A Discrete Time Based Approach for Release Time Analysis of Software

P. K. Kapur¹, Sunil Kumar Khatri² and Anshul Tickoo³

¹Amity International Business School, Amity University Uttar Pradesh, Noida, Uttar Pradesh, India
E-Mail: pkkapur1@gmail.com
²Amity Institute of Information Technology, Amity University Uttar Pradesh, Noida, Uttar Pradesh, India
E-Mail: sunilkkhatri@gmail.com
³Amity School of Engineering and Technology, Amity University Uttar Pradesh, Noida, Uttar Pradesh, India
E-Mail: anshultickoo@hotmail.com

accepted February 21, 2012

Summary
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. An important decision problem that the management encounters is to determine when to stop testing and release the software system to the user. Such a problem is known as "Software Release Time Problem". We propose an optimization problem of determining the optimal time of software release based on goals set by the management in terms of cost, reliability and failure intensity etc. subject to the system constraints. 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 reaching desired level of reliability (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. A continuous time model has limitations in its application on many real life applications having discrete time data. Due to this reason, in this paper a discrete model is proposed using a discrete logistic function and an exact solution is obtained using probability generating function (PGF). A numerical illustration is provided towards the end of the paper.

Key words: Multi-attribute utility theory (MAUT), release time problem, software development life cycle (SDLC).
1. INTRODUCTION

Today Computer systems play a dominant role in every area in modern society. There has been an exponential boom in computer industry. Computers have brought an information revolution. Implicitly, the term refers to the sweeping changes brought about in computing and information technology. With the advent of the computer age, computers as well as the software running on them are playing a vital role in our daily lives. As more and more systems are being automated mankind’s dependence on computers is rapidly increasing. Software that drives a computer system is a product of human work and is very likely to contain faults. But with scientific software development technology, quality control and systematic testing, their number could be minimized.

In a Software Development Life Cycle (SDLC) the testing phase is given a lot of importance. Testing consumes around half the resources. Software testing is an important software quality assurance activity. A successful test should show that a program contains bugs rather than that it works fine. Since software testing consumes 40%-80% of the development costs, how to reduce its cost and improve its quality has always been a big challenge to the software engineering community. Functional or black box testing is a type of testing where test cases that simulate the usage of the software at the user end are run (executed). Any departure from specifications is termed as a failure [5]. An effort is immediately made by another team of personnel to ascertain the cause of the failure and subsequently remove it. The first process is called failure observation process and the second, fault removal process. With each fault removal, reliability of the software increases. Software reliability can be defined as the probability of non-failure of software during a specified time period under given testing environment. Need for a quantitative measure of reliability was felt and consequently Software Reliability Growth Models (SRGM) were developed. An SRGM provides a mathematical relationship between the number of faults removed and the testing time (CPU time or calendar time). It is used as a tool to monitor the progress of testing phase through quantifying various reliability measures of the software system such as reliability growth, remaining number of faults, mean time between failures etc. A number of SRGM have been developed in the literature under particular set of assumptions and testing environment and many of them are based upon the Non-Homogeneous Poisson Process (NHPP) assumption [9, 15,19]. NHPP based software reliability growth models are classified into two groups. The first group of models uses the execution time (i.e. CPU time) or calendar time to describe the software failure and fault removal phenomenon. Such models are called continuous time models. The second types of models are discrete time models, these models use the number of test cases executed as a unit for measuring the testing process. Most of the observed software failure data sets are discrete and as such these models many times provide better fit than their continuous time counterparts. There are only a few studies in the literature on discrete software reliability modeling and most of the research in software reliability modeling has been carried on the continuous time models. An NHPP based SRGM describes the failure or removal phenomenon during testing and operational phase. Using the data collected over a period of time of the ongoing testing, one can estimate the number of faults that can be removed by specified time $t$ and hence the reliability.

The importance of testing phase in Software Development Life Cycle (SDLC) cannot be undermined. But early release of the software is also necessary due to many reasons ranging from marketing considerations to increase in cost. Therefore it is important to know the time to stop the testing. The optimal testing time is a function of several factors; size of software, level of reliability desired, personnel available, market environment, cost associated in its development.
In the present study we have first discussed about fault removal phenomenon using discrete K-G model and then we have focused on two different attributes on the basis of which optimal release time is decided.

The paper is outlined as follows: In Sec 2, we have discussed about a SRGM that takes two stages to remove a fault and discrete K-G model. Sec 3 discusses the cost function, Sec 4 discusses about the estimated results of the model. In Sec 5 we have used MAUT as an evaluation approach to formulate the problem followed by numerical illustration in Sec 6, Sec 7 comprises of the conclusion remarks followed by acknowledgement in Sec 8.

2. SOFTWARE RELIABILITY GROWTH MODEL

Reliability assessment is of undue importance to both the developers and user, it provides a quantitative measure of the number of remaining faults, failure intensity, and a number of decisions related to the development, release and maintenance of the software in the operational phase.

Most of the testing methods aim to uncover the faults lying in the software. When a fault is exposed, the corresponding fault is repaired. This task of failure observation and fault removal gives an indication of improved system reliability. Software reliability assessment during different phases of software development is an attractive approach to the developer as it provides a quantitative measure of what is most important to them "software quality" [5]. Now the question arises how we can measure the observed system reliability. Now comes the role of software reliability modeling, a sub field of SRE. The reliability models known as Software Reliability Growth Models (SRGM) can be used here to estimate the current level of reliability achieved and to predict the time when the desired system reliability can be achieved.

SRGM play an important role due to their ability to predict the fault detection / removal phenomenon during testing. Several classes of SRGM 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 [1]. 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 [5, 9, 15].

Kapur and Garg [5] proposed the delayed S-shaped model whereas Ohba [17] and Yamada [11] also proposed S-shaped models. Several Other researchers have developed S-shaped models under different testing environment and assumptions [20, 27]. The general building block of these SRGMS most of which are Non Homogeneous Poisson Process (NHPP) based is the following differential equation:

\[
\frac{d}{dt} m(t) = b(t)[a(t) - m(t)],
\]

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 (1) can also be suitably modified to capture testing phenomenon more
realistically. The model proposed by Kapur and Garg [5] is one of the most popular model in software reliability area and can fit on many types of failure data of testing phase because of its flexible nature. This model is based upon the following additional assumption: On a failure observation, the fault removal phenomenon also removes proportion of remaining faults, without their causing any failure.

The following system of differential equations describes the same:

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

(2)

where,

\(m(t)\): expected mean number of faults removed at time \(t\),

\(b(t)\): rate of fault detection per remaining faults,

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

\(\beta\): constant parameter in logistic learning-process function.

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

(3)

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

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

(4)

Many Multi-up gradation models have been proposed by Kapur et al[5]. Some regarded the release of new software to be depending on all the precious releases [8] and some considered its dependency on just previous release [6, 7, 22, 23]. Tandem Data [26] comprises of four successive releases. It should be noted that, in the present case we have treated the 4th release as a new version of the software but the defect count 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 [19]. However in practical situation as the testing goes on the experience of the testing team increases with the software under testing and therefore it is expected that fault removal rate per remaining error will follow a logistic learning function.

The reliability function is the basic building block of all the NHPP models existing in the software reliability engineering literature. The models assume diverse testing environments like distinction between failure and removal processes, learning of the testing personal, possibility of imperfect debugging and error generation etc. Knowing \(m(t)\), the reliability function of software in time interval of length \(x\) is given as:

\[R(x | t) = e^{-\left(m(t+x) - m(t)\right)}\]

(5)
2.1 The Discrete K-G Model

A very large number of Continuous Time Models have been developed in the literature to monitor the fault removal process and measure and predict the reliability of the software systems. The K-G model is a main representative of the continuous time models. When this model is in continuous time, there are limitations in its application on many real life applications having discrete time data. A discrete model conserves the properties of the continuous model, and so the parameter estimation would likely be simpler and more accurate.

This model incorporates the learning process of testing team into SRGM. Accordingly, following the general assumptions of a discrete SRGM the testing process is modeled by the following difference equations:

\[
\frac{m(N+1) - m(N)}{\delta} = \frac{b(a - m(N))}{1 + \beta(1 - b\delta)^{N+t}},
\]

where \(\delta\) is a constant time interval and \(m(N)\) represents the expected number of faults in discrete time \(N\).

Solving eqn 6 using initial condition \(m(N = 0) = 0\), we get:

\[
m(N) = \frac{a(1 - (1 - b\delta)^N)}{1 + \beta(1 - b\delta)^N}.
\]

3. MODELING OF COST FUNCTION

The software performance in 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 [18] proposed a cost function for the total cost incurred during testing given as:

\[
C(N) = C_1m(N) + C_2[a - m(N)] + C_3N
\]

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,

\(m(N)\) is the expected number of faults removed in \(N\) test runs,

\(C(N)\) is the total cost in fault removal.
A firm never wants to spend more than its capacity, therefore the first attribute that we consider is:

\[
\text{Min: } C_a = \frac{C(N)}{C_B} \tag{10}
\]

where,

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

4. PARAMETER ESTIMATION

To verify the model we have used Tandem computers failure data set [26]. This data set includes software faults for four separate releases. We have extensively worked on the fourth release of this data. This is because after checking all the 4 releases, it was noted that release 4 was having maximum S-character and the use of discrete K-G 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 1. Figure 1 shows the estimated values of the number of faults removed for the release.

*Table 1. Parameter Estimation*  

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>( a )</td>
<td>0.23</td>
<td></td>
</tr>
<tr>
<td>( b )</td>
<td>3.475</td>
<td></td>
</tr>
<tr>
<td>( \Delta )</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 1. Actual vs Predicted*
From the graph we notice the behaviour of actual faults data for software release and observe that it is S-shaped in nature. This is further justified by the use of K-G function to detect the faults in the software. The model gives very good fit as exhibited by the values of various comparison criteria.

5. USING MULTI-ATTRIBUTE UTILITY THEORY (MAUT) AS AN EVALUATION APPROACH

Multi-attribute utility theory (MAUT) combines a class of psychological measurement models and scaling procedures that can be applied to the evaluation of alternatives with multiple value relevant attributes. For example, MAUT can be used to analyze preferences between cars described by the attributes cost, comfort, prestige, and performance. MAUT may also be applied as a decision aiding technology for decomposing a complex evaluation task into a set of simpler subtasks. For example, the decision maker such as a Software engineer might be asked to assess the utility of each alternative with respect to each attribute and to assign importance weights to each attribute. 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. An appropriate combination rule is used to aggregate utility across attributes.

We can say that, Multi-Attribute Utility Theory (MAUT) is a label for a family of methods. These methods are a 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 of a set of alternatives [24]. 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 [12,16]. It has been used in a broad range of fields including energy, manufacturing and services, public policy and healthcare. The MAUT process can provide a framework through which multiple objectives and uncertainty can be combined to aid managers in making decisions.

5.1 Building the Utility Function

There are 6 steps [24] involved in determining the utility value.
1. Attribute Selection.
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 [3, 13]. Another importance is that it provides feasibility to consider the alternative on the continuous scale [2, 13]. Multi-attribute expected utility theory [3,12], was explicitly designed for decisions under risk. The utility function U obtained with this approach not only preserves the decision maker's riskless preference order, but also may be used in expected utility computations to select among risky alternatives.

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 reliability. A Multi-Attribute Utility Function (MAUF) is defined as:
\[ U(x_1, x_2, \ldots, x_n) = f \left[ u_1(x_1), u_2(x_2), \ldots, u_n(x_n) \right] = \sum_{i=1}^{n} w_i u_i(x_i) \]  \hspace{1cm} (11)

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

### 5.1.1 Attribute Selection

An important 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 reliability. These attributes are two important factors for determination of optimal planning testing time/resource of software. The objective function of cost attribute is given in Sec-3 and this attribute should be minimizing. The objective of software reliability \( R(N) \) is formulated as:

\[ R(h \mid N) = e^{(-m(N+h)-m(N))} \]  \hspace{1cm} (12)

Huang et al., (2005) defined a new measure for software reliability as the ratio of the number of cumulative faults detected at to the number of initial faults in the software system, i.e.,

\[ R_i(N) = \frac{m_i(N)}{a_i} \]  \hspace{1cm} (13)

Both forms are useful but we use second form because of simplicity and direct relation with number of faults removed. Second attribute for 4th release is given as:

\[ \text{Max}: R_4(N) = \frac{m(N)}{\text{initial fault content on 4th release}} \]

It may be noted that reliability function has increasing behaviour and approaches to 1 when \( N \) becomes infinity large.

### 5.1.2 Attribute Bounds Selection

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 upper and lower bounds of an attribute are chosen by the designer. 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 [12]. In the present study, using the concept of Lei et al [13], suppose that the single utility function for cost is to be determined, the lowest and highest values of cost are selected first as $C_i^0$ and $C_i^1$. At these boundary points, we have $u(C_i^0) = 0$ and $u(C_i^1) = 1$.

### 5.1.3 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 the probability $p$ of the most preferred alternative or $1-p$ of 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.

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

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 [12, 13].

The component utility function for attribute $i$ ($u_i$) is assessed by the use of lottery [13, 21]. The three data points used to determine the unknown coefficients are obtained from the equation $u(x) = pu(x^0) + (1-p)u(x^w)$, where $x$ is the certainty value, $x^0$ is the best alternative, and $x^w$ is the worst alternative (Refer Fig 2.). Given that the utility is scaled between 1 and 0, $u(x^0) = 1$, $u(x^w) = 0$, so $u(x) = p$.

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

### 5.1.5 Estimation of Scaling Constants

In this section we have discussed about estimation of weight parameter, $w_i$. The weights reflect the relative importance of moving an attribute from worst to finest level, they are defined on ratio scale. Different approaches for obtaining numerical weight have been proposed, including direct tradeoff methods, direct judgment of swing weight and lottery-base utility assessment [12, 13]. By these methods, different importance can be assigned 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 \( R \) as software development cost and measurement of failure reliability. (Let \((R^H, C^H)\) and \((R^L, C^L)\) denote the finest and worst possible consequence, (see in Figure 2) respectively. There is a certain joint outcome \((R^H, C^L)\) comprised two attribute \( C \) and \( R \) at the best and worst level with probability \( p \) and \((1 - p)\), respectively.) In these situations, the weight for attribute \( R \) equals \( p \), where \( p \) is the indifference probability between them, [13, 25].

![Diagram](image)

**Figure 2. Two Choices for determining scaling constants (Source: Li et al [13])**

### 5.1.6 Development of Multi Attribute Utility Function (MAUF)

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 [12]. In the present work, the additive form of the MAUF is given as:

\[
\text{Max}: U(R, C) = w_R \times u(R) - w_C \times u(C)
\]

\[ w_R + w_C = 1 \]  \hspace{1cm} (14)

where \( w_R \) and \( w_C \) are the weight parameters for attribute \( R \) and \( C \) respectively, \( u(R) \) and \( u(C) \) are the single utility function for each attribute. It may noted that the \( U(R, C) \) function is of Max type and it has been written in terms of \( R \) and \( C \). From manager point of view, \( R \) 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, \( N^* \) will be obtained.

### 6. NUMERICAL ILLUSTRATION

Tandem Data [26] comprises of four successive releases. The proposed decision model has been validated for its Fourth release. The 4\textsuperscript{th} version of software is released after 19 weeks. In this paper our main aim is to find out the optimal time for the release.

The determination of optimal planning testing time is done using the methodology as described in section 5.1.

1) Selecting the Attributes

In the present problem, two attributes as cost and measurement of reliability 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 (eqn 4), Figure 3. The Reliability function, $R(N)$ is given by:

$$\text{Max} : R_a = \frac{m(N)}{a}.$$  

For other attribute i.e. cost we use the cost model as discussed earlier in Section (3):

$$\text{Min} : C_a = \frac{C(N)}{C_B}.$$  

We set parameters $C_1 = 7, C_2 = 15, C_3 = 6$ and $C_B = 1400$ as parameter of cost function. The cost function is then calculated by the value of estimated parameters given in the Table 1.

II): 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:

- under minimization cost strategy, management indicates that at least 50% of budget must be consumed,
- for reliability attribute, management has verified that at least 50% of software faults should be detected and the more the better,
- 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 $C^{\text{low}} = 0.5$ and the highest budget consumption $C^{\text{high}} = 1$. The lowest reliability requirement is $R^{\text{low}} = 0.5$ and the highest reliability for this release considered as $R^{\text{high}} = 1$.

III): 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 [13]. The parameters $y_1$ and $y_2$ are determined. Specifically, we have the following equations:

$$u(C_i) = 2C_i - 1, \quad u(R_i) = 2R_i - 1.$$  

IV): Crediting the weights

Management has claimed that it is indifferent between two choices when $p$ is equal to 0.5, hence $w_{C_i} = 0.5$. It is easy to calculate $w_{R_i}$ based on the sum of weight parameters is equal to one, therefore $w_{R_i}$ is also equal to 0.5.

V): 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 4:

$$\text{Max} u(R_a, C_a) = w_{R_i} \times u(R_a) - w_{C_i} \times u(C_a)$$  

$$w_{R_i} + w_{C_i} = 1,$$

$$\frac{C(N)}{C_B} \leq 1$$  

$$\text{(16)}$$
The above function is maximized by using of Maple package and the optimal time to release is $N^* = 21$. Figure 4 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 21 (that is why we consider the optimal time of release to be this). The behavior of cost function shows in Figure 5.

**Figure 3. Number of faults against time**

**Figure 4. The behaviour of multi-attribute utility function**
In this paper, a mathematical model has been discussed based on Multi Attribute Utility Theory, to determine the optimal time to release the software to the user. Software companies plan successive releases for their product because of two aspects of economics and software engineering. Due to limitation of time and resource, software firms do not attempt to deliver a complete and perfect product in one development cycle. If the release of the software is unduly delayed, the manufacturer (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 the manufacturer’s reputation. On one hand, the software is expected to be tested in such a manner that it costs reasonable on the other hand, reliability is also important because it is also an important aspect of software quality. The difficulty is that both of these factors are contradicting with each other. The management has to make a decision to determine when to stop testing and release the software. In order to make this decision, this problem of determining the optimal time of software release has been formulated. 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.

**Acknowledgement**

The research work presented in this paper is supported by grants to the first 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


