Chapter-5

Irregular Fluctuation Based Successive Software Release Models

One of the most significant and fascinating business developments in software, in recent years is the growing importance of up gradations. Survival in this new business environment calls for delighting the users with quality software products. Consequently the advancements are increasingly seen as an important tool by progressive software organization. It is the intense global competition in the dynamic environment that has lead to a technological substitution of software product in the market. The software developers are trying very hard to project themselves as organizations that provide better value to its customer. One major way they have found to increase the market presence is by offering new functionalities in the software periodically. 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 it operational reliability level desired by the firm, a new version is released. The term upgrade refers to the replacement of a product with a newer version of the same product. It is most often used in computing and consumer electronics, generally

This chapter is based on the following papers:


meaning a replacement of hardware, software or firmware with a newer or better version, in order to bring the system up to date or to improve its characteristics. As the software firms are involved in developing complex software system with a sharp eye on the market competition, the quality of their product is always under check.

Firms that have upgraded their software skillfully adapting the technological advancements into their products and processes have not only survived but prospered. Expanding high technology has had a significant impact on virtually all industries. Today better known software developing companies like Microsoft, IBM, Adobe and Wipro etc. are known for their innovation strategies and frequent introduction of an advanced version of the software. This successful introduction contributes substantially to long-term financial success and is an effective strategy to increase primary demand. It strengthens the competitive position of the company in the market. But the risk is formidable as for most software organizations especially the development are associated with high cost and risks. This is because upgrading a software application is a complex task where 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. 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 (MSNBC, 2005), 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 (Khataneh and Mustafa, 2009; Kapur et al., 2010). Similar gaffes have occurred from important government systems to freeware on the internet. Sometimes Upgrades can worsen a product and user may prefer an older version. Figure 5.1 depicts the various time periods for the releases.
The software failures may be due to errors, ambiguities, oversights or misinterpretations of the specifications that the software is supposed to satisfy, incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems. A plethora of software reliability models have been developed in the literature. (Goel and Okumoto, 1979) proposed a SRGM, which describe the fault detection rate, as a non homogeneous Poisson process (NHPP). Later, based on this model, eminent researchers (Yamada, 1984; Kapur and Garg, 1992; Pham 2006) after bringing some new assumptions, proposed several reliability models in the literature.

The NHPP model (described in detail in Section 1.7.7 of Chapter 1) enables us to characterize software reliability growth process simply by supposing an appropriate mean value function of the NHPP. A number of faults are detected and removed during the long testing period before the system is released to the market. However, the users then find number of faults and the software company then release an updated version of the system. Thus in this case the number of faults that remain in the system can be considered to be a stochastic process with continuous-state space. Several continuous-state space SRGM based on stochastic differential equations of \( du \) type to measure the reliability growth of software have been developed till yet corresponding to discrete-state space NHPP based SRGM. Based on this objective, some new SRGMs that depends on (stochastic differential equations) SDE have been proposed in this

![Figure 5.1: Various Releases](image-url)
The modeling framework is based on unification scheme by Kapur et al (2010) as described in Section 1.8.4; Chapter 1. The present chapter is divided into two sections. The first section 5.1 discusses the successive software release modeling framework in which the numbers of faults removed in a particular release are dependent on just previous release and not on all the earlier releases. The rate of fault removal is taken as to be logistic. Section 5.2 describes an advanced modeling framework with the concept of faults severity. Though these two sections discuss models under different set of assumptions but they have one thing in common i.e. presence of “fluctuation” in the fault detection rate. Applicability and effectiveness of the discussed models have been carried out on Tandem data set. MSE has been used as the measure of ‘Goodness of fit’. It has been observed that the results are fairly accurate and close to the observed values.

5.1 STOCHASTIC DIFFERENTIAL EQUATION BASED MODELING FOR MULTIPLE GENERATIONS OF SOFTWARE

5.1.1 Notations

\(m^*(t)\) or \(E(m(t))\) Expected number of faults detected in the time interval \((0, t]\) during testing phase.

\(m(t)\) Number of faults detected during the testing time \(t\) and is a random variable.

\(w(t)\) One dimensional Weiner process.

\(\beta_i\) constant parameter describing learning in the fault removal rate; \(i=1\) to 4.

\(F(t)\). Probability distribution function for the time testing is done

\(f(t)\). Probability density function for the time testing is done

\(a_i\) Constant representing the initial number of faults lying dormant in the software when the testing starts for \(i\)th release; \(i=1\) to 4.

\(a\) Total fault content( \(a = a_1 + a_2 + a_3 + a_4\) )

\(t_{i-1}\) Time for \(i^{th}\) release \(i=1\) to 4.

\(\sigma\) Positive constant representing magnitude of the irregular fluctuation.
\( b_i \) fault removal per remaining faults; \( i=1 \) to 4.

\( \gamma(t) \) Standard Gaussian white noise.

### 5.1.2 Basic Assumptions

The Proposed models are based on the following assumptions.

1. Software systems are subject to failure during execution caused by a fault remaining in the system.
2. The software fault detection process is modeled as a stochastic process with a continuous state space.
3. Failure rate of the software is equally affected by the faults remaining in the software.
4. All faults are mutually independent from failure detection point of view.
5. No new fault is introduced into the system and the faults are debugged perfectly.
6. The proportionality of fault detection/isolation/correction is constant.
7. Let \( m(t) \) be a random variable which presents the number of software faults detected in the software system up to testing time \( t \). The faults detected in \( t + \Delta t \) are proportional to the mean number of faults remaining in the system.

Recently (Kapur et al., 2010) developed a multi up-gradation reliability model, considering that cumulative faults in each generation 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/up-gradations) 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). Several NHPP based SRGMs have been developed in the literature, treating fault detection process during testing phase as a discrete counting process.
Yamada, Nishigaki, Kimura 2003 asserted that if the size of the software system is large then the number of fault detected during the testing phase becomes large and the change of the number of faults which are detected and removed through each debugging activities becomes sufficiently small compared with the initial fault content at the beginning of the testing phase. Therefore, in order to describe the stochastic behavior of the fault detection process, a stochastic model with continuous state space can be used. (Yamada et al., 2003; Lee, Kim and Park 2004; Yamada and Tamura, 2006) have studied the stochastic behavior of fault detection process described by stochastic process model with continuous state space. Recently Kapur et al 2007 have also proposed a SDE based flexile SRGM.

5.1.3 Successive Software Releases: Model Development

The term upgrade refers to the replacement of a product with a newer version of the same product. It is most often used in computing and consumer electronics, generally meaning a replacement of hardware, software or firmware with a newer or better version, in order to bring the system up to date or to improve its characteristics. Although developers produce upgrades in order to improve a product, there are risks involved—including the possibility that the upgrade will worsen the product. Upgrades of software introduce the risk that the new version (or patch) will contain a bug, causing the program to malfunction in some way or not to function at all. A user may prefer an older version even if a newer version functions perfectly as designed.

A mathematical model to capture the risk involved in introducing new functionalities is being modeled using SDE based modeling approach. We have considered randomness in the detection function and developed a multiple release software reliability growth model. This model captures the faults which are identified at the time of adding features in the software. Also at the time of adding new features in the software, when the testing team checks the code developed earlier for the parent software, it may find some bugs which remain in the software. This model is based on the assumption that software is never bug free and whenever, we are testing software of a higher version for bugs there is always a chance that some bugs are left over in the earlier versions and are
removed at a later stage. The present framework is based on Kapur and Garg Flexible Model. The K-G model can be described by following mathematical structure:

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

(5.1)

where \( m(t) \) is the cumulative number of faults removed in the software by time \( t \); \( a \) is the finite number of fault content present in the software \( b \) is the constant fault detection rate and \( \beta \) is the learning parameter.

### 5.1.4 SDE Based Modeling For Each Release: Framework For Modeling

Several SRGM are based on the assumption of NHPP, treating the fault detection process during the testing phase as a discrete counting process.

A plethora of mathematical models have been discussed in the literature to capture the cumulative number of faults removed in the software. Using the hazard rate approach in deriving the mean value function of cumulative number of faults removed, we have:

Let \( \{m(t), t \geq 0\} \) be a random variable which represents the number of software faults detected in the software system upto testing time \( t \). Suppose that \( m(t) \) takes on continuous real value. The NHPP models have treated the software faults detection process in the testing phase as discrete state space. However, if the size of the software system is large then the number of faults detected during the testing phase also is large and change in the number of faults, which are corrected and removed through each debugging, becomes small compared with the initial fault content at the beginning of the testing phase. So, in order to describe the stochastic behavior of the fault detection process, we can use a stochastic model with continuous state space. Since the latent fault in the software system are detected and eliminated during the testing phase, the number of faults remaining in the software system gradually decreases as the testing progresses.
So the corresponding differential equation is given by:

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

(5.2)

It might happen that the rate is not known completely, but subject to some random environmental effect, so that we have:

\[
r(t) = \frac{f(t)}{1-F(t)} + \text{"noise"}
\]  

(5.3)

Let \( \gamma(t) \) be a standard Gaussian white noise and \( \sigma \) be a positive constant representing a magnitude of the irregular fluctuations. So the differential equation can be written as:

\[
\frac{dm(t)}{dt} = \left[ \frac{f(t)}{1-F(t)} + \sigma \gamma(t) \right](a-m(t))
\]  

(5.4)

The above equation can be extended to the following stochastic differential equation of an \( \text{ito} \) type:

\[
dm(t) = \left[ \frac{f(t)}{1-F(t)} - \frac{\sigma^2}{2} \right](a-m(t))dt + \sigma(a-m(t))dW(t)
\]  

(5.5)

Where \( W(t) \) is a one-dimensional Wiener process, which is formally defined as an integration of the white noise \( \gamma(t) \) with respect to time \( t \). Using the fact that the wiener process \( w(t) \), is a Gaussian process and has the following properties:

\[
\Pr[w(0) = 0] = 1, \\
E[w(t)] = 0; \\
E[w(t)w(t')] = \min[t, t']
\]

And on applying initial condition \( m(0) = 0 \); we get \( m(t) \) as follows:
\( m(t) = a[1 - (1 - (F(t)))e^{-\sigma W(t)}] \) \hspace{1cm} (5.6)

whose expected value gives us

\[ m^*(t) = E(m(t)) = a[1 - (1 - (F(t)))e^{-\frac{\sigma^2}{2}}] \] \hspace{1cm} (5.7)

### 5.1.4.1 Release 1

Testing phase is the important phase in the software development life cycle (SDLC). Testing starts once the code of software is written. Before the release of the software in the market the software testing team tests the software thoroughly to make sure that they remove maximum number of bugs in the software. Although it is not possible to remove all the bugs in the software virtually. 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 perfectly and mathematical equation for it is given as:

\[ m^*_1(t) = a_i F_i(t) \] \hspace{1cm} \text{for } 0 < t < t_i \hspace{1cm} (5.8)

where,

\[ F_i(t) = \left[ 1 - \frac{(1 + \beta_i) e^{-\frac{1}{2} h t (1 + \beta_i)^2}}{1 + \beta_i e^{-h t (1 + \beta_i)^2}} \right] \]

### 5.1.4.2 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, besides this the testing team considers dependency and effect of adding new functionalities with existing system. In this period when there are two versions of the software, while we are in the testing phase of the Release 2, we are also in the operational phase for the
Release 1, (refer figure 5.2 in order to understand the relation between testing phase and operational phase of each release). The leftover fault content of the first version i.e \( a_i(1-F_i(t_i)) \) interacts with new fault detection rate i.e. \( F_2(t-t_i) \). In addition a fraction of faults generated due to enhancement of the features also get removed with this new rate. The mathematical equation of these finite numbers of faults removed can be given by:

\[
m_2^*(t) = (a_2 + a_i(1-F_i(t_i))).F_2(t-t_i), t_i \leq t \leq t_2
\]

(5.9)

where \( F_2(t-t_i) = \left[ 1 - \left( \frac{1 + \beta_2}{1 + \beta_2 e^{-b_2 t}} \right) e^{-b_2 t_i \frac{1}{2} \sigma^2 t} \right] \)

**Figure 5.2:** Testing process for multi release software
5.1.4.3 Release 3

When the new Up-gradation/add-ons is made in the software for the second time again; new lines of code are generated. This new code is integrated with the existing code and a testing is started again. It is known that bugs present in the software are infinite. Therefore during the testing in this phase a lot of bugs which have been left in primitive stage and first up-gradation are removed. This helps in removing more and more bugs from the developed code. Technological changes and stiff competition forces the software developer to add certain more features to the software. Now as discussed above, the testing team starts testing the upgraded system and simultaneously keeps a check on the operational phase of Release 2 as well (Refer Figure 5.2). Therefore, the leftover faults from Release 2 i.e. $a_2 (1 - F_2(t_2 - t_1))$ interact with the new fault detection/correction rate $F_3(t - t_2)$. Besides this the testing team removes the new faults with this new fault detection rate for the existing system. The mathematical equation of these finite numbers of faults removed can be given by:

$$m_3^*(t) = (a_3 + a_2 (1 - F_2(t_2 - t_1))) F_3(t - t_2), t_2 \leq t \leq t_3$$  \hspace{1cm} (5.10)

where,

$$F_3(t - t_2) = \left[ 1 - \left( \frac{1 + \beta_3}{1 + \beta_3 e^{-b_3 t}} \right) e^{-b_3 t + \frac{1}{2} \sigma^2 t} \right]$$

In the above situation, the newly developed code for third release, the code developed for second release are tested and the cumulative numbers of faults are removed with a failure rate of $F_3(t - t_2)$. Thus we say that in the third stage, a finite number of faults are left over from release which are now getting removed with a different testing effort and under different testing conditions governed by the failure distribution.
Chapter 5  Irregular Fluctuation Based Successive Software Release Models

5.1.4.4 Release 4

The process of Up-gradation is an ongoing process. These up-gradation/add-ons keep on continue till the product is there in the market. This experience helps in improving the value of product and also helps in increasing the reliability of the product as more and more faults are removed when testing and integration of code is done. We discuss a case when the new features are added in the software for the third time.

\[ m_4^*(t) = (a_4 + a_3(1 - F_3(t_3 - t_2))).F_4(t - t_3) \quad , t_3 \leq t \leq t_4 \]  \hspace{1cm} (5.11)

\[ F_4(t - t_3) = \left[ 1 - \left( \frac{1 + \beta_4}{1 + \beta_4 e^{-b_4 t}} \right) e^{-b_4 (t - t_3)} \right] \]

5.1.5 Data Set and Data Analysis

To check the validity of the proposed model, it has been tested on software data collected from Tandem computers (Pham and Zhang 2003). The data set presents the failure data from four major releases of software product at Tandem computers. The parameters present in the above sets of equation were estimated using nonlinear least squares (NLLS) by software package SPSS. Estimated value of diffusion parameters of each of the four releases given in Table 5.1. Table 5.2 shows the comparison criterion of the four software releases. From the tables it can be observed that all the model coefficients are highly significant that justifies the approach considered in the modeling. The goodness of fit curves for the proposed model are given in Fig 5.3, 5.4, 5.5 and 5.6 respectively for the four Releases.
Table 5.1: Parameter Estimates

<table>
<thead>
<tr>
<th>Release</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>110.82</td>
<td>124.37</td>
<td>62.5925</td>
<td>44.983</td>
</tr>
<tr>
<td>$b_i$</td>
<td>0.1720</td>
<td>0.2535</td>
<td>0.5684</td>
<td>0.2669</td>
</tr>
<tr>
<td>$\beta_i$</td>
<td>1.2046</td>
<td>3.7784</td>
<td>16.266</td>
<td>2.1116</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>0.0002</td>
<td>0.001</td>
<td>0.001</td>
<td>0.3537</td>
</tr>
</tbody>
</table>

Table 5.2: Comparison Criteria

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Release 1</th>
<th>Release 2</th>
<th>Release 3</th>
<th>Release 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^2$</td>
<td>0.989</td>
<td>0.995</td>
<td>0.996</td>
<td>0.994</td>
</tr>
<tr>
<td>Bias</td>
<td>0.4352</td>
<td>0.3400</td>
<td>0.0762</td>
<td>-0.0509</td>
</tr>
<tr>
<td>$MSE$</td>
<td>8.9742</td>
<td>6.0013</td>
<td>1.7849</td>
<td>1.0711</td>
</tr>
<tr>
<td>Variation</td>
<td>3.0417</td>
<td>2.4925</td>
<td>1.3931</td>
<td>1.0620</td>
</tr>
<tr>
<td>RMSPE</td>
<td>3.0727</td>
<td>2.5156</td>
<td>1.3952</td>
<td>1.0632</td>
</tr>
</tbody>
</table>
Figure 5.3: Goodness of fit of Release 1.

Figure 5.4: Goodness of fit of Release 2.
Figure 5.5: Goodness of fit of Release 3.

Figure 5.6: Goodness of fit of Release 4.
5.2 A STOCHASTIC FORMULATION OF SUCCESSIVE SOFTWARE RELEASES WITH FAULT SEVERITY

In recent years, the dependence on a computer system has become large in our social life. Therefore, it becomes more important for software developers to produce highly reliable software systems. Due to timely demand and competitive nature of the market of software product, firms are frequently releasing their upgraded versions of the base software. Many Software Reliability Growth Model (SRGM) have been developed by software developers and managers in tracking and measuring the growth of reliability. As the size of software system is large and the number of faults detected during the testing phase becomes large, so the change of the number of faults that are detected and removed through each debugging becomes sufficiently small compared with the initial fault content at the beginning of the testing phase. In such a situation, we can model the software fault detection process as a stochastic process with continuous-state space. In this section, we derive a stochastic differential equation of \( it^o \) type based multi-upgradation model with severity of faults and effect of learning. Moreover, we discuss the extension of the modeling framework used in previous section (5.1). Along with consideration that the faults removed are dependent on the present release and on its just previous release only rather than depending on all the earlier releases, 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 Yamada function.

5.2.1 Notations

\[ m(t) \] Number of faults detected during the testing time \( t \) and is a random Variable.

\[ m^*(t) \text{ or } E[m(t)] \] The mean value function or the expected number of faults detected or removed by time \( t \).

\[ f(t) \] Probability density function

\[ F(t) \] Probability distribution function

\[ t_{i-1} \] Time for \( i^{th} \) release (\( i=1 \) to 4).
\[ a_i \quad \text{Initial fault content for } i^{th} \text{ release (i=1 to 4)} \]

\[ a \quad \text{Total Initial fault content in the software} \]

\[ \lambda_i \quad \text{Fraction of simple faults of } i^{th} \text{ release; } i=1,2,3,4. \]

\[ (1-\lambda_i) \quad \text{fraction of hard faults of } i^{th} \text{ release; } i=1,2,3,4 \]

\[ r(t) \quad \text{Time dependent fault detection rate function.} \]

\[ \sigma \quad \text{Positive constant representing magnitude of the irregular fluctuation.} \]

\[ b_i \quad \text{fault removal per remaining faults; i=1 to 4.} \]

\[ \gamma(t) \quad \text{Standard Gaussian white noise.} \]

\[ w(t) \quad \text{One dimensional Weiner process.} \]

**Framework of Modeling**

We first discuss various assumptions used in developing the multi release modeling framework based on SDE.

**5.2.2 Assumptions**

1. The software fault-detection process is modeled as a stochastic process with a continuous state space.

2. The number of faults (both simple and hard) in the software system gradually decreases as the testing procedures go on.

3. Software is subject to failures during execution caused by faults remaining in the software.

4. The faults existing in the software are of two types: simple and hard. They are distinguished by the amount of effort needed to remove them.

5. During the fault isolation, no new fault is introduced into the system and the faults are debugged perfectly.
5.2.3 Outlining Frequent Releases

As more reliance is placed on software systems it is essential that they operate in a reliable manner. The expanding integration of Internet technologies and the growth in electronic commerce have resulted in rising demand for software products. And as computer systems in business and government alike continue to become more sophisticated, growing numbers of software engineers are expected to be needed to implement safeguard and update systems. One major way to increase the market presence is by offering new functionalities in the software periodically. Technological changes are happening very fast and these new innovations which are nothing but the extensions of the base software often behave like a new software in market (Looy et al (2000)). In general when software reaches a level when it attains its operational reliability level desired by the firm, a new version is released. The term upgrade refers to the replacement of a product with a newer version of the same product. Upgrading a software application is a complex task. The upgraded and existing system may differ in the performance, interface and functionality. Only selected components of an application are changed while the other parts of the application continue to function. Although the developers upgrades the software in order to improve the software product, but there is also the possibility that the upgrade version will worsen, that’s why there is risk involved in upgrading the software system.

The traditional software reliability growth model fails to capture the error growth due to the software enhancements in user-end (Refer Figure 1.22 from Chapter 1). 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. Consider Figure 1.23 from chapter 1 which depicts the increase in failure rate due to the addition of new features in the software (Obha (1984), Kapur and Garg (1999), Kapur et al (2010)).

Recently Kapur et.al (2010) developed a multi up-gradation reliability model, considering that cumulative faults in each generation depend on all previous releases and also assumes that fault is removed with certainty. 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/up-gradations) to the existing software system and that there is possibility of different types of faults lying in the system. In this paper we develop the relationship between features enhancement and software faults removal to develop a mathematical model. The mathematical structure identifies the behavior of faults removed during the testing of the software. The model is developed for four software releases in the software. It assumes that when the software is upgraded for the first time, some new functionality is added to the software. The new code written for the software enhancement leads to some faults in the software which are detected during the testing of the software. During the testing of the newly developed code, there is a possibility that the certain faults were lying dormant in the software which were not removed or detected in the just previously released software version. The testing team also removes these faults before releasing the upgraded version of software in the market.

5.2.4 SDE Based Modeling Framework

An SRGM can describe a software fault-detection phenomenon or a software failure occurrence phenomenon in the testing or operational phase by applying stochastic and statistical theories. Especially an NHPP which treats the fault-detection phenomenon as a discrete-state space has been often applied to software reliability growth modeling. The NHPP model enables us to characterize software reliability growth process simply by supposing an appropriate mean value function of the NHPP (Kapur and Garg 1992). A number of faults are detected and removed during the long testing period before the system is released to the market. However, the users then find number of faults and the software company then release an updated version of the system. Thus in this case the number of faults that remain in the system can be considered to be a stochastic process with continuous-state space Kapur et al (2007c).

For modeling we have used a unification approach and for that we have used the hazard rate approach in deriving the mean value function of cumulative number of faults removed. Therefore we assume:
Let \( \{m(t), t \geq 0\} \) be a random variable which represents the number of software faults detected in the software system up to testing time \( t \). Suppose that \( m(t) \) takes on continuous real value. The NHPP models have treated the software faults detection process in the testing phase as discrete state space. However, if the size of the software system is large then the number of faults detected during the testing phase also is large and change in the number of faults, which are corrected and removed through each debugging, becomes small compared with the initial fault content at the beginning of the testing phase. So, in order to describe the stochastic behavior of the fault detection process, we can use a stochastic model with continuous state space. Since the latent fault in the software system are detected and eliminated during the testing phase, the number of faults remaining in the software system gradually decreases as the testing progresses.


\[
\frac{dm(t)}{dt} = \frac{f(t)}{1-F(t)}(a-m(t)) \quad (5.12)
\]

It might happen that the rate is not known completely, but subject to some random environmental effect, so that we have

\[
r(t) = \frac{f(t)}{1-F(t)} + "noise" \quad (5.13)
\]

where, \( r(t) \) is the time dependent fault detection/correction rate.

Let \( \gamma(t) \) be a standard Guassian white noise and \( \sigma \) be a positive constant representing a magnitude of the irregular fluctuations. So equation (1) can be written as:

\[
\frac{dm(t)}{dt} = \left[ \frac{f(t)}{1-F(t)} + \sigma \gamma(t) \right](a-m(t)) \quad (5.14)
\]
The above equation can be extended to the following stochastic differential equation of an \*\ito\*\ type:

\[
dm(t) = \left[ f(t) - \frac{\sigma^2}{2} \right] (a - m(t)) dt + \sigma (a - m(t)) dW(t)
\]  

\[(5.15)\]

Where \( W(t) \) is a one-dimensional Wiener process, which is formally defined as an integration of the white noise \( \gamma(t) \) with respect to time \( t \).

On applying initial condition \( m(0)=0 \); we get \( m(t) \) as follows:

\[
m(t) = a[1 - (1 - (F(t))) e^{-\sigma W(t)}] 
\]  

\[(5.16)\]

As we know that the Brownian motion or Wiener Process follows normal distribution. The density function of \( w(t) \) is given by:

\[
f(w(t)) = \frac{1}{\sqrt{2\pi}} \exp \left\{ -\frac{w(t)^2}{2\sigma^2} \right\}.
\]  

\[(5.17)\]

Using the fact that the Wiener process \( w(t) \), is a Gaussian process and has the following properties:

\[
\Pr[w(0) = 0] = 1,
\]

\[
E[w(t)] = 0;
\]

\[
E[w(t)w(t')] = \min[t, t']
\]

the mean number of detected fault is given as:

\[
m^*(t) = E(m(t)) = a[1 - (1 - (F(t))) e^{-\frac{\sigma^2 t}{2}}]
\]  

\[(5.18)\]

\section*{5.2.5 Software Reliability Models Based on Fault Severity}

In literature, growth curves have been proposed to represent the removal process of different type of faults. In this framework, we assumed that removal of simple faults in nature follows exponential curve. For other faults, which are more severe in nature,
Yamada function has been used to model the fault removal phenomenon. It is assumed that only two types of faults exist in software; 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:

$$\frac{dm(t)}{dt} = b(a_1 - m(t))$$  \hspace{1cm} (5.19)

Eq. (5.19) describes the failure observation, fault isolation and fault removal processes as one stage process.

Solving the differential equation (5.19) under the boundary condition \(m(t = 0) = 0\), we get:

$$m_1(t) = a_1 \left[1 - e^{-b_1 t}\right] = a_1 F(t)$$  \hspace{1cm} (5.20)

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:

$$\frac{dm_1(t)}{dt} = b(a_2 - m_1(t))$$  \hspace{1cm} (5.21)

$$\frac{dm_2(t)}{dt} = b(m_1(t) - m_2(t))$$  \hspace{1cm} (5.22)

The first stage of the two-stage process is given by Eq. (5.21). This stage describes the failure observation process. The second stage of the two-stage process given by Eq. (5.22) describes the fault removal process. Solving, the above differential equations under the boundary condition, \(m_1(t = 0) = 0\) and \(m_2(t = 0) = 0\), we get

$$m_2(t) = a_2 \left[1 - ((1 + b_2 t)e^{-b_1 t}\right] = a_2 F(t)$$  \hspace{1cm} (5.23)

Alternately this equation can also be generated by using the following differential equation.
\[
\frac{dm(t)}{dt} = \frac{b^2 t}{1 + b t} (a - m(t))
\]  

(5.24)

This equation also yields the same result as given by equation (5.23) (Kapur et al 2011)

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

(5.25)

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

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

which can also written in the format given below:

\[
m(t) = \begin{cases} 
  a \left[ 1 - e^{-b_1 t} \right] & \text{for simple faults} \\
  a \left[ 1 - (1 + b_1 t) e^{-b_2 t} \right] & \text{for hard faults}
\end{cases}
\]

(5.26)

Therefore, using \( F(t) \) from equation (5.26), assumption from equation (5.25), equation (5.18) can be re-written as:

\[
m^*(t) \text{ or } E(m(t)) = \lambda_i a_i \left[ 1 - e^{-b_i t + \frac{1}{2} \sigma_i^2 t} \right] + (1 - \lambda_i) a_i \left[ 1 - (1 + b_i t) e^{-b_i t + \frac{1}{2} \sigma_i^2 t} \right]
\]

(5.27)

or

\[
E(m(t)) = \lambda_i a_i F_{i,s}(t) + (1 - \lambda_i) a_i F_{i,h}(t)
\]

(5.28)

Note that the first index in \( F_{i,s/h}(t) \) show the release \( (i) \) and second index show fault severity (s for simple and h for hard).

Refer Figure 5.1, in order to understand the pattern of releasing of all four generations where \( t_{i-1} \) is the time of releasing the new versions. In this model without incorporating the possibility of changing of nature of faults, we consider that simple faults of old code is removed as simple fault in new release and hard faults of older code are removed by new detection/ correction rate for hard faults only.
5.2.6 Modeling for each Release

5.2.6.1 Release 1

Foundation and Structure of software is represented in market in the first release itself. Hence a company pays more attention on it. Therefore, the developer do not cease testing process after release but keeps a check and analyze the feedback of error in current release in the operational phase. This model classifies fault into two types as simple and hard fault. Some of removed faults are simple i.e. $\lambda_1 a F_{i,1}(t)$ and some are hard i.e. $(1-\lambda_1).a F_{i,2}(t)$. Now it should be noted that we can’t remove all the faults lying dormant in the system and so some faults remain in the code when we release software. The mathematical equation of these finite numbers of faults removed is given as:

$$m_i^*(t) = \lambda_1 a F_{i,1}(t) + (1-\lambda_1).a F_{i,2}(t), \ 0 \leq t \leq t_i$$

(5.29)

5.2.6.2 Release 2

Due to fierce competition and technological changes the software developer is forced to add new features to the software. New features added to the software leads to
complexity and increase in the fault content of the software. 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. Thus, we say that while we are in the testing phase for Release 2, we are simultaneously in the Operational phase for Release 1. (Refer figure 5.2 for the detailed view). Therefore, in this period, when there are two versions of the software, the left over simple and hard fault content of the first version interact with new detection/correction rate. 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 rates i.e. $F_{2,1}(t-t_1)$ and $F_{2,2}(t-t_1)$ for simple and hard faults respectively. Thus, using the concept described by Figure 5.7, the resulting equations are as following:

$$m_2^*(t) = \lambda_2, a_2, F_{2,1}(t-t_1) + (1-\lambda_2), a_2, F_{2,2}(t-t_1)$$
$$\lambda_1, a_1, (1-F_{1,1}(t_1)), F_{2,1}(t-t_1)$$
$$+ (1-\lambda_1), a_1, (1-F_{1,2}(t_1)), F_{2,2}(t-t_1); \quad t_1 \leq t < t_2$$

Where,

$$F_{2,1}(t-t_1) = \left[1 - e^{-b_1, (t-t_1) + \frac{1}{2} \sigma^2(t-t_1)}\right]$$

$$F_{2,2}(t-t_1) = \left[1 - (1+b_2, (t-t_1)) e^{-b_2, (t-t_1) + \frac{1}{2} \sigma^2(t-t_1)}\right]$$

Equation (5.30) can be rewritten in the following manner

$$m_2^*(t) = \left\{ \begin{array}{c}
\lambda_2, a_2 + \lambda_1, a_1, (1-F_{1,1}(t_1)), F_{2,1}(t-t_1) \\
+ (1-\lambda_2), a_2 + (1-\lambda_1), a_1, (1-F_{1,2}(t_1)), F_{2,2}(t-t_1); \quad t_1 \leq t < t_2
\end{array} \right.$$
5.2.6.3 Release 3

With two prior Releases in the market, the software firm adds certain more functionality in order to enhance the functioning of software. Like Release 2, in this Release also, the testing team has to perform the task of simultaneous action of keeping an eye on the leftover fault from the previous Release i.e. Release 2 and on the new faults that generate due to new add-ons. Here, the newly generated simple and hard faults are removed with new simple rate i.e. $F_{3,1}(t-t_2)$ and hard rate $F_{3,2}(t-t_2)$ respectively. Also, the remaining simple and hard faults interact with the new simple and hard detection rates respectively. Hence, using Concept from Figure 5.7 and equation (5.18) again the resulting equation can be written as:

$$m^*_3(t) = \begin{cases} \sum_{\rho} \lambda_3.a_3 + \lambda_2.a_2.(1-F_{2,1}(t_2-t_1)) \cdot F_{3,1}(t-t_2) & \text{for } t_2 \leq t < t_3 \\ + \left(1-\lambda_3.a_3 + (1-\lambda_2.a_2.(1-F_{2,2}(t_2-t_1)) \right) \cdot F_{3,2}(t-t_2) \end{cases}$$  \tag{5.32}

Where,

$$F_{3,1}(t-t_2) = \left[1 - e^{-b_3.(t-t_2) + \frac{1}{2}\sigma^2(t-t_2)} \right]$$

$$F_{3,2}(t-t_2) = \left[1 - (1+b_2.(t-t_2))e^{-b_2.(t-t_2) + \frac{1}{2}\sigma^2(t-t_2)} \right]$$

and $p, r, q, s =$ proportion of new simple and hard faults generated due to add ons.

$q, s =$ proportion of remaining faults (simple and hard respectively), left from previous Release i.e. Release 2

The enhancement continuous and the software is upgraded for the third time. On similar basis, as done for earlier Releases, the mathematical equation for Release 4 can be given as:
\[ m_4^*(t) = \left\{ \lambda_4, a_4, (1 - F_{3,1}(t_3 - t_2)) \right\} \cdot F_{4,1}(t - t_3) \quad \text{for} \quad t_3 \leq t < t_4 \\
+ \left\{ (1 - \lambda_4), a_4, (1 - F_{3,2}(t_3 - t_2)) \right\} \cdot F_{4,2}(t - t_3) \] 

(5.33)

where, rates are as in earlier Release i.e Release 3.

5.2.7 Data Set and Data Analysis

To check the validity of the proposed model, it has been tested on software data collected from Tandem computers (Pham and Zhang 2003). The data set presents the failure data from four major releases of software product at Tandem computers. The parameters present in the above sets of equation were estimated using nonlinear least squares (NLLS) by software package SPSS. Estimated value of diffusion parameters of each of the four releases given in Table 5.3. Table 5.4 shows the comparison criterion of the four software releases. From the tables it can be observed that all the model coefficients are highly significant that justifies the approach considered in the modeling. The goodness of fit curves for the proposed model are given in Fig 5.8, 5.9, 5.10 and 5.11 respectively for the four Releases.

---

**Table 5.3: Parameter Estimates**

<table>
<thead>
<tr>
<th>Release</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_i)</td>
<td>115.08</td>
<td>121.99</td>
<td>62.35</td>
<td>42</td>
</tr>
<tr>
<td>(b_1)</td>
<td>0.2</td>
<td>0.3270</td>
<td>0.5954</td>
<td>0.3103</td>
</tr>
<tr>
<td>(b_2)</td>
<td>0.202</td>
<td>0.3286</td>
<td>0.5956</td>
<td>0.3092</td>
</tr>
<tr>
<td>(\lambda_i)</td>
<td>0.551</td>
<td>0.5086</td>
<td>0.4733</td>
<td>0.4419</td>
</tr>
<tr>
<td>(\sigma)</td>
<td>0.257</td>
<td>0.4955</td>
<td>0.749</td>
<td>0.480</td>
</tr>
</tbody>
</table>
Table 5.4: Comparison Criteria

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Release 1</th>
<th>Release 2</th>
<th>Release 3</th>
<th>Release 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^2$</td>
<td>0.988</td>
<td>0.994</td>
<td>0.995</td>
<td>0.994</td>
</tr>
<tr>
<td>Bias</td>
<td>0.3353</td>
<td>0.2737</td>
<td>0.0264</td>
<td>-0.0509</td>
</tr>
<tr>
<td>MSE</td>
<td>9.7684</td>
<td>9.3870</td>
<td>4.7229</td>
<td>1.0711</td>
</tr>
<tr>
<td>Variation</td>
<td>3.1881</td>
<td>3.1352</td>
<td>2.2697</td>
<td>1.0620</td>
</tr>
<tr>
<td>RMSPE</td>
<td>3.2057</td>
<td>3.1471</td>
<td>2.2698</td>
<td>1.0632</td>
</tr>
</tbody>
</table>

Figure 5.8: Goodness of fit for Release 1
Figure 5.9: Goodness of fit for Release 2

Figure 5.10: Goodness of fit for Release 3
Figure 5.11 : Goodness of fit for Release 4

As can be depicted from the graphs, Release 3 is having more S-shape character followed by Release 2 and Release 4 but with lower rates. Only release 1 has exponential shape. Also the value of $\lambda$ in Release 1 and Release 2 is substantial which shows both (simple and hard) faults are in equivalent proportions. We detect more percentage of hard faults in Release 3. The proposed model give very good fit as exhibited by the values of various comparison criteria.