^{-b_i(t-t_{i-1})}}{1 + \beta_i e^{-b_i(t-t_{i-1})}} \right)^{\alpha_i(1-a_i)} \right]
\]

and \( a_i = \frac{a_i}{1-a_i} \); \( i = 3..n \).

2.3.5.2. Multi Up-Gradation Model Based on Normal Distribution. SRGM2

where

\[
m_i(t) = \left( a_i + a_i^* (1 - F_i^*(t_1)) \right) \left[ 1 - \left( \frac{(1 + \beta_i) e^{-b_i(t-t_{i-1})}}{1 + \beta_i e^{-b_i(t-t_{i-1})}} \right)^{\alpha_i(1-a_i)} \right] \nonumber
\]

\[
m_i(t) = (a_i^* + a_i^* (1 - F_i^*(t_1))) F_i^*(t-t_1) , \quad t_1 < t \leq t_2
\]

\[
m_i(t) = (a_i^* + a_i^* (1 - F_i^*(t_1))) F_i^*(t-t_1) , \quad t_{i-1} < t \leq t_i
\]

\[
\begin{aligned}
\Phi(t, \mu, \sigma) &= \int_{-\sigma}^{t} \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} ds \\
&= \Phi_i(t, \mu_i, \sigma_i) = a_i^* \Phi_i(t, \mu_i, \sigma_i), \quad 0 \leq t \leq t_i \\
\end{aligned}
\]

\[
m_i(t) = (a_i^* + a_i^* (1 - F_i^*(t_1))) \Phi_i^*(t-t_1, \mu_2, \sigma_2) , \quad t_i < t \leq t_2
\]
\[ m_i(t) = (a_i^* + a_i^*(1-\Phi^*_{i-1}(t_{i-1} - t_{i-2}, \mu_{i-1}, \sigma_{i-1})).\Phi^*(t-t_{i-1}, \mu_i, \sigma_i), \quad t_{i-1} < t \leq t_i \]

Where \( \Phi(t, \mu_i, \sigma_i) \) is normal distribution and \( a_i^* = \frac{a_i}{1-\alpha_i} ; i = 1..n \).

### 2.3.6. Data Set, Model Validation

As described in section 2.15, the MUSRGMs have been tested on tandem computer (Wood, 1996, Pham, 2006) four-release data set \((n = 4)\).

namely Mean squared error (MSE); Coefficient of multiple determinations \((R^2)\); Bias, Root Mean Square Prediction Error (RMSPE), Variation are used to compare and analyze the fitness of the model.

### 2.3.7. Parameter Estimates and Goodness of Fit Criteria

Estimated value of parameters of each releases are given in Table 2.3(a) and Table 2.3(b) for SRGM1 and SRGM2.

Table 2.4(a) and Table 2.4(b) summarizes the MSE, \(R^2\), Bias, RMSPE and Variation for SRGM1 and SRGM2 for four successive releases of software system.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Releases</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>113.12</td>
</tr>
<tr>
<td>( b_i )</td>
<td>0.1969</td>
</tr>
<tr>
<td>( \beta_i )</td>
<td>0.8960</td>
</tr>
<tr>
<td>( \alpha_i )</td>
<td>0.001</td>
</tr>
<tr>
<td>( \rho_i )</td>
<td>0.7415</td>
</tr>
</tbody>
</table>
### Table 2.3(b): Parameter estimates for normal distribution

<table>
<thead>
<tr>
<th>Parameters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>105.1243</td>
<td>73.79</td>
<td>62.894</td>
<td>42.145</td>
</tr>
<tr>
<td>$\mu_i$</td>
<td>5.66891</td>
<td>3.70193</td>
<td>1.071801</td>
<td>0.40391</td>
</tr>
<tr>
<td>$\sigma_i$</td>
<td>5.34131</td>
<td>3.6468</td>
<td>1.076723</td>
<td>2.05970</td>
</tr>
<tr>
<td>$\alpha_i$</td>
<td>0.01</td>
<td>0.3995</td>
<td>0.013665</td>
<td>0.001</td>
</tr>
<tr>
<td>$\rho_i$</td>
<td>0.851</td>
<td>0.669</td>
<td>0.866399</td>
<td>0.75209</td>
</tr>
</tbody>
</table>

### Table 2.4(a): Goodness of fit criteria for K-G model

<table>
<thead>
<tr>
<th>Criteria</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^2$</td>
<td>0.989</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td><em>Bias</em></td>
<td>0.578</td>
<td>0.522</td>
<td>0.622</td>
<td>-0.006</td>
</tr>
<tr>
<td><em>Variation</em></td>
<td>3.079</td>
<td>2.784</td>
<td>1.447</td>
<td>0.944</td>
</tr>
<tr>
<td><em>MSE</em></td>
<td>9.3462</td>
<td>7.615</td>
<td>2.309</td>
<td>0.8455</td>
</tr>
<tr>
<td><em>RMSPE</em></td>
<td>3.133</td>
<td>2.832</td>
<td>1.576</td>
<td>0.944</td>
</tr>
</tbody>
</table>

### Table 2.4(b): Goodness of fit criteria for normal distribution

<table>
<thead>
<tr>
<th>Criteria</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^2$</td>
<td>0.996</td>
<td>0.994</td>
<td>0.989</td>
<td>0.992</td>
</tr>
<tr>
<td><em>Bias</em></td>
<td>0.0216</td>
<td>0.0536</td>
<td>-0.7983</td>
<td>0.5938</td>
</tr>
<tr>
<td><em>Variation</em></td>
<td>1.2380</td>
<td>2.8914</td>
<td>2.9675</td>
<td>1.5369</td>
</tr>
<tr>
<td><em>MSE</em></td>
<td>1.4565</td>
<td>7.9235</td>
<td>7.3926</td>
<td>2.5903</td>
</tr>
<tr>
<td><em>RMSPE</em></td>
<td>1.2382</td>
<td>2.8919</td>
<td>3.0731</td>
<td>1.6476</td>
</tr>
</tbody>
</table>
Figure 2.10 and Figure 2.11 show the estimated and the actual values of the number of faults removed for four releases of SRGM1 and SRGM2.

**Figure 2.10:** Goodness of fit of releases due to K-G model
Figure 2.11: Goodness of fit of releases due to normal distribution function

2.3.8. Data Analysis

From the Table 2.4(a) and Table 2.4(b) it is clear that the value of $R^2$ (MSE) for SRGM-2 in first release is higher(lower) than that SRGM1 and provides better goodness of fit for first release. Also the value of $R^2$ (MSE) for SRGM-1 in second, third and fourth is higher(lower) than that the value of $R^2$ for SRGM-2.

In addition after the model validation we found that GO and Yamada (Goel and Okumoto, 1979, Yamada et al., 1983, Goel, 1985) model fail to give good result but K-G model (SRGM1) and normal distribution function based (SRGM2) provide better goodness of fit. It can be further observed that the proposed SRGM1 give a better fit to the observed data and are significantly better than the SRGM2. This result also
demonstrates that the SRGM1 can be successfully used to formulate different patterns of environmental factors in SRGMs.