Reliability Analysis and Optimal Release Time for a Software Using Multi-Attribute Utility Theory

Ompal Singh¹, P. K. Kapur² and Deepti Aggrawal¹

¹ Department of Operational Research, University of Delhi, Delhi-110007, India
E-Mail: drompalsingh@live.com, deepti.aggrawal@gmail.com
² Amity International Business School, Amity University Uttar Pradesh, Noida, India
E-Mail: pkkapur1@gmail.com

Summary
Growing international competition has increased the need for all developers to ensure the level of quality and reliability of their products at the lowest cost before release. Software users demand faster deliveries, cheaper software and quality product whereas software developers aim at minimizing their developing cost and meeting the competitive requirements. The resulting situation calls for trade-off between the two. Therefore, it is necessary to manage the software development process in terms of reliability and scheduled delivery. A new idea for the software release mechanism is presented in this paper. To estimate and predict software reliability, the structure of SRGMs developed in past is utilized to formulate an optimization model for software release time decision considering the effect of fault generation and imperfect debugging with learning process. The paper proposes a decision theoretic approach, Multi Attribute Utility Theory (MAUT), to help the testing team in deciding how long to allow for the testing and release it to the user. The objective of this paper is to determine the optimal release time based on the optimization criteria like maximizing the release index, cost minimization and budgetary constraints etc. Actual software reliability data has been used to demonstrate the capability of the model. The result shows that the proposed model is a useful and effective decision making tool for finding out the release time.

Key words: Non-homogenous Poisson process, optimal release time, software reliability, Hump-Shaped curve, rapid release index, multi attribute utility theory (MAUT).

1. INTRODUCTION

As computer applications permeate our daily life, reliability becomes a very important characteristic of the computer systems. Software reliability is consequently one of the most important features for
a critical software system. According to the ANSI definition [17], software reliability is defined as the probability of failure-free software operation for a specified period of time under a specified environment. In practice, it is very difficult for the project managers to measure software reliability and quality.

The primary objective of the study of software reliability is to increase the probability so that a completed program will work as intended by the customer. Hence measuring and computing the reliability of a software system is very important. Many companies are spending enormous amount of cost during development to achieve the good quality software product. Taking it into consideration, many researchers have been involved in studies on predicting number of defects in the system. In the past few decades, many software reliability growth models (SRGM) had been developed to evaluate the software reliability [18, 29, 31]. These SRGM provide the mathematical relation between time span and cumulative faults which are discovered during the testing. Various NHPP based SRGMs have been built upon various assumptions. Goel and Okumoto [24] proposed an exponential SRGM, which is characterized by the time and cumulative number of failures. Yamada [34] has proposed a NHPP S-shaped software reliability growth model, called as delayed S shaped model and Inflection S shaped model.

The testing of modern software systems to remove remaining defects is a difficult and costly business. A problem of great importance in the development of software system is when to stop testing and release the system. The longer the testing phase is, the higher the reliability is, and the smaller the operation cost is. However, delays in software release increase testing and other costs. Hence, the optimum length of the testing phase can be determined by trading-off the two competing facets. A model for software reliability assessment during testing phase is called a software reliability growth model (SRGM). Optimal release problems based on a SRGM have been studied by many authors [10, 14, 22, 23, 31]. In these models, the software system is developed, tested, its errors are corrected and it is released to operational phase (users end) at time T.

Most SRGM are characterised by the mean value function of non-homogeneous Poisson process and uses the past failure data collected during testing phase to predict the quality of the software. Generally software testing is not always perfect in nature, testing is influenced by several factors. Due to the complexity nature of faults during testing, faults are not completely removed. There is a chance that one fault may influence another one, which cause for additional faults in the software.

The objective of this paper is to find an optimal timing of releasing the software in the market incorporating the concept of MAUT. An overview of previous NHPP models and a brief description of this proposed model are given in section 2 assuming that failure intensity proportional to the faults remains in the software. Section 3 evaluates the developed model followed by the parameter estimation for a software release data set. We have further analysed software release time in section 4 for the Tandem Computers [29] for their forth release using the utility theory based on release frequency and cost criteria. In section 5, a numerical illustration is presented. Section 6 examines the reliability of the software using Laplace trend analysis.

2. REVIEW OF SRGMS (IMPERFECT DEBUGGING ENVIRONMENT)

Due to high cost of fixing failures, safety concerns, and legal liabilities, organizations need to produce software that is reliable. Debugging process is usually imperfect because during testing all software faults are not completely removed as they are difficult to locate or new faults might be introduced. Hence, it is of great importance to investigate the effect of the imperfect debugging on software development cost, which, in turn, might affect the optimal software release time or operational budget. In real software development environment, the number of failures observed
need not be same as the number of errors removed. If the number of failures observed is more than the number of faults removed then we have the case of imperfect debugging. Due to the complexity of the software system and the incomplete understanding of the software requirements, specifications and structure, the testing team may not be able to remove the fault perfectly on detection of the failure and the original fault may remain or get replaced by another fault. The first phenomenon is known as imperfect debugging of type-I, the second is called fault generation which can also be termed as another type of imperfect debugging, we call it imperfect debugging of type-II.

The concept of imperfect debugging was first introduced by Goel [5]. He introduced the probability of imperfect debugging in Jelinski and Moranda [6]. Kapur and Garg [8] introduced the imperfect debugging in Goel and Okumoto. They assumed that the FRR per remaining faults is reduced due to imperfect debugging. Thus the number of failures observed by time infinity is more than the initial fault content. Although these two models describe the imperfect debugging phenomenon yet the software reliability growth curve of these models is always exponential. Moreover, they assume that the probability of imperfect debugging is independent of the testing time. Thus, they ignore the role of the learning process during the testing phase by not accounting for the experience gained with the progress of software testing. Actually, the probability of imperfect debugging is supposed to be a maximum in the early stage of the testing phase and is supposed to reduce with the progress of testing. All these models are continuous time models.

2.1 Model Formulation

In this section we review the well-known NHPP SRGMs and introduce our new general model that incorporates the imperfect debugging concept. Let \( \{N(t), t \geq 0\} \) be a counting process representing the cumulative number of software failure by time \( t \). The \( N(t) \) process is shown to be a NHPP with a mean value function \( m(t) \) which can be obtained by solving the following differential equation:

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

where \( m(t) \), \( a(t) \), \( b(t) \) and \( \rho \) are the mean value function of detected faults by time \( t \), the fault content function, time dependent fault detection rate function and probability of perfect debugging respectively. Given different expressions and explanations of \( a(t) \) and \( b(t) \), many SRGMs can be developed [25].

Also, due to error generation, we have considered the following form of \( a(t) \) for the proposed model:

\[
a(t) = a + \alpha m(t)
\]  

where \( a \) denotes the total fault content in software and \( \alpha \) represents the rate at which the faults may be introduced during debugging process.

The basic assumption illustrated by the above formulation is that the failure rate at current time is determined by the product of the fault detection rate function and the number of remaining faults. The fault detection process of 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 model [24]. Moreover, to account for the
"learning" effects of the testing team, the increasing fault detection rate function is used and these models are S-shaped models as discussed in [32, 34].

As soon as the fault detection rate 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 fault detection rate follows a hump-shaped curve. Logistic model [26] is one of the most popular models in software reliability area and can fit on many types of failure data of testing phase not only because of it has good flexible nature to describe the hump-shaped curve, it can also provide a more reasonable starting point with a non-zero value. In order to describe this special characteristic, the first derivative of logistic function is selected [15] and it is given by:

\[ b(t) = \frac{b^2 \beta e^{-bt}}{(1 + \beta e^{-bt})^2} \]  

(3)

where, \( b \) and \( \beta \) are the proportionality constant failure detection rate and the learning parameter respectively. It is worth noting here that \( b(t) \) reaches its maximum value \( b_{\text{max}} = \frac{b^2}{4} \) at:

\[ t_{\text{max}} = \frac{\ln(\beta)}{b} \]  

(4)

The mean value function for the proposed model is taken on the basis of Li et al (2011) as follows:

\[ \frac{dm(t)}{dt} = B'(t) \rho (\alpha(t) - m(t)) \]  

(5)

where, \( B'(t) = B(t) - B(0) \) and \( B(t) \) is the integration of \( b(t) \) (given by equation 3) over the time period \((0, t]\).

Solving the differential equation (5) under the boundary condition \( m(t = 0) = 0 \), we get:

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

(6)

3. PARAMETER ESTIMATION

The success of mathematical modelling approach to reliability evaluation depends heavily upon quality of failure data collected. The parameters of the SRGMs are estimated based upon these data. Hence efforts should be made to make the data collection more explicit and scientific. It is more difficult to find the solution for on linear models using Least Square method and requires numerical algorithms to solve it. Statistical software packages such as SPSS helps to overcome this problem. SPSS is a comprehensive and flexible statistical analysis and data management system. For the estimation of the parameters of the proposed model, SPSS Regression Model has been used.

For validation of the proposed model, failure data set is cited from Tandem computers [29] that include software faults for four releases. The fourth release data has been examined specifically.
Estimated values of parameters for the developed model are given in Table 1. Figure 1 shows the estimated values of the number of faults removed for the releases. Based on data available given in Table 1, the performance analysis of the proposed model is done by the five common criteria $R^2$, Bias, Variation, RMSPE, MSE [8,9,25] and shown in Table 2.

### Table 1. Parameter estimation

<table>
<thead>
<tr>
<th></th>
<th>Release 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>45</td>
</tr>
<tr>
<td>$b$</td>
<td>0.36</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.001</td>
</tr>
<tr>
<td>$\beta$</td>
<td>5.29</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0.39</td>
</tr>
</tbody>
</table>

### Table 2. Comparison results

<table>
<thead>
<tr>
<th></th>
<th>Release 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>M.S.E</td>
<td>2.14</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0.991</td>
</tr>
<tr>
<td>Bias</td>
<td>0.527</td>
</tr>
<tr>
<td>Variation</td>
<td>1.4</td>
</tr>
<tr>
<td>RMSPE</td>
<td>1.49</td>
</tr>
</tbody>
</table>

The fitting of the proposed model to actual data is graphically illustrated in Figure 1. It is clearly seen from the figure that the proposed model fits given DS quite accurately. From the above plot, it can be examined that the behaviour of actual faults data for software releases is S-shaped in nature. The result is encouraging in terms of goodness of fit and predictive validity due to their applicability and flexibility.

![Figure 1. Release actual vs. predicted](image)

### 4. DETERMINING OPTIMAL RELEASE TIME

No software can be tested indefinitely in order to make it bug free since users of the software want faster deliveries and constraint on development cost. An important decision problem of practical concern is to determine when to stop testing and release the software system to the user known as "Release Time Problem". 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 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. Many researchers in literature have studied various release time problems for different SRGMs [6, 8, 9, 12]. Min Xie [30] attempted to determine the optimal release time of software using the SRGM proposed by Obha and Chou [20], incorporating the second type of imperfect debugging i.e. error generation.

In this paper, we determine the optimal release time of software using the imperfect fault-debugging model. We have included cost of fixing an error due to imperfect fault debugging during testing phase in the cost model and determined the release time. If we want the reliability level to be achieved by the optimal time of software release, we should incorporate the desired reliability level either as a constraint of release time problem or as an objective of Bi-criteria release time problem. However, we may not obtain a minimum cost at the desired reliability level; therefore release time is determined by a trade-off between rapid release index and cost by applying multi-attribute utility theory (MAUT) in our decision model.

4.1 Multi-Attribute Utility Function Approach

Multi-Attribute Utility Theory (MAUT), developed by Keeney and Raiffa in 1976 [12], is used to rank alternatives or make a choice against two or more attributes based on expected utility theory. Expected utility theory states that if an appropriate utility is assigned to each possible consequence and the expected utility of each alternative is calculated, then the preferred decision is the option whose expected utility has the highest value. In this theoretical framework, the technique decomposes the multi-attribute utility function into a more practical form for elicitation based on a set of assumptions, namely preferential independence and utility independence [13]. The concept of utility independence can be viewed as a special condition of the concept of preferential independence [12]. This suggests that the preferential independence assumption is a weaker assumption than utility independence. In MAUT, each alternative is assessed for desirability using a number of attributes. Utility function is a device which quantifies the full range of uncertainty and the decision maker's attitude toward risk by assigning a numerical index to varying levels of criterion satisfaction [12]. The decision maker’s preference attitude is reflected in the shape of the utility curve. The following five steps describe the process of the MAUT application:

1. Setting an objective and establishing attributes.
2. Determining the multi attribute utility function form.
3. Deriving the single utility functions for each attribute.
4. Calculating the scaling constants in the multi attribute utility function.
5. Aggregating single utility functions.

Under multi-attribute utility theory, a decision maker formulates his or her preferences in terms of a scalar function, called as utility function, over the domain of attribute values. This function defines the decision maker's preferred attribute values and the trade-offs between different attributes. Uncertainty is modelled probabilistically as a distribution over the possible attribute values for a given action on the part of a decision maker. For a properly constructed utility function, the most preferred action of the decision maker corresponds to the action that maximizes expected utility. In
the same regard, two separate utility assessments have been identified. The objective list utilized for this preliminary analysis is minimization of cost and maximizing of rapid release index. Mathematically, one can state an objective according to multi-attribute utility theory as:

\[
U(d_1,d_2,...,d_n) = f[u_1(d_1),u_2(d_2),...,u_n(d_n)] = \sum_{i=1}^{n} \lambda_i u_i(d_i)
\]

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

(7)

Here, \( U \) is a multi-attribute utility function over all utility. \( u_i(d_i) \) is single utility function measuring the utility of attribute \( i \); and \( d_i \) is level of \( i^{th} \) attribute. The scaling constants \( \lambda_i \) represent the different importance weights for the utilities of attributes (also called the relative importance). By maximizing the multi-attribute utility function, the best alternative is obtained, under which the attractiveness of the conjoint outcome of attributes is optimized [15]. 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 [4, 15]. Another importance is that it provides feasibility to consider the alternative on the continuous scale [2, 15].

4.2 Developing the Utility Function

To formulate the utility function, a five step methodology (as discussed in Sec 4) has been used i.e. firstly, we establish the attributes. Secondly, the component utility functions (\( u_i \)) are assessed. And then weight parameters are evaluated and finally the best alternative is obtained by maximizing the multi-attribute utility function.

I): Establishing the Attributes

A vital decision problem that firms encounters 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 release index and cost. Using the concept of quantification from [3, 4]; the objective of rapid release index \( \beta \) is formulated as:

\[
\max \beta = \frac{b(t)}{b_{\max}}
\]

(8)

where, \( \beta \) is the rapid release index and is taken as to be one of the attributes to be considered in MAUT. Particularly, a maximum value of it indicates a rapid release and it reaches its maximum at the time \( t_{\max} \).

The software performance during the field is dependent on the reliability level achieved during testing. In general, it is observed the 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 [23] proposed a cost function for the total cost incurred during testing given as:
\[ C(t) = C_1 m(t) + C_2 (a - m(t)) + C_3 t \]  

(9)

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 another attribute that we consider in MAUT is Cost:

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

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

**II): Actuary of Components Utility Functions**

Here we have discussed about assigning values to the components of utility function. The single utility function for each attribute represents management’s satisfaction level towards the performance of each attribute. It is usually assessed by a few particular points on the utility curve [12]. The component utility function for attribute \( i, u(i) \) is assessed by the use of lottery [15, 26] as following:

\[ u_i(d_i^{CE}) = p u_i(d_i^{MU}) + (1-p) u_i(d_i^{LU}) \]  

(10)

To find \( p \), for a given \( d_i^{CE} \), the firm needs to ask from decision maker or else use the lottery theory. Three data points obtained from the above equation are used to determine the unknown coefficients in the utility function. The three data points are \( u_i(d_i^{MU}) = 1, u_i(d_i^{LU}) = 0 \) and \( u_i(d_i^{CE}) = p \). More specifically, using the concept of Li et al [15], suppose that the single utility function for cost is to be determined, the least and maximum values of cost are selected first as \( C^0 \) and \( C^1 \). At these boundary points, we have \( u(C^0) = 0 \) and \( u(C^1) = 1 \).

Many functional forms of utility function exist like linear, exponential and many more. It finally, needs to examine through interviews, surveys or lottery, to determine functional form of utility functions either an additive or exponential form [see Eq.(11)]. 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 form should be used. Otherwise, if management is not risk neutral then the exponential form will be selected:

\[ u(C)=x+yC \text{or } u(C)=x+yC^n \]  

(11)

Where, \( x, y \) and \( n \) are constant parameters which guarantee the normalization of utilities between 0 and 1. Note that the additive form of multi-attribute utility function is based on the utility independence and the additive independence assumptions [12,13].

**III): Evaluating the Scaling Constants in the Multi Attribute Utility Function**


Keeney and Raiffa [12] emphasized that scaling constants are not importance weights for attributes but rather are re-scaling factors that make single attribute assessments consistent with the overall assessment. There are many sophisticated ways to assess the scaling constants. Probabilistic scaling method, one of the most widely used methods, was employed in the present study. In probabilistic scaling method, the scaling constants are first arranged in descending order of importance. Then the decision maker is asked to compare a certain scenario and a lottery. The certain scenario is comprised of the most important attribute $d_j$ at its best level and the others at their worst levels. The lottery is comprised of all attributes at their best levels with probability $p$ ($0<p<1$) and with all attributes at their worst levels with probability $(1-p)$. The decision maker is asked to give the value $p$ such that the respondent is indifferent between the certain scenario and the lottery. Equating the utilities of the two scenarios, it follows that $p$ equals to the scaling constant $\lambda_i$ for the most important attribute $d_j$. To obtain other scaling constants, assessments based on pair wise trade-offs were recommended, because it provides more reliable and meaningful responses than assessments with multi-attribute lotteries[12]. Then trade-offs between a pair of attributes are established to explore the relationships of the two scaling constants by finding the indifference point in decision problems [1]. The procedure is continued to obtain as many equations as required to obtain all scaling constants [28].

Consider two attributes $C$ and $\beta$ as software development cost and rapid release index. Let $(\beta^H, C^u)$ and $(\beta^l, C^l)$ denote the most and least possible consequence, (see right hand side in Figure.2) respectively. There is a certain joint outcome $(\beta^M, C^L)$ comprised two attribute $C$ and $\beta$ at the maximum and lowest level with probability $p$ and $(1-p)$ respectively. In these situations, the weight for attribute $U$ equals $p$, where $p$ is the indifference probability between them [13, 27].

![Figure 2. Two Choices for determining scaling constants (Source: Li et al [15])](image)

IV): Maximization of Multi-Attribute Utility Function

Finally, in this step we can calculate MAUF based on the previous steps. The additive form of the MAUF in our problem is given as:

$$\text{Max}: \quad U(\beta, C) = \lambda_\beta u(\beta) - \lambda_C u(C)$$

$$\lambda_\beta + \lambda_C = 1$$  \hfill (12)$$

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

5. NUMERICAL EXAMPLE

Tandem Data [29] comprises of four successive releases. The proposed decision model has been validated for its fourth release. The fourth version of software is released after 19 weeks. In this paper we examine optimal time for the release and try to find whether the testing time for the release is insufficient or the software has been under/over tested.

I): Quantifying the Attributes

In the present problem, two attributes as cost and rapid release index are selected. These attributes are two important factors for determination of optimal planning testing time of software. As already defined in Section 4, \( \beta \) is given by \( \beta = \frac{b(t)}{h_{max}} \).

Also it is worth noting that \( b(t) \) reaches its maximum value \( b_{max} = \frac{b^2}{4} \) at \( t_{max} = \frac{\ln(\beta)}{b} \).

Therefore, \( \beta = \frac{4\beta e^{-bt}}{(1 + \beta e^{-bt})^2} \)

Although releasing the software quickly is important but in several cases, if this attribute is used as solitary attribute, it might cause risk for company and users as well. Based on this idea, manager uses this 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 4.2.

Min: \( C = \frac{C(t)}{C_{\beta}} \)

We set \( C_1 = 15, C_2 = 25, C_3 = 5 \) and \( C_4 = 4500 \) as parameter of cost function. The cost function is then calculated by the value of estimated parameters given in the Table 1.

II): Assessment of Components Utility Functions

The single utility function is elicited based on the management’s own strategy for each attribute. In our numerical example, management scenarios are given as:

- Management demonstrates its risk neutral attitude for each attribute.
- Under the rapid release strategy, management has verified that at least 60% of software faults should be detected at time \( t_{max} \) and the large the better; its highest rapid release expected value is 100%.
- Considering cost minimization, management indicates that at least 50% of budget must be consumed.

According to the above strategy, some important points on the utility curve are obtained. In particular, the lowest budget consumption requirement is \( C^L = 0.5 \) and the highest budget consumption \( C^U = 1 \). The least rapid release requirement is \( \beta^L = 0.6 \) and the maximum rapid release expectation for this release considered as \( \beta^U = 0.9 \).

Also 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 [4].
By using the concept from section 4; \( x \) and \( y \) are determined. Thus, we get the following equations:

\[
u(C) = 2C - 1 \quad ; \quad u(\beta) = \frac{10}{3} \beta \quad - 2
\]

**III): Crediting the Weights**

In this stage, the weight parameter \( \lambda \) is estimated by comparing the two choices in Figure 2, by lottery approach \([13,14]\). Management has claimed that it is indifferent between these two choices when \( p \) is equal to 0.5, hence \( \lambda_c = 0.5 \). It is easy to calculate \( \lambda_\beta \) based on the sum of weight parameters is equal to one, therefore \( \lambda_\beta \) is also equal to 0.5.

**IV): Maximization of Multi-Attribute Utility Function (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 3.:

\[
\begin{align*}
\max u(\beta, C) &= \lambda_\beta \times u(\beta) - \lambda_c \times u(C) \\
\lambda_\beta + \lambda_c &= 1, \\
\frac{C(t)}{C_s} \leq 1
\end{align*}
\]

The above function is maximized by using Maple Software package and the optimal release time; \( T^* = 15.346 \). Figure 3 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 15 (that is why we consider the optimal time of release to be this). Figure 4 represents the behaviour of the cost function. According to Tandem data failure, real time to 4th release is 19 weeks. Based on optimal result, we can say that software in this release is over tested. It has been released four weeks earlier if the given attributes are to be considered.

![Figure 3. The multi-attribute utility function against time](image)

Further, we have also taken into consideration the major concern of the software development firm’s appropriate time for release of up-graded version of the software with the required level of reliability. In order to overcome 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 [7, 16]. This concept is very important in analysing the behaviour of reliability in testing phase for determining appropriate time for new release.

6. LAPLACE TREND TEST

We have applied the Laplace trend test to the failure data, in order to determine whether the software in our release underwent a reliability growth or not. We divide 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:

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

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 [7,16]. Figure 5 shows the Laplace trend test results for the fourth release of the data set, as reported by tandem computer [29]. We can see that \( u(k) \) values are positive from beginning because of adding some new functionality in the code. However, from period 4\(^{th}\) to 11\(^{th}\) we see some fluctuations but this fluctuation doesn’t affect the reliability much and the trend line has decreasing behaviour from 11th week onwards and after 13th week, this behaviour completely stabilizes which means that reliability increases monotonically.
7. CONCLUSION

A vital decision problem that the management encounters is to determine when to stop testing and release the software system to the user. If the release of the software is unduly delayed, the software developer may suffer in terms of penalties and revenue loss, while a premature release may cost heavily in terms of fixes (removals) to be done after release, which consequently might harm his reputation. On one hand, the software is expected to be tested in such a manner that it costs reasonable on the other hand, rapid release index is also an important factor because it is directly related with quality of the software. It can be easily seen that both of these factors are contradicting with each other. In order to make the judicious decision, the problem of determining the optimal time of software release has been formulated based on the concept of multi-attribute utility theory in terms of cost and rapid release index subject to the system constraints. The use of SRGMs to depict software reliability provides a statistical foundation to establish optimal release time for software testing. In future we can think of using some more attributes to decide on optimal release time of the software, keeping in mind the importance of attributes in deciding the release policy. In the present case the warranty cost can be added in the cost model of our decision model and that way we can further refine the decision model developed in this paper.

Acknowledgement

The research work presented in this paper is supported by grants to the second author from Department of Science and Technology (DST) Grant No SR/S4/MS: 600/09, India.

Authors express their deep sense of gratitude to The Founder President of Amity Universe, Dr. Ashok K. Chauhan for his keen interest in promoting research in the Amity Universe and always been an inspiration for achieving greater heights.

REFERENCES


