Chapter-6

**Optimal Release Planning of Software**

The quality of the software system usually depends on how much time testing takes and what testing methodologies are used. On one hand, more the time people spend on testing, more the errors can be removed, which leads to more reliable software; however, the testing cost of the software will also increase during the process. On the other hand, if testing time is too short, the cost of the software could be reduced, but the customers may take higher risk of buying unreliable software (Kapur et al 2011, Pham 1999, Zhang 1998). This will also increase the cost during the operational phase, since it is much more expensive to fix an error during the operational phase than during the testing phase. Therefore, it is important to determine when to stop testing, and release the software.

In other words; Reliability, scheduled delivery and cost are the three main quality attributes for almost all software. The primary objective of the software developer’s to attain them at their best values, then only they can obtain long-term profits and make a brand image in the market for longer survival. The importance of reliability objective has escalated many folds as it is a user-oriented measure of quality. Other reasons being, diversified implementation of software in the various domains around the world, critical dependency of the various systems worldwide on computing systems, global

This chapter is based on the following papers:


trades, highest order growth in the information technology and competition. Notwithstanding its unassailable value, there is still no way to test whether software is completely fault-free or can be made fault-free so that the highest possible value of reliability can be attained how long the testing is continued. On the other hand software users’ requirements conflict with the developers. Software users demand faster deliveries, cheaper software and quality product, whereas software developers aim at minimizing their development cost, maximizing the profit margins and meeting the competitive requirements. The resulting situation calls for tradeoffs between conflicting objectives of software users’ requirements with the developers. As a course of best alternative the developer management must determine optimally when to stop testing and release the software system to the user focusing on the users’ requirements, simultaneously satisfying their own objectives. Such a problem is known as software release time decision (SRTD) problem in the literature of software reliability engineering.

Timely release of software provides dual advantage to the developers. First, they obtain maximum returns on their investments, reduce the development costs, meet the competitive goals and increase the organizational goodwill in the market. Second, they can satisfy the conflicting user requirements if the software release time is determined by minimizing the total software cost whereas the goal of reliability is achieved, etc. This implies the advantage of offering the software at an economic price with higher quality level. Delay in software release imposes the burden of penalty cost/revenue loss and the product may suffer from obsolescence in the market. In contrast to this in case of a premature release the developer may have to spend lot of time and effort to fix the software faults after release and suffer from goodwill loss. Hence one must determine the optimal time to release the software before launching in order to reduce the dual losses that can be imposed on the developers related to both early release and late release. Such a problem of software reliability engineering discipline can be formulated as an optimization problem with single or multiple objectives under some well-defined sets of system, technical and management or user-defined constraints.

The optimal release time is a function of several factors, viz., size, level of reliability desired, skill and efficiency of testing personal, market environment, penalty and/or
opportunity loss costs due to delay in release and penalties/warranty cost due to failure in user phase, etc. Software release time determination has remained a prime field of study for many eminent researchers in the field of software engineering, reliability modeling and optimization over the years. The optimization problem of determining the optimal time of software release is mainly formulated based on goals set by the management in terms of cost, reliability and failure intensity, etc. subject to the constraints. Many problems have been formulated and solved by many researchers in the literature (Okumoto and Goel, (1980), Yamada and Osaki, (1987), Kapur and Garg, (1990, 1991), Yun and Bai, (1990), Kapur and Bhalla (1992), Kapur et al (1993, 1994, 1999, 2011), Pham (1996), Pham and Zhang (1999), Huang et al (1999), Huang (2005), Huang and Lyu (2005)) considering different objectives, constraint set and applying various optimization techniques to achieve the best solution. It is to note that in this chapter the release time optimization models make use of software reliability growth models to determine the relationship between the testing progress (in terms of cost incurred, failure exposure or reliability growth) and time.

Consequently we can articulate that in defining important software cost factors, cost model should help software developers and managers answer the following questions:

- What information does a manager or software developer need to determine the release of software from current software testing activities?
- How should resources be scheduled to ensure the on-time and efficient delivery of a software?
- Is the software product sufficiently reliable for release (eg, have we done enough testing)?

This chapter focuses on the formulation of different classes of release time problems, analysis of the formulated problems, problem solution using different optimization techniques and real life applications of the problems. It aims to help answer the above questions by determining the optimal release timing and policy of the software systems. The chapter is divided in two sections. In the first section (Section 6.1) we primarily develop a successive release software reliability growth model based on stochastic
differential equations of $i t^2 o$ type and then make use of this SRGM to formulate a bi-
criterion problem of simultaneously maximizing reliability and minimizing cost. 
Section 6.2 describes an advanced modeling framework for deciding the release time of 
software. Here we have formulated an optimal release planning problem based on 
multi-attribute utility theory considering two conflicting attributes as cost and failure intensity. Furthermore, a trend line for the reliability of the software has also been 
explained to depict the behavior of reliability. 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.

6.1 STOCHASTIC DIFFERENTIAL EQUATION BASED MODELING FOR 
MULTIPLE GENERATIONS OF SOFTWARE AND OPTIMAL RELEASE 
PLANNING

This section is basically an extension done to the Section 5.1 of Chapter 5 where we 
had formulated a successive release framework dependent on just previous release 
incorporating the randomness factor. We had discussed that testing is an efficient way 
to detect and remove faults so as to avoid failure of a software system, but even then 
exhaustive testing is impractical. Therefore, software developers need to decide when 
to stop testing the current release of the software and come up with up-graded version 
of the software system for the customers. Software release planning problems for single 
release software have been discussed and solved in different ways in literature. One of 
these is to find release time so that the total cost of software testing is 
minimized(Yamada et al 1983; Kapur et al 2004). Some of the release time problems 
are based upon maximizing the reliability of software. Models that minimize the 
number of remaining faults in the software or the failure intensity also lie under this 
category (Goel and Okkumoto 1979). Release time problems have also been formulated 
for minimizing cost with minimum reliability requirement or maximizing reliability 
subject to budgetary constraint (Kapur et al 2004). Bi-criterion release policy (Kapur 
and Garg 1990) that simultaneously maximizes reliability and minimizes cost have also 
been studied in literature for single release software.
The above literature review on release planning problems do not take into account the impact of coming up with multi releases of a software in release planning decisions. The framework developed here helps to answer the question of when to stop testing the current release of the software when we have the dual objective of minimizing the cost and maximizing the reliability of the version that has to be released into the market under the constraints of budget and reliability requirement. The formulated problem is solved using genetic algorithm.

6.1.1 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.

\( C_{n1} \) cost incurred on removing a fault during testing phase of \( n^{th} \) release.

\( C_{n2} \) cost incurred on removing a fault after the delivery of \( n^{th} \) release.

\( C_{n3} \) testing cost per unit
6.1.2 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.

Using concept of \( m(t) \) being a random variable to represent the number of faults detected in the software and the model development framework form section 5.1 chapter 5, 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:

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

(6.1)
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.

Again using the facts as described in Section 5.1.4 of Chapter 5, the expected number of faults removed is given by:

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

Using this framework we have a look at the structure of all the four releases.

6.1.3 Modeling Successive Releases

6.1.3.1 Release 1

Let the First Release of software be done at \( t = t_0 = 0 \). It is to note that we can’t remove all faults in a release and some of fault remained in the code when we release software. The mathematical equation of these finite numbers of faults removed is given as:

\[
m^*_1(t) = a_1 F_1(t) \quad 0 < t < t_1
\]

where,

\[
F_1(t) = \left[1 - \frac{(1 + \beta t)(1 + \beta e^{-h_1})}{1 + \beta e^{-h_1/2}}\right]
\]

6.1.3.2 Release 2

The scenario for the modeling of second release is not same as first releases. The modeling framework discuss the removal of the fault arising due to new functionality added to the software as well as by those faults which remained undetected during the testing of previous release. The leftover undetected fault content of the first release may calculated by \( a_1(1 - F_1(t_1)) \). It may be noted that \( F_2(t - t_1) \) represents fraction of fault
detected/corrected during release 2. The mathematical equation of these finite numbers of faults removed can be given by:

\[ m_2^* (t) = (a_2 + a_1(1 - F_2(t_1))).F_2(t - t_1), t_1 \leq t \leq t_2 \]  \hspace{1cm} (6.4)

where \( F_2(t - t_1) = \left[ 1 - \left( \frac{1 + \beta_2}{1 + \beta_2 e^{-b_2 t}} \right) e^{-b_2(t_1 - t)} \right] \)

### 6.1.3.3 Release 3

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} (6.5)

where,

\[ F_3(t - t_2) = \left[ 1 - \left( \frac{1 + \beta_3}{1 + \beta_3 e^{-b_3 t}} \right) e^{-b_3(t_1 - t)} \right] \]

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.
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 i.e the software is released forth time

$$m_4^*(t) = (a_4 + a_3(1 - F_3(t_3 - t_2))).F_4(t - t_3), t_3 \leq t \leq t_4$$  \hspace{1cm} (6.6)

$$F_4(t - t_3) = \left[1 - \left(\frac{1 + \beta_4}{1 + \beta_4 e^{-b_4 t}}\right)\right]^{-b_4 t + \frac{1}{2} \sigma^2 t}$$

### 6.1.4 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 6.1.

Table 6.2 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 6.1, 6.2, 6.3 and 6.4 respectively for the four Releases.
### Table 6.1: 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 6.2: 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 6.1: Goodness of fit of Release 1.

Figure 6.2: Goodness of fit of Release 2.
Figure 6.3: Goodness of fit of Release 3.

Figure 6.4: Goodness of fit of Release 4.
6.1.5 Optimal Release Planning Problem

In developing software with successive releases, one of the most important decisions that the software development firm has to deal with is when to release the upgraded version in the market. This decision depends on the model used for describing the failure phenomenon and the criterion used for determining system readiness. The optimization problem of determining the optimal time of software release can be formulated based on goals set by the management. Firstly, the management may wish to determine the optimal release time such that total expected cost of testing in the testing and operation phase is minimum. Secondly, they may set a reliability level to be achieved by the release time. Thirdly, they may wish to determine the release time such that the total expected cost of the software is minimum and reliability of the software is achieved to a certain desired level. Such a problem is known as a Bi-criteria release time problem. For Bi-criteria release time problem, release time is determined by carrying a trade-off between cost and reliability. In this section, we will formulate such a Bi-criteria release problem for multi-version software.

6.1.5.1 Modeling Cost Function

Assume the firm has to deliver the $n^{th}$ release of the software. Then, the cost function will include cost of removing faults during testing phase of the $n^{th}$ release and cost of failure and removal of faults after the delivery of the $n^{th}$ release and unit cost of testing during the testing phase of $n^{th}$ release. Therefore, if $C_{n1}$ is cost incurred on removing a fault during testing phase of $n^{th}$ release; $C_{n2}$ is cost incurred on removing a fault after the delivery of the $n^{th}$ release of software system; $C_{n3}$ is the testing cost per unit testing time and resources, then the total cost of testing of $n^{th}$ release $C_n$ is given by:

$$C_n(t) = C_{n1}m_n(t) + C_{n2}\left((a_n + a_{n-1}\left(1-F_{n-1}(t_{n-1})\right))m_n(t)\right) + C_{n3}t;$$  \hspace{1cm} (6.7)

Equation (6.7) can be re-written as:
$$C_n(t) = C_n a_n \left[ 1 - \frac{(1 + \beta_n)}{1 + \beta_n e^{-b_n t}} \right] e^{-b_n t + \frac{1}{2} \sigma^2 t} + C_{n2} \left[ a_n + a_{n-1} \left( 1 - \frac{(1 + \beta_{n-1})}{1 + \beta_{n-1} e^{-b_{n-1} (t-t_{n-1})}} e^{b_{n-1} (t-t_{n-1}) + \frac{1}{2} \sigma^2 (t-t_{n-1})} \right) \right]$$

$$a_n \left( 1 - \frac{(1 + \beta_n)}{1 + \beta_n e^{-b_n t}} \right) e^{-b_n t + \frac{1}{2} \sigma^2 t} + C_{n3}$$

(6.8)

6.1.5.2 Reliability Evaluation

Reliability of software is defined as “The probability that the system will not fail during $$(t, t+x)$$ $$(x \geq 0)$$ given that the latest failure occurred at $$t$$”. Therefore software reliability is represented mathematically as

$$R(t) \equiv R(x | t) = \exp^{-(m(t+x)-m(t))}$$

(6.9)

6.1.5.3 Modeling Release Time Problem

An optimal bi-criterion release planning for multi-upgraded software that maximizes the reliability and minimizes the cost of testing of the release that it to be brought into market under the dual constraints of budget and achieving a desired level of reliability is formulated as

$$\max R(x / T)$$

$$\min C(T)$$

$$s.t$$

$$C(T) \leq C_B$$

$$R(x / T) \geq R_0$$

$$T \geq 0, R_0 < 1$$

Alternately, we may write
\[
\max \log R(x/T) \\
\min \overline{C}(T) \\
s.t \\
\overline{C}(T) \leq 1 \\
R(x/T) \geq R_0 \\
T \geq 0, 0 < R_0 < 1
\] (6.11)

where, \( \overline{C}(T) = \frac{C(T)}{C_B} \). The above equation may be reduced to a single objective optimization problem by introducing \( \lambda_i \) \((i = 1, 2)\) the priority for the \( i^{th} \) component.

Thus the previously stated formula is further reformulated as:

\[
\min \text{imise } K(T) = \lambda_1 \overline{C}(T) - \lambda_2 \log R(x/T) \\
s.t \\
\overline{C}(T) \leq 1 \\
R(x/T) \geq R_0 \\
T \geq 0, 0 < R_0 < 1
\] (6.12)

6.1.6 Solution Methodology

The problem developed in equation (6.12) is solved via Genetic Algorithm. (The GA algorithm steps for solving the release planning problem are described in detail in Section 1.13 in Chapter 1)

6.1.7 Numerical Example

As an example here we choose the same data set of four releases taken in section 6.1.4. In this data set first, second and third release have already been into market. The problem formulated in section 6.1.5.3 (equation 6.12) determines when to stop testing the fourth release of the software such that the cost of testing is minimized and reliability is maximized.
In order to determine the optimal release time and optimal resource consumption for the fourth release we make use of the estimated values of the parameters of third and fourth release given in Table 6.1. With these parameter values we solved the following problem using genetic algorithm method. Further we assume $C_1 = 100$, $C_2 = 150$, $C_3 = 50$ and it is desired that at least 0.95 proportion of faults should be removed from 4th release. The budget is assumed to be 10000 units and $x$ in the reliability evaluation is taken as 7 days (i.e. 1 week). The problem is solved using MATLAB software under VC++ (6.0) compiler.

\[
\text{Min } \sum_{t=1}^{T} C_4(t) = C_4 \cdot \left(1 - \left(1 + \frac{1}{e^{b_3(t-t_3)}} \right) e^{-b_4(t-t_3) + \frac{1}{2} \sigma^2(t-t_3)} \right) \\
\quad + C_4 \cdot \left(1 - \left(1 + \frac{1}{e^{b_3(t-t_3)}} e^{-b_4(t-t_3) + \frac{1}{2} \sigma^2(t-t_3)} \right) \right) \\
\quad + C_4 \cdot \left(1 - \left(1 + \frac{1}{e^{b_3(t-t_3)}} e^{-b_4(t-t_3) + \frac{1}{2} \sigma^2(t-t_3)} \right) \right)
\]

\[
\text{Max } R(x/T) = e^{-4(m_4(t+x)-m_4(t))}
\]

subject to

\[
C_4(T) \leq 10000 \\
R(x/T) \geq 0.85
\]

Writing an alternate form for equation (6.13) we have:

\[
\min \overline{C}_4(T) \\
\max \log R(x/T) \\
s.t. \quad \overline{C}_4(T) \leq 1 \\
R(x/T) \geq 0.85
\]

where, $\overline{C}_4(T) = \frac{C_4(T)}{10000}$. 
Assuming that both the objectives in equation (6.14) carry equal importance i.e. 
\[ \lambda_1 = 0.5 \quad \text{and} \quad \lambda_2 = 0.5 \] we have the problem as:

\[
\begin{align*}
\min & \quad K(T) = 0.5\overline{C}(T) - 0.5\log R(x/T) \\
\text{s.t} & \quad \overline{C}(T) \leq 1000 \\
& \quad R(x/T) \geq 0.85
\end{align*}
\] (6.15)

The above problem is solved using GA. The parameters used in GA evaluation are given in Table 6.3. The crossover method taken is simulated binary crossover (SBX), and selection criterion is tournament selection without replacement.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Population Size</th>
<th>Number of Generations</th>
<th>Crossover Probability</th>
<th>Mutation Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>50</td>
<td>25</td>
<td>0.9</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Upon solving the problem the optimal time for stop testing the fourth release came out be 77 week (which is 25 weeks after third release). The reliability of the software was obtained as 0.88 and the optimal minimum cost attained was 8382.72 units.

6.2 A MULTI-ATTRIBUTE APPROACH FOR RELEASE TIME AND RELIABILITY TREND ANALYSIS OF A SOFTWARE

In a Software Development Life Cycle (SDLC) the testing phase is given a lot of importance. But testing cannot be done indefinitely due to many reasons ranging from marketing considerations to increase in cost. Hence it is pertinent to find the optimal release time during testing phase. Firms routinely face the challenging decision of when to stop testing and release the product in the market. It has been observed that the pace
of introduction of new software is much higher in recent years in comparison to any time in the past. Thus to get the competitive edge it is critical to know about the optimal entry time. Too late an entry is likely to lead to significant loss of opportunity. On the other hand early introduction of a product might hinder its growth due to lack of receptiveness of users towards new technology. Many release time problems with optimization criteria like cost minimization, reliability maximization and budgetary constraints etc. have been discussed in the literature. The release time problem discussed in this section depends on the combined effect of cost and the Failure Intensity function; therefore we formulated an optimal release planning problem based on Multi-Attribute Utility Theory (MAUT). A numerical illustration is provided towards the end of the paper. Furthermore, a trend line for the reliability of software has also been plotted to show the behaviour of reliability.

6.2.1 The Modeling Framework Employed

All the release time problems require a functional relationship between the fault exposure and time (preferably calendar time). Software Reliability Growth Models (SRGMs) play an important role due to their ability to predict the fault detection / removal phenomenon during testing. Several classes of SRGMs have been proposed and validated on test data in the literature. One group of models that has been widely used and researched is the Non-homogeneous Poisson Process (NHPP) models. Models under this group are distinguished from each other by the form of the mean value functions of NHPP that describes the failure phenomenon (Bardhan, 2002). The functional forms are either exponential or S-shaped. There exist other exponential and S-shaped models for counting the number of failures or removals without underlying NHPP (Kapur and Garg, 1990, Kapur et al, 2011a, Musa et al, 1987).

Yamada et al. (1984a) proposed the delayed S-shaped model, Ohba (1984) and Kapur & Garg (1990, 2011) also proposed S-shaped models but under different set of assumptions. Several Other researchers have developed S-shaped models under different testing environment (Pham, 2006, Xie, 1991). The general building block of these SRGMS most of which are Non Homogeneous Poisson Process (NHPP) based is the following differential equation:
$\frac{d}{dt} m(t) = b(t)[a(t) - m(t)]$ \hspace{1cm} (6.16)

where, $m(t)$: Cumulative number of faults removed at time $t$, mean value function of NHPP.

$a(t)$: Potential fault content at time $t$.

$b(t)$: Rate of fault detection per remaining faults

For different assumptions regarding testing processes many forms of $a(t)$ and $b(t)$ can be suggested. Differential equation (6.16) can also be suitably modified to capture testing phenomenon more realistically. In most of the SRGMs it has been assumed that the fault detected is immediately removed. But the model proposed by Yamada (1984a) can be described as a two stage phenomenon incorporating the time lag between the failure observation and its subsequent removal. The following system of differential equations describes the same:

$\frac{d}{dt} m_f(t) = b[a - m_f(t)]$ \hspace{1cm} (6.17)

$\frac{d}{dt} m(t) = b[m_f(t) - m(t)]$ \hspace{1cm} (6.18)

where $m_f(t)$: cumulative number of failures at time $t$.

$b$: rate of fault detection per remaining faults (a constant)

$a$: fault content of the software at the beginning of testing (a constant)

On solving the above equation and using the initial condition, at $t=0$, $m(t)=0$:

$m(t) = a[1 - (1 + bt)e^{-bt}]$ \hspace{1cm} (6.19)

Equation (6.19) can be also generated by using unification approach (Kapur et al 2011a) in the following manner:
\[ \frac{d}{dt} m(t) = \frac{b^2 \cdot f}{1 + b \cdot f} [a - m(t)] \]  

where \( m(t), a \) and \( b(t) \) are the functions as defined earlier.

Many related models have been proposed by Kapur et al. Some regarded the release of new software to be depending on all the previous releases (Kapur et al 2010a) and some considered its dependency on just previous release (Kapur et al 2010b, Singh et al, 2011a, 2011b). It should be noted that, in the present case we have treated the 3rd release as a new version of the software and that the bugs of previous release and the current release do not correlate with each other. This is because of the assumption that the leftover uncorrected faults of the previous release (if any) will be counted again in the system.

The testing process of traditional software relies on a specified testing team, where the number of testers is generally stable. Therefore, the constant fault detection rate has become a common assumption, such as in the famous Goel–Okumoto (GO) model (1979). But the two stages Model (Yamada) gives better explanation of the actual scenario i.e. the faults are first detected and then removed. Therefore, to account this, the increasing fault detection rate function is used. This model is S-shaped model in nature, as discussed by (Kapur et al 1999, 2011a and Yamada 1984).

It is a fact that reliability is considered to be the basic building block of all the NHPP models existing in the software reliability engineering literature. But failure intensity function is also as important as reliability. For the present study we have considered that as soon as the failure intensity reaches at the peak, it starts decreasing as the probability of existence of errors in the software on the execution path becomes less over a period of time. Accordingly, it is reasonable to assume that the failure intensity function follows a hump-shaped curve. Furthermore, hump-shapedness of the function justifies the normalization condition of the function. Knowing \( m(t) \), the failure intensity function \( \lambda(t) \) (which describes this special characteristic) can be obtained as:

\[ \lambda(t) = \frac{dm(t)}{dt} \]  

(6.21)
6.2.2 Parameter Estimation

To verify the model we have used Tandem computers failure data set (Pham and Zhang 2003). This data set includes software faults for four separate releases. We have extensively worked on the third release of this data. This is because after checking all the 4 releases, it was noted that release 3 was having maximum S-character and the use of Yamada model further justifies it. The total testing time and number of software failures for each week are recorded. Furthermore, we have used Method of Least Squares and applied statistical package for social science “SPSS” software for evaluation of parameters. The parameters of this release are estimated and the related mean value functions are obtained.

Estimated values of parameters for model in the release are given in Table.6.4. Figure 6.5 shows the estimated values of the number of faults removed for the release. Based on data available given in Table 6.4, the performance analysis is done by the five common criteria $R^2$, Bias, Variation, RMSPE, MSE are shown in Table 6.5.

<table>
<thead>
<tr>
<th>parameters</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>65.790</td>
</tr>
<tr>
<td>$b$</td>
<td>0.61</td>
</tr>
</tbody>
</table>

Table 6.5: Comparison Results

<table>
<thead>
<tr>
<th>Release</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>M.S.E</td>
<td>8.027</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0.991</td>
</tr>
<tr>
<td>Bias</td>
<td>0.00749</td>
</tr>
<tr>
<td>Variation</td>
<td>2.958</td>
</tr>
<tr>
<td>RMSPE</td>
<td>2.147</td>
</tr>
</tbody>
</table>
From the graph we notice the behavior of actual faults data for software release and observe that it is S-shaped in nature. This is further justified by the use of Yamada function to detect the faults in the software. The model give very good fit as exhibited by the values of various comparison criteria.

In the later part of the discussion, we have also forecasted the behavior of reliability of the software and verified it using Laplace Trend Test (Kanoun et al. 1991, Luo et al., 2007).

### 6.2.3 Release Time Problem

Optimal release time determination in the testing phase is a typical application of Software reliability models. Software release time problems have been discussed and solved in different ways. One of these is to find release time so that the total cost incurred during remaining phases (i.e. testing and operational) of the SDLC is minimized (Kapur et al. 1999, Okkumotto and Goel 1979). Some of the release time problems are based upon reliability criterion alone. Optimization models that minimize the number of remaining faults in the software or the failure intensity also fall under this category (Bardhan 2002, Kapur et al. 1999, 2011a, Yamada et al. 1984b). Release
time problems have also been formulated for minimizing cost with reliability requirement or maximizing reliability subject to budgetary constraint (Kapur et al 1999,2011a, Kapur and Garg 1990). Bi-criterion release policy (Kapur et al 1994, 1999, 2011a) simultaneously maximizes reliability and minimizes cost subject to reliability requirement and testing resource availability constraints. Mathematical programming methods have been used to find solutions to such problems.

The quality of a software system is usually managed or controlled during the testing and maintenance phases. If the length of software testing is long, it can remove many software errors in the software system and its reliability increases. However it may cause a significant financial loss for the software company by increasing the testing cost and delay in software delivery. Further, releasing software to market before measuring desired level of failure intensity (which is fixed by the manager) may increase the maintenance cost during operational phase as well as create risk to lose future market. To trade-off between two conflicting objectives, multi-attribute utility theory (MAUT) is applied in our decision model.

6.2.4. Using Multi-Attribute Utility Theory (MAUT) as an Evaluation Approach

Software Engineers are always making decisions. Poor decisions could result in the loss of money, resources, and time. Therefore, it is important to make logical and well reasoned decisions. However, the decision process can prove to be quite complicated, especially when tradeoffs need to be made. Given the complexity of technology and systems, when there are dozens of attributes, there can be hundreds of alternatives to choose from, which can lead to a seemingly infinite number of possible combinations. So, how does one choose the best combination? The use of utility theory in decision making creates a mathematical model to aid the process. It gives the decision maker the ability to quantify the desirability of certain alternatives. Utility theory is for design scenarios where uncertainty and risk are considered (Thurston 2006).

Therefore we can say that, Multi-Attribute Utility Theory (MAUT) is a label for a family of methods. These methods are means to analyze situations and create an evaluation process. The objective of MAUT is to attain a conjoint measure of the attractiveness (utility) of each outcome from a set of alternatives (Thurston 2006).
Thus, the method is recommended when prospective alternatives must be evaluated to
determine which alternative performs best. It is based on certain set of assumptions
(Keeney and Raiffa 1976, Neumann and Morgenstern 1947). It has been used widely
including energy, manufacturing and services, public policy and healthcare. The
MAUT process can provide a framework through which multiple objectives and
uncertainties can be combined to aid managers in making decisions.

6.2.5 Building the Utility Function

To determine the utility value, or the desirability, of the design, there are 6 steps
(Thurston 2006).

1. Selection of Attributes.
2. Verify relevant attribute conditions or bounds.
3. Use the lottery (described below) to determine the designer’s preference.
4. Evaluate Single Attribute Utility function (SAUF)
5. Credit Allotments and trade-off preferences.
6. Converting SAUF into Multi-Attribute Utility function (MAUF).

MAUT has gained a lot of importance in recent years as it represents the scenario of
management appropriately. It has strong theoretical foundations based on expected
utility theory (Fishburn 1970, Li et al 2011). Another importance is that it provides
feasibility to consider the alternative on the continuous scale (Ferreira et al 2009, Li et
al 2011).

In the present study, we have identified two separate utility assessments. The objective
list utilized for this preliminary analysis is minimization of cost and maximization of
measurement failure intensity. A Multi-Attribute Utility Function (MAUF) is defined
as

\[ U(x_1, x_2, ..., x_n) = f\left[u_1(x_1), u_2(x_2), ..., u_n(x_n)\right] = \sum_{i=1}^{n} w_i u_i(x_i) \]  

where, \( \sum_{i=1}^{n} w_i = 1 \)

where, 

U is a multi-attribute utility function over all utility;
\( u_i(x_i) \) is single utility function measuring the utility of attribute \( i \);

\( x_i \) is level of \( i^{th} \) attribute.

\( w_i \) represent the different importance weights for the utilities of attributes

By maximizing the multi-attribute utility function, the best alternative is obtained, under which the attractiveness of the conjoint outcome of attributes is optimized (Li et al 2011).

We now discuss the methodology that has been utilized in formulating the utility function.

### 6.2.5.1 Selection of Attributes

A vital decision problem that firms encounter is to determine when to stop testing and release the software to user. If the release of the software is unduly delayed, the software developer may suffer in terms of revenue loss. The optimization problem of determining the time of software release can be formulated based on goals set by the firm in terms of cost and failure intensity. Using the concept of quantification from Lie et al (2011); the objective of failure intensity \( \lambda(t) \) is formulated as

\[
\max f = \frac{\lambda(t)}{\lambda_{\text{max}}}
\]

(6.23)

Where, \( f \) is the measurement of failure intensity and is taken as to be one of the attributes to be considered in MAUT.

The software performance during the field is dependent on the reliability level achieved during testing. In general, it is observed that longer the testing phase, the better the performance. Better system performance also ensures less number of faults required to be fixed during operational phase. On the other hand prolonged software testing unduly delays the software release.

Considering the two conflicting objectives of better performance with longer testing and reduced costs with early release, GO (1983) proposed a cost function for the total cost incurred during testing given as:
\[ C(T) = C_1 m(T) + C_2 \left[ a - m(T) \right] + C_3 T \]  \hspace{1cm} (6.24)

where,

- \( C_1 \) be the cost of fixing a fault during testing phase.
- \( C_2 \) be the cost of fixing a fault during operational phase.
- \( C_3 \) is the testing cost per unit testing time.
- \( m(T) \) is the expected number of faults removed till time \( T \).
- \( C(T) \) is the total cost in fault removal.

A firm never wants to spend more than its capacity, therefore the next attribute that we consider is:

\[ \text{Min: } C_i = \frac{C(T)}{C_B} \]  \hspace{1cm} (6.25)

where,

- \( C_B \) is the total budget allocated to the firm.

### 6.2.5.2 Selection of Attribute Bounds

The upper and lower bounds of an attribute are chosen by the designer. It is possible to use mathematical optimization techniques to choose the limits, however there is no rule as to the size of the range. The range of the attribute can change the weight of the scaling factors, when using the multi-attribute utility model. SAUF represents management’s satisfaction level towards the performance of each attribute. It is usually assessed by a few particular points on the utility curve (Keeney and Raiffa 1976). In the present study, using the concept of Lei et al (2011), suppose that the single utility function for cost is to be determined, the lowest and highest values of cost are selected first as \( C_4^0 \) and \( C_4^1 \). At these boundary points, we have \( u(C_4^0) = 0 \) and \( u(C_4^1) = 1 \).

### 6.2.5.3 Lottery

The lottery is the step in the process where the designer's preferences are determined. In this step, the designer needs to make a decision between two choices. The first choice is to have the probability \( p \) for the most preferred alternative or \( 1-p \) for the least preferred alternative. The second choice is the absolute certainty of a particular alternative, or the
certainty value, between the most and least preferred. The goal of the lottery is to determine the probability $p$ where the decision maker is indifferent between the two choices. The indifference between the two choices is called certainty equivalence.

### 6.2.5.4 Development of Single Attribute Utility Function (SAUF)

SAUF is obtained by using a set of lottery questions based on certainty equivalence. They are monotonic functions, where the finest outcome is set at 1, and the worst at 0. SAUF are then developed to describe the designer's compromise between the finest and worst alternatives based on the lottery questions.

Many functional forms of utility function exist like linear, exponential etc. An analytical function is typically used for preference description, and exponential functions are usually used to describe its shape. The general form is $u(x) = y_1 + y_2 e^{rx}$, where $y_1$ and $y_2$ are parameters which guarantee the utility is normalized between 0 and 1, and “$r$” is the risk coefficient which shows degree of risk attitude, reflecting rate at which risk attitude changes with different attribute level. It may be noted that we use lottery when there is a preference or indifference between two lotteries. If they are equal to each other, management is risk neutral and the linear (additive) form $u(x) = y_1 + y_2 x$ should be used. Otherwise, if management is not risk neutral then the exponential form will be selected. Furthermore, it is to note that the additive form of multi-attribute utility function is based on the utility independence and the additive independence assumptions (Keeney and Raiffa 1976, Li et al 2011).

The component utility function for attribute $i$ ($u_i$) is assessed by the use of lottery (Li et al 2011, Seung and Zhang 2011). The three data points used to determine the unknown coefficients are obtained from the equation $u(x) = pu(x^o) + (1-p)u(x^*)$, where $x$ is the certainty value, $x^o$ is the best alternative, and $x^*$ is the worst alternative (Refer Fig 6.6.). Given that the utility is scaled between 1 and 0,

$$u(x^f) = 1,$$
$$u(x^w) = 0,$$
\[ u(x) = p. \]

Therefore, to find \( p \), for a given \( x \), the firm needs to ask from decision maker or else use the lottery theory.

### 6.2.5.5 Credit Allotment to Weights

In this section we have discussed about estimation of weight parameter, \( w_j \). The weights are assumed to reflect the relative importance of moving an attribute from worst to finest level. Thus they are defined on ratio scale. Many approaches for obtaining numerical weight have been proposed, including direct trade-off methods, direct judgment of swing weight and lottery-base utility assessment (Keeney and Raiffa 1976, Li et al 2011). By these methods, Management can assign different importance to each attribute. In our case the number of attributes considered are only two and in this case use of the probabilistic scaling (lottery weight) technique is recommended (useful when there is small number of attribute).

Consider two attributes \( C \) and \( f \) as software development cost and measurement of failure intensity. Let \( (f^h, c^h) \) and \( (f^l, c^l) \) denote the finest and worst possible consequence, (see right hand side in Figure.6.6) respectively. There is a certain joint outcome \( (f^h, c^l) \) that comprises of two attributes \( C \) and \( f \) at the best and worst level with probability \( p \) and \((1-p)\), respectively (Li et al 2011, Winterfeldt and Edwards 1986).

![Figure 6.6: Two Choices for determining scaling constants (Source: Li et al (2011))](image-url)
6.2.5.6 Development of Multi Attribute Utility Function (MAUF)

When certain independence conditions are met, a mathematical combination of all the SAUF, with scaling constants, results in the MAUF, which is the overall utility function with all attributes considered. Scaling constants reflect designer's preference on the attributes, which is based on scaling constant lottery questions and preference independence questions. The form of the MAUF function depends upon the particular independence conditions fulfilled by the different SAUF (Keeney and Raiffa 1976). In the present work, the additive form of the MAUF is given as:

$$Max: \, U(f,C) = w_f \times u(f) - w_C \times u(C)$$

$$w_f + w_C = 1$$

(6.26)

where $w_f$ and $w_C$ are the weight parameters for attribute $f$ and $C$ respectively. $u(f)$ and $u(C)$ are the single utility function for each attribute. It may be noted that the $U(f,C)$ function is of Max type and it has been written in terms of $f$ and $C$. From managers point of view, $f$ is to be maximized while $C$ is to be minimized. To synchronize the two utilities together, we put '−' sign before cost utility. By maximizing this multi-attribute utility function, the optimal time to release, $T^*$ will be obtained.

6.2.6 Numerical Illustration

Tandem Data (Pham and Zhang) comprises of four successive releases. The proposed decision model has been validated for its third release. The 3rd version of software is released after 12 weeks. In the developed framework we investigate about optimal time for the release and try to find whether:

- Testing Time for the release is sufficient.
- The software has been under tested.
- The software has been over tested.
To answer these questions, the MAUT as discussed in section 6.2.5 is used. The determination of optimal planning testing time is done using the methodology as described in section 6.2.5.1.

6.2.6.1 Selecting the attributes

In the present problem, two attributes as cost and measurement of failure intensity are selected. These attributes are two important factors for determination of optimal planning testing time of software.

We have already calculated the expected number of faults removed from the software (equation 6.19). On further differentiating it we get the intensity function \( \dot{\lambda}(t) \).

\[
\dot{\lambda}(t) = b^2 t e^{-bt}
\]

It is worth noting that \( \dot{\lambda}(t) \) reaches its maximum value \( \dot{\lambda}_{\text{max}} = be^{-1} \) at \( t_{\text{max}} = \frac{1}{b} \). From here, we find out \( \max f = \frac{\dot{\lambda}(t)}{\dot{\lambda}_{\text{max}}} \) as defined earlier (equation 6.23).

Although maximizing failure intensity is important but in several cases, if this attribute is used as a solitary attribute, it might cause risk for company and users as well. Based on this idea, manager uses reliability attribute as risk-relief measure involved with the project. For other attribute i.e. cost we use the cost model as discussed earlier in Section (6.5.1.1)

\[
\text{Min: } C_4 = \frac{C(T)}{C_B}
\]

We set parameters \( C_1 = 15, C_2 = 18, C_3 = 3 \) and \( C_B = 2500 \) as parameters of cost function. The cost function is then calculated using the value of estimated parameters as given in the Table.6.4.

6.2.6.2 Selecting the Bounds

The single utility function for each attribute is elicited based on the management’s strategy. In our numerical example, management strategies are given as:
• For failure intensity attribute, management has verified that at least 60% of software faults should be detected; its highest expected value is 100%.

• Under minimization cost strategy, management indicates that at least 50% of budget must be consumed.

• Management demonstrates its risk neutral attitude for each attribute.

According to the above strategy, some important points on the utility curve are obtained. In particular, the lowest budget consumption requirement is $C_w^4 = 0.5$ and the highest budget consumption $C_b^4 = 1$. The lowest failure intensity requirement is $f_w^4 = 0.1$ and the highest reliability for this release considered as $f_b^4 = 0.6$.

### 6.2.6.3 Using Lottery Theory to elicitate SAUF

The linear form of the single utility function is selected, based on management’s risk neutral attitude towards these two attributes and simple structure which is applicable in several areas (Li et al 2011). By using the concept from section 6.2.5.4 parameters $y_1$ and $y_2$ are determined. Specifically, we have the following equations:

$$u(C_4) = 2C_4 - 1$$
$$u(f) = 2f - 0.2$$

### 6.2.6.4 Crediting the weights

In this stage, the weight parameter $w_c$ is estimated by comparing the two choices in Figure.6.6, by lottery approach (Keeney and Raiffa 1976, Li et al 2011). Management has claimed that it is indifferent between these two choices when $p$ is equal to 0.5, hence $w_c = 0.5$. It is easy to calculate $w_f$ based on the sum of weight parameters equal to one, therefore $w_f$ is also equal to 0.5.

### 6.2.6.5 Developing MAUF

Here, based on the single utility functions and the weight parameters which have been determined in previous steps, the MAUF is evaluated and is shown in Figure.6.7.
Chapter 6

Optimal Release Planning of Software

\[ \text{Max} u(f, C_i) = w_j \times u(f) - w_{C_i} \times u(C_i) \]

\[ w_j + w_{C_i} = 1, \]

\[ \frac{C(T)}{C_B} \leq 1 \]  \hspace{1cm} (6.27)

The above function is maximized by using Maple package and the optimal time to release comes out to be \( T^* = 16.542 \). Figure 6.6 shows the multi attribute utility function. From the curve it can be noted that the value of utility function starts to decline after reaching time around 16 (that is why we consider the optimal time of release to be this). Figure 6.7 represents the behavior of the cost function. According to Tandem data failure, real time to 3\(^{rd}\) release is 12 weeks. Figure 6.8 depicts the hump-shaped failure intensity function. After it reaches to the highest value, it starts to decline and gives the graph the present shape. Based on optimal result, we can say that software in this release should be kept under testing for around four more weeks.

Figure 6.6: The multi-attribute utility function against time
Figure 6.7: The behavior of cost function

Figure 6.8: Behavior of failure intensity function
Further, we have also taken into consideration the major concern of software development firm’s appropriate time for release of software with the desired level of reliability. In order to look into this problem and determine whether the software underwent a reliability growth or decrease during the testing, we have applied Laplace trend test to the failure data (Kanoun et al 1991, Luo and Bergender 2007).

This concept is very important to analyze behavior of reliability in testing phase for determining appropriate time for any release.

### 6.2.7 Laplace Trend Test

In order to determine whether the software underwent a reliability growth or not, we have applied the Laplace trend test to the failure data. Since the Yamada model is failure count model, we divided the time interval \( (0, t] \) into \( k \) units of time of equal length and defined the formulation of the Laplace trend factor, \( u(k) \), as follows:

\[
u(k) = \frac{\sum_{i=0}^{k} (i-1)n(i) - \frac{k-1}{2} \sum_{i=0}^{k} (n(i))}{\sqrt{\frac{k^2-1}{12} \sum_{i=0}^{k} (n(i))}}
\]  

(6.28)

where \( n(i) \) is the number of failures observed during unit time \( i \).

Positive values of \( u(k) \) indicate a decrease in terms of software reliability, whereas negative values indicate reliability growth (Kanoun et al 1991, Luo and Bergender 2007).

Figure 6.9 shows the Laplace trend test results for the third release of the data set, as reported by tandem computer (Pham and Zhang 2003).
We can see that $u(k)$ values are beginning with negative from beginning. However, in period from 2\textsuperscript{nd} to 5\textsuperscript{th} we see some fluctuations but this fluctuation doesn’t affect the reliability much and the trend line has decreasing behavior from 5\textsuperscript{th} week onwards and after 8\textsuperscript{th} week, this behavior completely stabilizes which means that reliability grew monotonically.

**Figure 6.9: Laplace Trend Test**