Software Up-gradations and Optimal Release Planning

P K Kapur
Amity International Business School, Amity University, Noida, Uttar Pradesh, India

ABSTRACT
Intense global competition in the dynamic environment has lead to up-gradations of software product in the market. The software developers are trying very hard to project themselves as organizations that provide better value to their users. One major way to increase the market charisma is by offering new functionalities in the software periodically. But these intermittent add-ons in the software lead to an increase in the fault content. Thus, for modelling the reliability growth of software with these up-gradations, we must consider the failures of the upcoming release and the faults that were not debugged in the previous release. Based on this idea, a mathematical modelling framework for multiple releases of software products has been proposed. The model uniquely identifies the faults left in the software when it is in operational phase during the testing of the new code. The model has been validated on real data set. Now, since the proposed structure is dependent only on time, it can be categorized under one dimensional modelling outline. But the need of the hour is to consider other factors (available resources; coverage, etc) simultaneously. Therefore, using a Cobb Douglas production function we have extended our own modelling framework and developed a two dimensional software reliability growth model for multi releases which concurrently takes into consideration testing time and the available resources. Another major concern for the software development firms is to plan the release of the upgraded version. In a Software Development Life Cycle the testing phase is given a lot of importance. But testing cannot be done indefinitely, hence it is pertinent to find the optimal release time during testing phase. Too late an entry is likely to lead to significant loss of opportunity and on the other hand early release of any software product might hinder its growth due to lack of receptiveness of users towards new expertise. Therefore, timing plays a very important role. In software world we term this problem as Release Time Problem. Many release time problems with optimization criteria like cost minimization, reliability maximization and budgetary constraints etc. have been discussed in the literature. We have formulated an optimal release planning problem which minimizes the cost of testing of the release that is to be brought into market under the constraint of removing a desired proportion of faults from the current release. The problem is illustrated using a numerical example, and is solved using Genetic Algorithm. Further, we have also discussed the release time problem based on a new concept of Multi-Attribute Utility Theory that takes into consideration two conflicting attributes simultaneously. This framework has also been illustrated using a numerical example.

Keywords
Software Reliability, Multi up-gradation, Multi Attribute Utility Theory, Software Reliability Growth Model, Cobb Douglas production function, Optimal Release Time.

1. INTRODUCTION
The demand for continuous service in mission- and safety-critical software applications, such as Internet infrastructure, aerospace, telecommunication, military defense and medical applications is expanding .The intense global competition in the dynamic environment has lead to a technological substitution of software product in the market. The software developers are trying very hard to project themselves as organizations that provide better value to its customer. One major way to increase the market presence is by offering new functionalities in the software periodically. Technological breakthroughs are happening rapidly and these new innovations often take form of a new product. The concept of performance of a new technology generation over its life cycle has been explained by using well known s-shaped curve or sigmoid curve. It has been seen that in the initial period of the software more efforts are put increasingly so that overall performance of the technology can be improved till attaining its natural performance limit. In general when software reaches a level when it attains it operational reliability level desired by the firm, a new version is introduced and the software gets upgraded. The term upgrade refers to the replacement of a product with a newer version of the same product. It is most often used in computing and consumer electronics, generally meaning a replacement of hardware, software or firmware with a newer or better version, in order to bring the system up to date or to improve its characteristics. As the software firms are involved in developing complex software system with a sharp eye on the market competition, the quality of their product is always under check. Performance of a software system is dependent on its user's needs and requirements. While a system's performance may remain the same or even improve, the user may come to believe that the system is declining in performance as technology changes. Although technological obsolescence is present in any industry, its speed is more pronounced in the software industry. Therefore, it is critical to have a constant look on the state of a software system which includes the views of its customers. This index allows us to incorporate customer views directly into the process, and to develop an operational framework for the analysis of warranty, maintenance, and upgrade policies .Upgrading a software application is a complex process. The new and the old component may differ in the functionality, interface, and performance. Only selected components of an application are changed while the other parts of the application continue to function. This process leads to an increase in the fault contents. The software testing team is always interested in knowing the bugs present in the software. Therefore they continuously keep on testing the software. Although developers produce upgrades in order to improve a product, there are risks involved, including the possibility that the upgrade will worsen the product. Upgrades of software introduce the risk that the new version (or patch) will contain a bug, causing the program to malfunction in some way or not

to function at all. For example, in October 2005, a glitch in a software upgrade caused trading on the Tokyo Stock Exchange to shut down for most of the day [22,23]. Similar gaffes have occurred: from important government systems to freeware on the internet. Upgrades can also worsen a product subjectively. A user may prefer an older version even if a newer version functions perfectly as designed. The above phenomenon is generally due to software failure. Software failures may be due to errors, ambiguities, oversights or misinterpretation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems. Abundance of software reliability models have been developed in the history of this subject. Goel and Okumoto [19] proposed an SRGM, which describes the fault detection rate, as a non homogeneous Poisson process (NHPP) assuming the hazard rate is proportional to remaining fault number.

The basic assumptions of their model were as follows:-
1. Software systems are subject to failure during execution caused by a fault remaining in the system.
2. Failure rate of the software is equally affected by the faults remaining in the software.
3. The number of faults detected at any time is proportional to the remaining number of faults in the software.
4. On a failure, repair effort starts and the fault is removed with certainty.
5. All faults are mutually independent from failure detection point of view.
6. The proportionality of fault detection/isolation/correction is constant.
7. The fault detection/ correction are modeled by non homogeneous poison process.
8. The number of faults in the beginning of the testing phase is finite.

In the last two decades several Software Reliability models have been developed in the literature showing that the relationship between the testing time and the corresponding number of faults removed. They are either Exponential or S-shaped or a mix of the two. The software includes different types of faults, and each fault requires different strategies and different amounts of testing effort to remove it. Ohba [17] refined the Goel-Okumoto model by assuming that the fault detection/removal rate increases with time and that there are two types of faults in the software. SRGM proposed by Bittanti et al. and Kapur and Garg [5,9] has similar forms as that of Ohba but is developed under different set of assumptions but all are flexible in nature. Bittanti et al. [30] proposed an SRGM exploiting the fault removal (exposure) rate during the initial and final time epochs of testing. Whereas, Kapur and Garg [5,9] describe a fault removal phenomenon, where they assume that during a removal process of a fault some of the additional faults might be removed without these faults causing any failure. These models can describe both exponential and S-shaped growth curves and therefore are termed as flexible models. Later, Kapur et al.[9] proposed an SRGM with three types of fault. The first type is modeled by an Exponential model of Goel and Okumoto [19]. The second type is modeled by Delayed S-shaped model of Yamada et al. [28, 29]. The third type is modeled by a three-stage Erlang model proposed by Kapur et al [5,9]. The total removal phenomenon is again modeled by the superposition of the three SRGMs. They extended their model to cater for more types of faults by incorporating logistic rate during the removal process.

Figure 1 depicts the increase in failure rate due to the addition of new features in the software. Due to the feature upgrades, the complexity of software is likely to be increased as the functionality of software is enhanced [8]. Even fixing bugs may induce more software failures by fetching other defects into software. But if the goal of the firm is to upgrade the software by enhancing its reliability then it is possible to incur a drop in software failure rate that can be done by redesigning or re-implementing some modules using better engineering approaches.

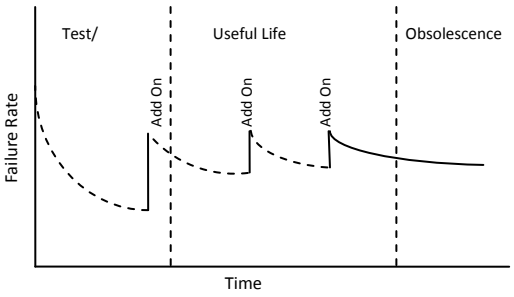

Fig 1: Failure rate curve due to Feature Enhancements for Software Reliability

2. NOTATIONS

m(t) : Number of faults detected during the testing time t

a_i : Constant, representing the initial number of faults lying dormant in the software when the testing starts for ith release; i=1 to 4 .

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

$f(t)$: Probability density function.

$F(t)$: Probability distribution function.

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

b_i : Fault removal per remaining faults; i=1 to 4.

β_i : Constant parameter describing learning in the fault removal rate; i=1 to 4.

3. MODELING SOFTWARE RELIABILITY FOR VARIOUS RELEASES

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

Let $\{N(t), t \geq 0\}$ be a counting process representing the cumulative number of software failures by time t. The N (t) process is shown to be a NHPP with a mean value function m (t).Mean value function represents the number of faults removed my time t.

$$\Pr(N(t)=n)=\frac{(m(t))^n}{n!}\exp((-m(t)),\text{n=1,2....} \quad (1)$$

$$\mathbf{m(t)=\int_{O}^{t}\lambda(x)dx}$$

So,

$$\frac{dm(t)}{dt}=\frac{f(t)}{1-F(t)}\left(a-m(t)\right) \quad (2)$$

Solving the above equation using the initial condition m (0) = 0

$$m(t)=a\,F(t) \quad (3)$$

Release1

The most important phase in the software development life cycle is testing. Testing starts once the code of software is written. Before the release of the software in the market the software testing team tests the software rigorously to make sure that they remove maximum number of bugs in the software .Although it is not possible to remove all the bugs in the software practically. Therefore, when one software version is tested by the testing team, there are chances that they may detect a finite number of bugs in the code developed. These finite numbers of bugs are then removed perfectly and mathematical equation for it is given as under:-

$$m_1(t)=a_1F(t) \qquad 0<t<t_1 \quad (4)$$

Release 2

Due to fierce competition and technological changes the software developer is forced to add new features to the software. New features added to the software leads to complexity and increase in the fault content of the software. While testing the newly formed code, there is always a possibility that the testing team may find some faults which were present in previously developed code. Testing the newly developed code helps the developer to actually improve the software overall as it also removes some faults of previously developed code. In this period, when there are two versions of the software, $a_1\left(1-F_1(t_1)\right)$ the left over fault content of the first version interact with new detection/ correction rate. As a result of these interactions a fraction of faults which were not removed during the testing of the first version of the product gets removed. In addition, faults are generated due to the enhancement of the features, a fraction of these faults are also removed during the testing with new detection proportion i.e. $F_2(t-t_1)$. The change in the fault detection is due to change in time, change in the complexity due to new features, change in testing strategies etc. The two resulting equations are as following:

$$\begin{aligned} &m_1(t)=a_1F_1(t) && 0<t<t_1\\ &m_2(t)=a_2F_2(t-t_1)+a_1\left(1-F_1(t_1)\right)F_2(t-t_1) && t_1<t<t_2 \end{aligned} \quad (5)$$

Release 3

When the new functionalities are added in the software for the second time again new lines of code are developed. This new code is integrated with the existing code and a testing is started again. It is known that bugs present in the software are infinite. Therefore during the testing in this phase a lot of bugs which have been left in primitive stage and first add-ons are removed. This helps in removing more and more bugs from the developed code. In this period, when there are three versions of the software, $a_2\left(1-F_2(t_2)\right)$ the left over fault content of the second version interacts with a changed rate of fault detection/ correction. As a result of these interactions a fraction of faults which were not removed during the testing of the second version of the product gets removed.. As the new version of the software gets introduced, it brings in with it fault content to the software system. A proportion of these faults get removed when the testing team tests the new code and these faults are removed with the detection proportion $F_3(t-t_2)$. The three resulting equations are as following:

$$\begin{aligned} &m_1(t)=a_1F(t) && 0<t<t_1\\ &m_2(t)=a_2F_2(t-t_1)+a_1\left(1-F_1(t_1)\right)F_2(t-t_1) && t_1<t<t_2\\ &m_3(t)=a_3F_3(t-t_2)+a_2\left(1-F_2(t_2)\right)F_3(t-t_2) && t_2<t<t_3 \end{aligned} \quad (6)$$

In the above situation, the newly developed code for third release, the code developed for second release and the original code of the software are tested and the cumulative numbers of faults are removed with a failure rate of $F_3(t-t_2)$. In the third stage, we can identify that a finite number of faults are left over from first and second release which are now getting removed with a different testing effort and under different testing conditions governed by the failure distribution.

Release 4

The process of adding new functionalities is an on going process. These add-ons keep on happening till the product is there in the market. This phenomenon 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.

$$\begin{aligned} &m_1(t)=a_1F_1(t) && 0<t<t_1\\ &m_2(t)=a_2F_2(t-t_1)+a_1\left(1-F_1(t_1)\right)F_2(t-t_1) && t_1<t<t_2\\ &m_3(t)=a_3F_3(t-t_2)+a_2\left(1-F_2(t_2)\right)F_3(t-t_2) && t_2<t<t_3\\ &m_4(t)=a_4F_4(t-t_3)+a_3\left(1-F_3(t_3)\right)F_4(t-t_2) && t_3<t<t_4 \end{aligned} \quad (7)$$

The above equations are explained as follows: when the testing of the software began initially, the fault content is a_1 which gets reduced by the proportion $F_1(t_1)$. A proportion of faults $a_1F_1(t_1)$ get removed till time t_1. At this time the developers start to test the new version of the software. The new version of the software leads to a generation of faults in the software system due to the complexity added by new functionalities. When testing is in process, a_2 amount of faults are added in the software in the interval $[t_1,t_2]$. The second equation depicts that a proportion of these faults are removed while testing and also a part of leftover faults of the first version software product are removed with a rate which is different from the initial testing rate. Similarly when a new version of a software in introduced in the market for the third

time it further adds to a_3 amount of faults in the software. A fraction of these faults are removed during the testing. In addition to these faults a part of fault which was left over during the testing of the second version product .i.e. $a_2\left(1-F_2\left(t_2\right)\right)$. When the software is released for the forth time in the market a percentage of faults a_4 get removed with detection proportion $F_4\left(t-t_3\right)$. Similarly, the fault contents which were left over in the software system during the time of first, second and third releases gets removed with the proportion $F_4\left(t-t_3\right)$ and a part of their fault content gets removed during testing in the time interval[t3,t4].Here, we assumed that $F\left(t\right)$ follows a logistic distribution. This helps us in developing a flexible software reliability growth model, which is s-shaped in nature. The s-shapedness of model helps us to capture the non uniform nature of testing in the above developed model.

$$F_i\left(t\right)=\frac{\left(1-\exp\left(-b_i t\right)\right)}{1+\beta_i \exp\left(-b_i t\right)} \qquad (8)$$

4. Estimation of parameters, Model validation and Comparison Criteria

Parameter estimation is of prime significance in software reliability prediction. In our study we have used the Statistical Package for Social Sciences (SPSS) [5, 9]. The present study is based on the data available on Tandem Computers. Once the analytical solution for mean number of faults detected/removed by time t given by $m(t)$ that is mostly described by the non-linear functions is known for a given model, the parameters in the solution are required to be determined. Parameter estimation is achieved by extensively used estimation techniques for non-linear models method of Non-linear Least Square (NLLS). Figures (2,3,4,5): Release 1,2,3,4 respectively.

4.1 Figures and Tables

TABLE 1: parameter estimates

Release	1	2	3	4
a_i	110.82	124.37	62.5925	44.983
b_i	0.1720	0.2535	0.5684	0.2669
β_i	1.2046	3.7784	16.266	2.1116
σ	0.00002	0.001	0.001	0.3537

TABLE 2: comparison criteria

	Release 1	**Release 2**	**Release 3**	**Release 4**
R^2	.982	.995	.996	.995
Bias	.07041	.03703	.000541	0.01289
MSE	17.230	9.6021	3.4182	4.305
Variation	3.5645	2.8690	1.7207	1.9649

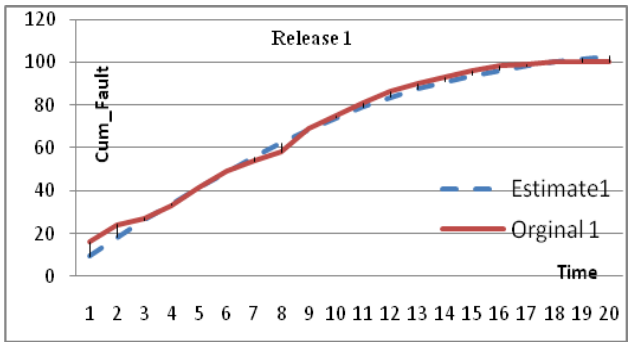

Fig-2

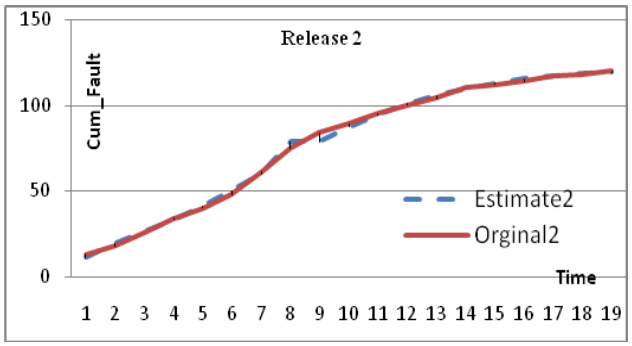

Fig-3

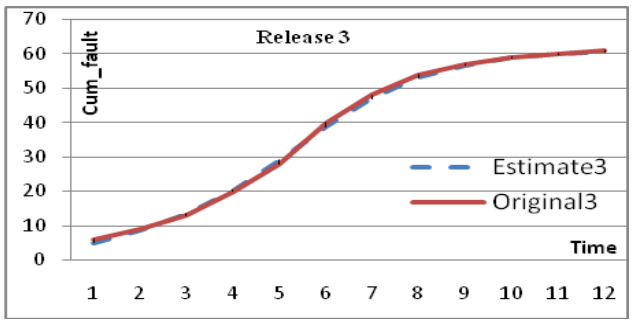

Fig-4

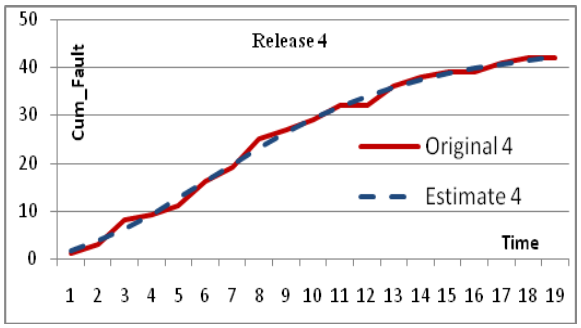

Fig-5

Fig: 2,3,4,5 respectively showing Goodness of Fit for Four Releases

5. TWO DIMENSIONAL MODELING FRAMEWORK

The heading of a section should be in Times New Roman 12- The two dimensional SRGM proposed in our work incorporating testing time and resource usage is based on Non Homogeneous Poisson Process (NHPP) (described in the previous section). However, in one dimensional analysis the object variable is dependent on one basic variable although the object takes on many different roles based upon its dependence on various other factors. Two dimensional models are used to capture the joint effect of testing time and testing resources on the number of faults removed in the software. Such two dimension models are also based on NHPP. In these models we define a two-dimensional stochastic process representing the cumulative number of software failures by time s and with the usage of resources u by $\{N(s,u), s \geq 0, u \geq 0\}$. The mean value function for a two-dimensional NHPP is formulated as:-.

$$\Pr(N(s,u)=n)=\frac{(m(s,u))^n}{n!}\exp(-m(s,u))$$, n=0, 1, 2…

In recent years, Ishii and Dohi proposed a two dimensional software reliability growth model and their application[31]. They investigated the dependence of test-execution time as a testing effort on the software reliability assessment, and validate quantitatively the software reliability models with two-time scales. Inoue and Yamada also proposed two dimensional software reliability growth models[32]. However their modeling framework was not a direct representative of using mean value functions to represent of fault removal process. They discussed software reliability assessment method by using two dimensional Weibull-type SRGM. Moreover, their modeling framework assumes software development as a single release process. Recently, Kapur et. al. proposed a two dimensional modeling framework which was applied in determining optimal allocation of testing time and resources simultaneously to a modular software system. In this section we develop a two-dimensional multi release model which incorporates the combined effect of testing time and resources in each release to remove the faults lying dormant in the software[33,34]. The model developed is based on the Cobb Douglas production function [34]. The functional form of production functions is extensively used to characterize the relationship of an output to inputs. It was proposed by Knut Wicksell (1851–1926), and tested against statistical evidence by Charles Cobb and Paul Douglas in 1900–1928. Cobb-Douglas function presents a simplified outlook of the economy in which production output is obtained by the amount of labor occupied and the amount of capital invested. While there are many factors influencing economic performance, their model demonstrated remarkable accuracy. The mathematical form of the production function is specified as:

$$Y=AL^{v}K^{1-v}$$

where: Y = total production (the monetary value of all goods produced in a year)

L = labor input

K = capital input

A = total factor productivity

v is elasticity of labor. This value is constant and is determined by available technology.

5.1. Notations

A	Initial number of faults.
B	Fault detection rate per remaining simple fault.
λ_i	Fraction of the fault for i^{th} release
a_i	Initial fault content for i^{th} release
S	Testing time.
U	Resources.
A	Resource Elasticity to Testing Time
m (s, u)	Cumulative number of faults removed by time s and with the usage of resources u.

5.2 Two Dimensional Software Reliability Growth Model for Single Release

The two dimensional flexible SRGM presented in this section was proposed by Kapur et al [34]. It is based on the following assumptions-

1. Failure /fault removal phenomenon is modeled by NHPP.
2. Software is subject to failures during execution caused by faults remaining in the software.
3. Failure rate is equally affected by all the faults remaining in the software.
4. Fault detection rate is non-decreasing time and resource-dependent function.
5. On a failure, the fault causing that failure is immediately removed and no new faults are introduced.
6. To cater the combined effect of testing time and resources we use Cobb-Douglas production function of the following form:

$$\tau \cong s^{\alpha}u^{1-\alpha} \qquad 0 \leq \alpha \leq 1 \tag{9}$$

s : testing time

u : testing resources (effort)

θ : Effect of testing time

Under the above assumptions the differential equation representing the rate of change of cumulative number of faults detected w.r.t. to the combined effect of time and resources is given as:

$$m'(\tau)=\frac{f(\tau)}{1-F(\tau)}\left(a-m(\tau)\right) \tag{10}$$

Integrating above equation with initial condition m(τ =0)=0, and using equation (10) we get

$$m(\tau)=a.F(\tau) \tag{11}$$

The above model can also be written in the form

$$m(s,u)=a.F(s,u) \tag{12}$$

Equation (12) is the generalized two-dimensional SRGM model (SRGMs). Substituting different types of distribution functions, we can obtain different mean value functions corresponding to them. The cumulative number of faults removed $m(\tau)$ is dependent on τ . τ is a two-dimensional

variable, with testing time *s* and testing effort *u* as its dimensions. The two-dimensional models are useful as they can show the effect of two aspects of a variable on which the result is dependent.

5.3 Modeling Two Dimensional Multi Release SRGM

In modeling the proposed model apart from the assumptions, we assume that while modeling the mean value function for the next release include the faults of the current release and the remaining faults of just previous release are considered.

Release 1

In multi release model first release is the first step for entering into the market. Hence company will have to pay attention on it because generally first impression counts for the last impression. Therefore to bang into the market, the firm will have to test the software rigorously with an attempt to remove maximum number of faults lying dormant in the software. But, because of time and resource constraints it is not practically possible for the testing team to remove all the errors and thus the initial release of the software is made, with some of the fault content remaining in it. Therefore, when one software version is tested by the testing team, there are chances that they may detect a finite number of bugs in the code developed. These finite numbers of bugs are then removed perfectly and mathematical equation for it is given as under:-

$$m_1(\tau) = a_1 F_1(\tau) \qquad 0 \le \tau \le \tau_1 \tag{13}$$

Release 2

After first release, in order to remain in the market company adds some new functionality into the software. New features added to the software leads to complexity and increase in the fault content of the software. At the same time the firm can't even neglect the errors that are reported in operational phase of the first release.

Therefore, the mathematical equation representing the cumulative number of faults removed in second release is given by:

$$m_2(\tau) = a_2 F_2(\tau - \tau_1) + a_1\left(1 - F_1(\tau_1)\right)\}F_2(\tau - \tau_1); \tau_1 \le \tau < \tau_2 \tag{14}$$

Similar to the arguments given in second release along with taking into consideration the fact that the next release will contain the remaining faults of just released version, we can have the mathematical equations for third, fourth and so on to i^{th} releases.

We follow Exponential distribution for removal process of the faults respectively i.e. $F_i(t)$ is the exponential probability distribution function related to ith release.

5.4 Data Description

We have used the Tandem Computer Company for four releases of software [6, 26]. The data set presents the failure data from four major releases of software product at Tandem computers. The first release consists of 100 faults count observed during 20 weeks with cumulative resources of 10000. In the second release 120 cumulative defects were found collected during 19 weeks with resource usage of 10272. Third release was tested for 12 weeks and 5053 cumulative resources were used and 61 bugs were observed. Last, fourth release was testes for 19 weeks which reported 42 errors and 11305 units resources were used.

5.5 Criteria for Model Comparison and Parameter Estimation

For estimating the parameter values of each release is taken as time of testing (in weeks) and u corresponds to resource usage. From estimation result of first release it was observed that total 9 faults were not removed during testing in which 7 faults simple faults and therefore they are also constituted the part of total faults for the second release. Similarly, the testing team was unable to remove 13 faults in the testing phase of second release, thereby formed the part of third release. In third release only 2 faults was not rectified. The parameter values of the proposed model obtained for the four releases are shown in table 3.

Table 3 Parameter Estimates for Four Releases [i=1, 2, 3, 4]

Parameters	a_i	b	λ_i	α
Release-1	124.57	0.004	0.671	0.460
Release-2	128.59	0.035	0.332	0.824
Release-3	54.15	0.014	0.682	0.541
Release-4	48.97	0.002	0.590	0.404

Table 4 Comparison Criteria for Four Releases

Criteria	R^2	Bias	Variation	MSE
Release-1	.990	0.403	2.81	7.71
Release-2	.995	0.214	2.159	7.065
Release-3	.995	0.050	1.490	1.909
Release-4	.992	0.075	1.106	1.163

The curve of the estimated values of the number of faults removed in different releases using the proposed modeling framework for the four releases is shown graphically in figures 6 to 9, respectively.

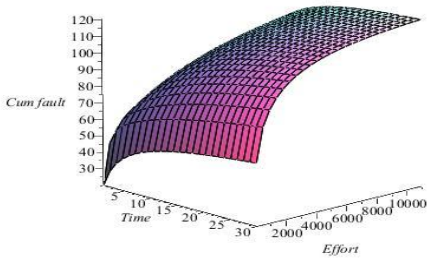

Fig 6: Goodness of Fit Curve for Release 1

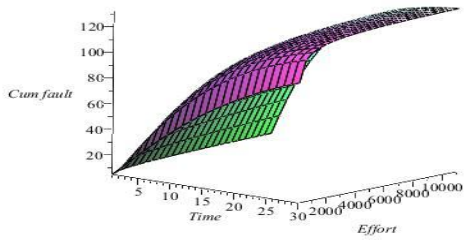

Fig 7: Goodness of Fit Curve for Release 2

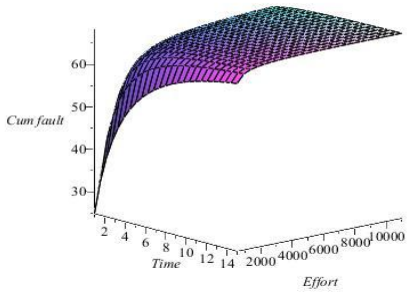

Fig 8: Goodness of Fit Curve for Release 3

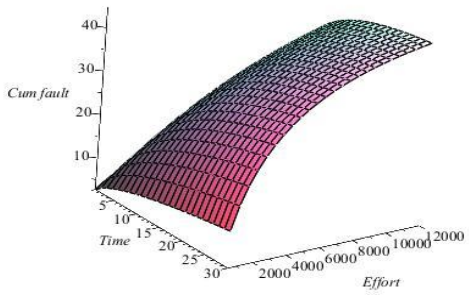

Fig 9: Goodness of Fit Curve for Release 4

6. RELEASE PLANNING PROBLEM

The heading of subsections should be in Times New Roman the release of the up-graded version. Owing to the prevailing paradox between software user's requirements and time and resources limitation for the developers; an imperative decision problem, which arises is to determine when to stop testing of the current release and come up with new version of the software system for the users. Such problems are called Optimal Release Planning Problems. Software users crave for faster deliveries; cheaper software as well as quality product whereas software developers desire to minimize their development cost, maximize the profit margins and meet the competitive requirements. Moreover, it is also a matter of fact that if the new release of the software is overly delayed, the manufacturer (software developer) may undergo thrashing by means of penalties and revenue loss, while a premature release of new version may cost heavily in terms of fixes (removals) to be done of that release in next release and this might also consequently harm manufacturer's reputation. Thus, a tradeoff between conflicting objectives is required.

After the study of existing literature, it has been observed that a lot of research has been done for the release of single version software systems. Okumoto and Goel [18] formulated optimum release time policies using the exponential SRGM. Yamada and Osaki [35] studied optimal release policies based on software cost and software reliability simultaneously for exponential, modified exponential and an S-shaped SRGM. The objective was to minimize the total software development cost subject to reliability less than a predefined reliability level or maximize reliability subject to cost not exceeding a predefined budget. In 1991 Kapur and Garg [11] formulated release policies incorporating the effect of testing resource expenditure for an exponential SRGM under the added assumption that testing resource curves are described by either exponential, Rayleigh or Weibull curve. Huang and Lyu[36] proposed an SRGM with generalized testing effort function and studied optimal release policies based on cost and reliability considering testing effort and efficiency. In 2007 Kapur et al. [37] proposed an SRGM with two types of imperfect debugging and determined the optimal release time of the software.

The release time problems discussed above were studied under the assumption that a software comes in single release; i.e. these researchers do not take into consideration the impact of coming up with multi releases of a software in release planning decisions. But when different versions of the software are to be released, then the firm cannot just plan the release of current version on the basis of testing progress of new code only. It has to consider the log reports of just previous release too. So, here we have formulated an optimal release planning problem which minimizes the cost of testing of the release that is to be brought into market under the constraint of removing a desired proportion of faults (which cannot be 1; as testing cannot be continued indefinitely) to be removed from the release. Another attracting feature of the formulated problem is that it not only considers time as an essential criterion for releasing the new version but also looks simultaneously for resources that govern the pace of testing.

The optimal release planning problem is complex non linear optimization problems and is solved using genetic algorithm (GA)[38]. Furthermore, we have also discussed the release time problem based on a new concept of Multi-Attribute Utility Theory [4,13,14] that takes into consideration two conflicting attributes simultaneously [16,24,25]. This framework has also been illustrated using a numerical example.

6.1 Release Time Problem (Solution Method-GA)

Problem Formulation

In planning the release decisions for software that is to be brought into the market with new versions; the firm has to take into consideration two things:

(i) Testing data of the new code.

(ii) Log reports of just previous release, i.e. bugs reported by the users in the operational phase of the version that had been there in the market.

Where in single release software systems, only (i) prevails, and if (ii) is not taken into consideration for the release planning of multi release software systems; then the notion of coming up with various versions gets lost. This is because the decision of releasing the software lays its foundation on the total faults that are removed from the software. In multi release software the bugs of just previous release also get added to the total faults of the release that is under testing phase. Therefore, to formulate an optimal release planning (ORP) for multi release software we require multi release software reliability growth model. Using the modelling framework as done in earlier Section, we will discuss the following release planning problem.

In the present problem, we consider minimizing the testing cost of the release that is under testing phase with a constraint of removing a desired proportion of faults.

Cost Function

Suppose 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 nth release and cost of failure and removal of faults after the delivery of the nth release and unit cost of testing during the testing phase of nth release. All these costs lead to the following form of the cost functions:

$$C_n(\tau) = C_{n1} m_n(\tau) + C_{n2}\left(\left(a_n + a_{n-1}\left(1 - F_{n-1}(\tau_{n-1})\right)\right) - m_n(\tau)\right) + C_{n3}\tau;$$

Using values of m(τ) we get cost function as:

$$C_n(s^\alpha u^{1-\alpha}) = C_{n1}\left[\frac{a_n\left(1-\exp\left(-b_n s^{\alpha_n} u^{1-\alpha_n}\right)\right)}{1+\beta_n \exp\left(-b_n s^{\alpha_n} u^{1-\alpha_n}\right)}\right] + C_{n2}\left(\left(\left(a_n + a_{n-1}\left(1 - \frac{\left(1-\exp\left(-b_{n-1} s^{\alpha_{n-1}} u^{1-\alpha_{n-1}}\right)\right)}{1+\beta_{n-1}\exp\left(-b_{n-1} s^{\alpha_{n-1}} u^{1-\alpha_{n-1}}\right)}\right)\right)\right) - \left[\frac{a_n\left(1-\exp\left(-b_n s^{\alpha_n} u^{1-\alpha_n}\right)\right)}{1+\beta_n \exp\left(-b_n s^{\alpha_n} u^{1-\alpha_n}\right)}\right]\right) + C_{n3} s^\alpha u^{1-\alpha};$$

where 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.
C_n is the total cost of testing of nth release.

Also, $\tau > \tau_{n-1}$

The total cost is to be minimized. Thus, we have that cost function is a convex function, So, when this cost function will be plotted with respect to time and resources frame. Hence release time problem now can be stated as:

Minimize

$$C_n(s^\alpha u^{1-\alpha}) = C_{n1}\left[\frac{a_n\left(1-\exp\left(-b_n s^{\alpha_n} u^{1-\alpha_n}\right)\right)}{1+\beta_n \exp\left(-b_n s^{\alpha_n} u^{1-\alpha_n}\right)}\right] + C_{n2}\left(\left(\left(a_n + a_{n-1}\left(1 - \frac{\left(1-\exp\left(-b_{n-1} s^{\alpha_{n-1}} u^{1-\alpha_{n-1}}\right)\right)}{1+\beta_{n-1}\exp\left(-b_{n-1} s^{\alpha_{n-1}} u^{1-\alpha_{n-1}}\right)}\right)\right)\right) - \left[\frac{a_n\left(1-\exp\left(-b_n s^{\alpha_n} u^{1-\alpha_n}\right)\right)}{1+\beta_n \exp\left(-b_n s^{\alpha_n} u^{1-\alpha_n}\right)}\right]\right) + C_{n3} s^\alpha u^{1-\alpha};$$

subject to

$$\frac{a_n\left(1-\exp\left(-b_n s^{\alpha_n} u^{1-\alpha_n}\right)\right)}{1+\beta_n \exp\left(-b_n s^{\alpha_n} u^{1-\alpha_n}\right)} (= m_n(s,u)) \geq \rho\left[a_n + a_{n-1}\left(1 - \frac{\left(1-\exp\left(-b_{n-1} s^{\alpha_{n-1}} u^{1-\alpha_{n-1}}\right)\right)}{1+\beta_{n-1}\exp\left(-b_{n-1} s^{\alpha_{n-1}} u^{1-\alpha_{n-1}}\right)}\right)\right]$$

(ORP Problem)

where ρ is desired proportion of faults to be removed from n[th] release.

The above problem is non linear in nature which is difficult to solve through traditional search and optimization methods. This requires use of some new optimization techniques based on natural evolution and natural genetics. Therefore to handle such difficulties we have solved it using Genetic Algorithm. The efficiency of GA lies in its ability of working with population of solutions and not an individual point. In addition GA breaks the limitation of differentiability.

Numerical Example

As an illustration, here we choose the same data set of four releases taken in earlier section. In this data set first, second and third release have already been into market. The problem formulated in section 6 determines when to stop testing the fourth release of the software such that the cost of testing is minimized.

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 1. With these parameter values we solved the following problem using genetic algorithm method given in section 6. Further we assume $C_1 = 10$, $C_2 = 15$, $C_3 = 5$ and it is desired that at least 0.95 proportion of faults should be removed from 4[th] release. The problem is solved using Matlab software under VC++ (6.0) compiler.

Minimize

$$C_4(\tau) = C_{4_1} m_4(\tau) + C_{4_2}\left(\left(a_4 + a_3\left(1 - F_3(\tau_3)\right)\right) - m_4(\tau)\right) + C_{4_3}\tau$$

subject to

$$m_4(\tau) \geq 0.95\left[a_4 + a_3\left(\left(1 - F_3(\tau_3)\right)\right)\right]$$

where $\tau \cong s^\alpha u^{1-\alpha} \qquad 0 \leq \alpha \leq 1$

$\tau > \tau_3$

The parameters used in GA evaluation for both the problems are given in Table 5. The crossover method taken is simulated binary crossover (SBX), and selection criterion is tournament selection without replacement.

Table 5: Parameters of GA

Parameter	Population Size	Number of Generations	Crossover Probability	Mutation Probability
Value	100	25	0.9	0.1

Upon solving the problem the optimal time for stop testing the fourth release came out be 72 week (which is 20 weeks after third release) with an optimal resource consumption of 39100.33 units. The minimum value of cost came out to be 1663.54.

6.2 Release Time Problem (Solution Method-MAUT)

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 [9,19]. 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 [1,9,29]. Release time problems have also been formulated for minimizing cost with reliability requirement or maximizing reliability subject to budgetary constraint [9,11]. Bi-criterion release policy [9,10] 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.

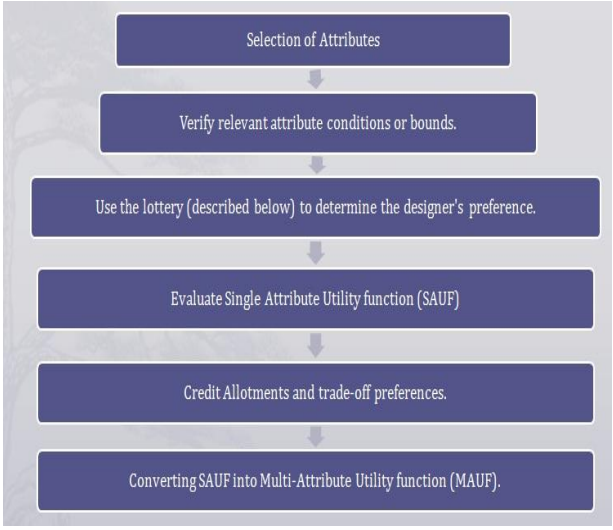

Fig 10. Steps for using MAUT as an Evaluation Approach

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].

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

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

$$where, \quad \sum_{i=1}^{n} w_i = 1 \tag{15}$$

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.

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 [13]; the objective of failure intensity λ(t) is formulated as

$$\max f = \frac{\lambda(t)}{\lambda_{\max}} \tag{16}$$

where, f_i 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 [18] proposed a cost function for the total cost incurred during testing given as:

$$C(T) = C_1 m(T) + C_2\left[a - m(T)\right] + C_3 T \tag{17}$$

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:

$$\text{Min: } C_4 = \frac{C(T)}{C_B} \quad (18)$$

where,

C_B is the total budget allocated to the firm.

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 [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_4^0 *and* C_4^1. At these boundary points, we have . $u(C_4^0) = 0$ *and* $u(C_4^1) = 1$.

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.

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 $u(x) = y_1 + y_2.e^{r.x}$, 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 $u(x) = pu(x^o) + (1 - p)u(x^*)$, where *x* is the certainty value, x^o is the best alternative, and x^* is the worst alternative (Refer Fig 2.). Given that the utility is scaled between 1 and 0,

$u(x^f) = 1$,

$u(x^w) = 0$, 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.

Credit Allotment to Weights

In this section we have discussed about estimation of weight parameter, w_i. 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 [12,13]. 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.2) 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 [13,25].

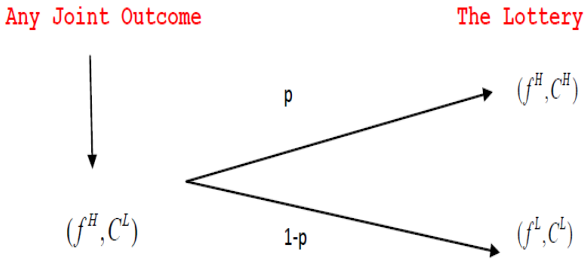

Fig 11: Two Choices for determining scaling constants (Source: Li et al [13])

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

$$Max:\ U(f,C_4) = w_f \times u(f) - w_C \times u(C_4)$$
$$w_f + w_{C_4} = 1 \qquad (19)$$

where w_f and w_{C_4} are the weight parameters for attribute f and C respectively. $u(f)$ and $u(C_4)$ are the single utility function for each attribute. It may be noted that the $U(f,C_4)$ function is of Max type and it has been written in terms of f and C_4. 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.

Numerical Illustration

Tandem Data [26] 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 this paper 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 is used. The determination of optimal planning testing time is done using the methodology as described in previous section.

We set parameters $C_1 = 15, C_2 = 18, C_3 = 3$ and $C_B = 2500$ as parameters of cost function. The bound are selected to the methodology discussed earlier. In particular, the lowest budget consumption requirement is $C_4^W = 0.5$ and the highest budget consumption $C_4^B = 1$. The lowest failure intensity requirement is $f_4^W = 0.1$ and the highest reliability for this release considered as $f_4^B = 0.6$.

By using the concept from earlier section parameters y_1 *and* y_2 are determined. Specifically, we have the following equations:

$$u(C_4) = 2C_4 - 1\ ;\ u(f) = 2f - .2$$

Here, based on the single utility functions and the weight parameters which have been determined in previous steps, the MAUF is evaluated

$$Max\, u(f,C_4) = w_f \times u(f) - w_{C_4} \times u(C_4)$$
$$w_f + w_{C_4} = 1,$$
$$\frac{C(T)}{C_B} \le 1 \qquad (20)$$

The above function is maximized by using Maple package and the optimal time to release comes out to be $T^* = 16.542$.

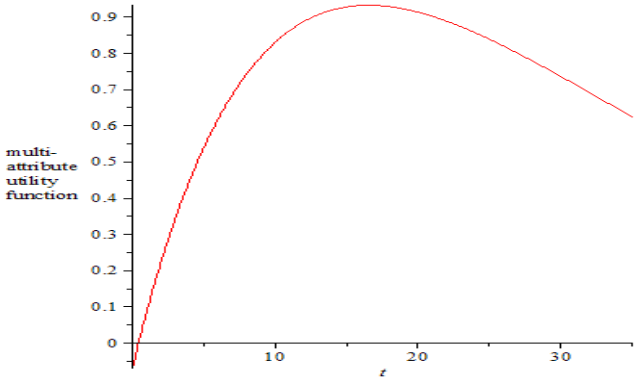

Fig 12: The multi-attribute utility function against time

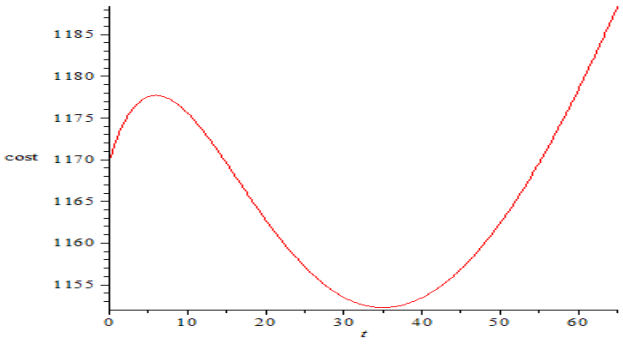

Fig 13: The behaviour of cost function

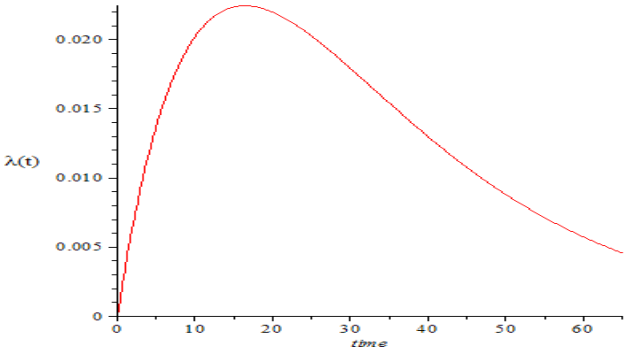

Fig 14: Behaviour of failure intensity function

Figure 12 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 13 represents the behaviour of the cost function. According to Tandem data failure, real time to 3th release is 12 weeks. Figure 14 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.

7. CONCLUSION

In software industry, up graded versions are made in the software at a very brisk speed. The life of software is very short in the environment of perfect competition market; therefore developer has to come up with successive releases to survive. But a matter of fact is that up-gradation of software application is a complex process. Upgrades of software introduce the risk that the new version will include a bug, causing the program to fail. To capture the effect of faults generated in the software, we have developed multi-release software reliability modelling framework. The model uniquely takes into account the faults of that release which is under the phase of testing (i.e. the release which is to be brought into market) and the faults left in the previous release (i.e. the release which is in operational phase). Further, the model is extended under the assumption that time and resources simultaneously are essential for modeling an accurate software reliability growth model. With this development structure, a relatively unexplored area of software reliability is investigated in this work. The proposed multi release two dimensional model is estimated on the real data set of four releases. Then the release planning problem is discussed in context of multiple releases. The problem has been solved using Genetic algorithm which minimizes the expected software cost subject to removing a minimum desired proportion of faults from the new version that is to be brought into the market. The formulated release planning problem helps in determining both optimal release time and optimal resource consumption simultaneously. At last, another methodology of MAUT has been applied to the release time problem in order to calculate whether the testing has been done appropriately or some more testing time would have required if the software developer had the task of minimizing cost and maximising failure intensity. A numerical illustration is also given for the developed optimal release planning problem.

Taking into consideration the environment of imperfect debugging and exploration of the possibility of including randomness in the fault detection rate forms the scope of future research.

8. REFERENCES

[1] Bardhan A K 'Modelling in Software Reliability and its interdisciplinary nature',Ph.D. Thesis, University of Delhi, Department of Operational Research, 2002.

[2] Ferreira R.J.P.., Almeida A.T., Cavalcante C A V., 'A multi-criteria decision model to determine inspection intervals of condition monitoring based on delay time analysis', Reliability Engineering and System Safety, 94, 905–912, 2009.

[3] Fishburn C P, 'Utility Theory for Decision Making', New York: Wiley, 1970.

[4] Kanoun K, Bastos M., Moreira J., 'A method for software reliability analysis and prediction application to the TROPICO-R switching system'. IEEE Trans. Software. Eng. 17 (4), 334–344. 1991.

[5] Kapur P K,, Pham H., Gupta A., Jha P C, 'Software Reliability Assessment with OR Applications', UK: Springer, 2011.

[6] Kapur P K, Yadavalli V S S, Aggarwal A G., Garmabaki A H S, 'Development of a multi-release SRGM incorporating the effect of bugs reported from operational phase', Communicated., 2012.

[7] Kapur P K, Anand A., Singh O., 'Modeling Successive Software Up-gradations with Faults of different Severity.' Proceedings of the 5th National Conference; INDIACom 2011, Eds. Prof M.N.Hoda ,Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi. pp 351-356, 2011.

[8] Kapur P K, Tandon A., Kaur G, 'Multi Up- gradation Software reliability Model' 2nd international conference on reliability, safety&hazard (ICRESH), pp. 468-474, 2010.

[9] Kapur P K, Garg R B , Kumar S, 'Contributions to hardware and software reliability' Singapore: World Scientific Publishing Co. Ltd, 1999.

[10] Kapur P K, Agarwala S, Garg R B, 'Bicriterion release policy for exponential software reliability growth model" Recherche Operationnelle – Operations Research, 28: 165-180, 1994.

[11] Kapur P K, Garg R B, 'Optimal release policies for software systems with testing effort' Int. Journal System Science, ,22(9), 1563-1571, 1990.

[12] Keeney R L, and Raiffa H, 'Decisions with Multiple Objectives: Preferences and Value Tradeoffs', New York, Wiley, 1976.

[13] Li, X. , Li Y. F., Xie M. and Ng S H, 'Reliability analysis and optimal version-updating for open source software', Information and Software Technology, Vol. 53, pp. 929–936, 2011.

[14] Luo,Y. Bergander T., 'Software reliability growth modeling using weighed laplace test statistic', Annual international Conference (COMPSAC 2007), 2:305–312, 2007.

[15] Musa J D., Iannino A., Okumoto K, 'Software reliability: Measurement, Prediction, Applications', New York: Mc Graw Hill, 1987.

[16] Neumann J V,and Morgenstern O., 'Theory of Games and Economic Behaviour (2nd ed.)', Princeton, NJ, Princeton University Press, 1947.

[17] Ohba, M 'Software reliability analysis models', IBM Journal of Research and Development,28: 428-443, 1984.

[18] Okumoto K. and Goel A L, 'Optimal release time for computer software' IEEE Transactions On Software Engineering.; SE-9 (3): 323-327, 1983.

[19] Okumoto K. and Goel A.L., 'Time dependent error detection rate model for software reliability and other

performance measures' IEEE Transactions on Reliability, R-28(3): 206-211, 1979.

[20] Pham H, 'Software Reliability', Singapore, Springer-Verlag Pte. Ltd, 2006.

[21] Seung C. and Zhang C.,, 'Developing socioeconomic indicators for fisheries off Alaska: A multi-attribute utility function approach', Fisheries Research, Vol. 112 No 3, pp. 117-126, 2011.

[22] Singh O., , Kapur P.K. , Anand A.,, 'A Stochastic Formulation of Successive Software Releases with faults Severity' proceedings of The IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), 6-9 December, Singapore, pp136-140, 2011.

[23] Singh O., Kapur P.K, Anand A., Singh J., 'Stochastic Differential Equation Based Modeling for Multiple Generations of Software and Optimal Release Planning', proceedings of 5th International Conference on Quality, Reliability and Infocom Technology (ICQRIT), Trends And Future 8Directions, Kathmandu, Nepal, SN-19, pc-19, 2011.

[24] Thurston L. D., "Multi-attribute Utility Analysis of Conflicting Preferences." Decision Making in Engineering Design. Ed. Kemper E. Lewis, et al. New York, New York: ASME Press, 125-133, 2006.

[25] Winterfeldt D.. And Edwards W. 'Decision Analysis and Behavioral Research", Cambridge, UK, Cambridge University Press, 1986.

[26] Wood. 'Predicting software reliability', IEEE Computer, Vol. 9, pp. 69-77, 1996.

[27] Xie M., 'Software Reliability Modeling', World Scientific Publishing, 1991.

[28] Yamada S.,. Ohba M, M. and Osaki S 'S-shaped software reliability growth models and their applications', IEEE Trans. on Reliability, Vol. 33 No. 4, pp. 289–292, 1984.

[29] Yamada S.,, Narihisa H, Osaki S..'Optimum release policies for a software system with a scheduled software delivery time', International Journal of Systems Science, Vol. 15 No. 8, pp. 905–914, 1984.

[30] Bittanti S., Bolzern P , Pedrotti E., Scattolini R . "A Flexible Modelling Approach For Software Reliability Growth" Software Reliability Modelling and Identification (Ed.) G. Goos and J. Harmanis, Springer Verlag, Berlin, pp. 101-140, 1988

[31] Ishii T. and Dohi, T. "Two-dimensional software reliability models and their application," in 12th Pacific Rim Intern. Symp. Dependa. Comput., 2006, pp. 3–10.

[32] Inoue S. and Yamada, S., "Two-Dimensional Software Reliability Assessment with Testing-Coverage," in Second International Conference on Secure System Integration and Reliability Improvement,, 2008.

[33] Kapur P K., Aggarwal A.., Kapoor K and Kaur G "Optimal Testing Resource Allocation for Modular Software Considering Cost, Testing Effort and Reliability using Genetic Algorithm" International Journal of Reliability, Quality and Safety Engineering, Vol. 16, No. 6, pp. 495–508, 2009.

[34] Kapur P. K., et al., "Two Dimensional Flexible Software Reliability Growth Model And Related Release Policy," in 4th National Conference; INDIACom-2010, New Delhi, 2010.

[35] Yamada S., Osaki S., "Optimal software release policies with simultaneous cost and reliability requirements", *European Journal of Operational Research*, vol. 31, no. 1, pp. 46-51, 1987

[36] Huang C.Y., Lyu M.R., "Optimal Release Time for Software Systems Considering Cost, Testing Effort, and Testing Efficiency", *IEEE Transactions on Reliability*, vol. 54, no. 4, Dec 2005.

[37] Kapur P.K., Gupta D., Gupta A., Jha P.C., "Effect of Introduction of faults and Imperfect Debugging on Release Time", *In Ratio Mathematica*, vol. 18, pp. 62-90, 2008

[38] Goldberg D.E., *Genetic Algorithms in Search of Optimization and Machine Learning*, Addison-Wesley, 1989.