COMMUNICATIONS
IN
DEPENDABILITY AND
QUALITY
MANAGEMENT
An International Journal

CDQM, An Int. J., Volume 17, Number 1, 2014, pp. 5-16

UDC 004.4:[517.93:519.245
ID NUMBER 207829004

Uncertainty Based Fault Removal Phenomenon and Successive Software Releases Planning

Ompal Singh[1], Adarsh Anand[1], Deepti Aggrawal[1]
and ***Ljubisa Papic[2]***

[1] Department of Operational Research, University of Delhi, Delhi,India
E-mail: drompalsingh@live.com, adarsh.anand86@gmail.com, deepti.aggrawal@gmail.com
[2] DQM Research Center, P. O. Box 132, 32102 Cacak, Serbia
E-Mail: dqmcenter@open.telekom.rs

accepted February 14, 2014

Summary
As the size of software system is large and the number of faults detected during the testing phase becomes large, so th change of the number of faults that are detected and removed through each debugging becomes sufficiently smal compared with the initial fault content at the beginning of the testing phase. In such a situation, we can model th software fault detection process as a stochastic process with continuous-state space. In the present article, we hav described the fault-detection process during the testing phase of successive releases of the software by applying mathematical technique of stochastic differential equations (SDE) of itô type. Logistic rate function has been chosen t identify and remove the faults lying dormant in the software. We have further investigated an optimal bi-criterio release planning for successive software releases that maximizes the reliability and minimizes the cost of testing of th release that is to be brought into market under the dual constraints of budget and achieving a desired level o reliability. Results are supplemented by a numerical example.

Key words: *Successive software release planning, multi-up gradations, software reliability growth model, stochastic differential.*

1. INTRODUCTION

Firms that have upgraded their software skillfully adapting the technological advancements into their products and processes have not only survived but prospered. Expanding high technology has had a significant impact on virtually all industries. Today better known software developing companies like Microsoft, IBM, Adobe and Wipro etc. are known for their innovation strategies and

frequent introduction of an advanced version of the software. This successful introduction contributes substantially to long-term financial success and is an effective strategy to increase primary demand. It strengthens the competitive position of the company in the market. But the risk is formidable as for most software organizations especially the development are associated with high cost and risks. This is because upgrading a software application is a complex task where the upgraded and existing system may differ in the performance, interface and functionality etc. although the developers upgrades the software in order to improve the software product, which also includes the possibility that the upgrade version will worsen. The testing team is always interested in knowing the bugs present in the software which will decide the utility of up-graded software. Safe up-gradation can improve the behavior of the system and can preserve market for company; however risky up-gradation can cause critical error in system. for example in October 2005, a glitch in a software upgrade caused trading on the Tokyo Stock Exchange to shut down for most of the day [17], in 1991 after changing three lines code in a signaling program which contained millions lines of code, the local telephone systems in California and the eastern seaboard came to stop [5,10]. Similar gaffes have occurred from important government systems [10] to freeware on the internet. Sometimes Upgrades can worsen a product and user may prefer an older version.

The software failures may be due to errors, ambiguities, oversights or misinterpretations of the specifications that the software is supposed to satisfy, incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems. A plethora of software reliability models have been developed in the literature. Goel and Okumoto [6] proposed a SRGM, which describe the fault detection rate, as a non homogeneous Poisson process (NHPP). Later, based on this model, eminent researchers like Yamada [13], Pham [3] and Kapur [7, 18] etc. after bringing some new assumptions, proposed several reliability models in the literature. The K-G model [8, 18] can be described by following mathematical structure:

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

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

Recently Kapur et.al [10] developed a multi up-gradation reliability model, considering that cumulative faults in each generation depend on all previous releases and also assumes that fault is removed with certainty. But the proposed model is based on the assumption that the overall fault removal of the new release depends on the reported faults from the just previous release of the software and on the faults generated due to adding some new functionalities (add-ons/up-gradations) to the existing software system. Therefore, it's not necessary to consider the faults of all previous releases. This takes less time of the testing team in comparison to test the complete software together (i.e. all releases together).

Several NHPP based SRGMs have been developed in the literature, treating fault detection process during testing phase as a discrete counting process. Yamada et. al [15] asserted that if the size of the software system is large then the number of fault detected during the testing phase becomes large and the change of the number of faults which are detected and removed through each debugging activities becomes sufficiently small compared with the initial fault content at the beginning of the testing phase [18]. Therefore, in order to describe the stochastic behavior of the fault detection process, a stochastic model with continuous state space can be used. Yamada, Nishigaki and Kimura [15], Yamda and Tamura [16]; Lee, Kim and Park [1] have studied the stochastic behavior

of fault detection process described by stochastic process model with continuous state space. Recently Kapur et.al [11] also proposed a SDE based flexile SRGM.

Although testing is an efficient way to detect and remove faults so as to avoid failure of a software system, but exhaustive testing is impractical. Therefore, software developers need to decide when to stop testing the current release of the software and come up with up-graded version of the software system for the customers. Software release planning problems for single release software have been discussed and solved in different ways in literature. One of these is to find release time so that the total cost of software testing is minimized [12, 14, 18]. Some of the release time problems are based upon maximizing the reliability of software. Models that minimize the number of remaining faults in the software or the failure intensity also lie under this category [6, 18]. Release time problems have also been formulated for minimizing cost with minimum reliability requirement or maximizing reliability subject to budgetary constraint [12]. Bi-criterion release policy [18] that simultaneously maximizes reliability and minimizes cost have also been studied in literature for single release software.

The above literature review on release planning problems do not take into account the impact of coming up with successive releases of a software in release planning decisions. This paper helps answer the question of when to stop testing the current release of the software when we have the dual objective of minimizing the cost and maximizing the reliability of the version that has to be released into the market under the constraints of budget and reliability requirement. The formulated problem is solved using genetic algorithm.

The rest of the paper is organized as follows: Sec 1 and 2 gives the information about Notations and assumptions used, Sec 3 enlightens the SDE based modeling framework. In Sec 4, we have discussed about data set and comparison criteria used. In Sec 5 bi-criterion release planning problem is formulated and genetic algorithm is presented. Sec 6 gives numerical illustration of the formulated problem. At last, conclusion followed by Acknowledgement and References has been provided.

Notations

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

$E(m(t)) = m^*(t)$: The mean value function or the expected number of faults detected or removed by 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.

$F_{ij}(t)$: Probability distribution function associated with each Release (i=1to 4), (j=1,2).

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

σ: Positive constant that represents the magnitude of the irregular fluctuation for faults before and after change point.

b_i: fault removal rate per remaining faults.

i=1 to 4.

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

2. ASSUMPTIONS

The proposed model is based on SDE of it^o type with following assumption:

1. The software fault detection Process is modeled as a stochastic process with a continuous state space.
2. Software is subject to failure during execution caused by faults remaining in the software.
3. Let m(t) be a random variable which presents the number of software faults detected in the software system up to testing time t. The faults detected in $t+\Delta t$ are proportional to the mean number of faults remaining in the system.

3. SDE BASED MODELING OF UP-GRADATIONS FOR EACH RELEASE

Yamada et al [15] proposed a simplified software reliability growth model to describe the fault detection process during the testing phase by applying *itô's* type Stochastic Differential Equation (SDE) and have compared the continuous-state space SRGM with the NHPP. Lee *et al.* [1] used SDE to represent a per-fault detection rate that incorporate an irregular fluctuation instead of an NHPP, and consider a per-fault detection rate that depends on the testing time *t*. Recently, Yamada et al [16] have proposed a flexible Stochastic Differential Equation Model describing a fault-detection process during the system-testing phase of the distributed development environment [18]. Using the hazard rate approach in deriving the mean value function of cumulative number of faults removed, we have:

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

So the corresponding differential equation is given by:

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

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

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

let $\gamma(t)$ be a standard Guassian white noise and σ be a positive constant representing a magnitude of the irregular fluctuations. So the above equation can be written as:

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

The above equation can be extended to the following stochastic differential equation of an $\hat{ito}$ type:

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

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

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

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

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

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

the expected value is:

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

Release 1

The most important phase in the software development life cycle is testing. 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. These finite numbers of bugs are then removed perfectly and mathematical equation for it is given as under:

$$m_1^*(t) = a_1 F_1(t) \qquad 0 < t < t_1 \quad (2)$$

where,

$$F_1(t) = \left[1 - \left(\frac{(1+\beta_1)}{1+\beta_1 e^{-b_1 t}}\right)e^{-b_1 t + \frac{1}{2}\sigma^2 t}\right]$$

Release 2

After first release, the company has information about the reported bugs from the users, hence in order to attract more customers, a company adds some new functionality to the existing software system. Adding some new functionality to the software leads to change in the code. These new specifications in the code lead to increase in the fault content. Now the testing team starts testing the upgraded system, besides this the testing team considers dependency and effect of adding new functionalities with existing system. In this period when there are two versions of the software, while we are in the testing phase of the Release 2, we are also in the operational phase for the Release 1, hence the leftover fault content of the first version i.e $a_1(1-F_1(t_1))$ interacts with new fault detection rate i.e. $F_2(t-t_1)$. In addition a fraction of faults generated due to enhancement of

the features also get removed with this new rate. The mathematical equation of these finite numbers of faults removed can be given by:

$$m_2^*(t) = (a_2 + a_1(1 - F_1(t_1)).F_2(t - t_1) \ , t_1 \le t \le t_2 \qquad (3)$$

where:

$$F_2(t - t_1) = \left[1 - \left(\frac{(1+\beta_2)}{1+\beta_2 e^{-b_2(t-t_1)}} \right) e^{-b_2(t-t_1)+\frac{1}{2}\sigma^2.(t-t_1)} \right]$$

Release 3

Technological changes and stiff competition forces the software developer to add certain more features to the software. Now as discussed above, the testing team starts testing the upgraded system and simultaneously keeps a check on the operational phase of Release 2 as well. Therefore, the leftover faults from Release 2 i.e. $a_2(1 - F_2(t_2 - t_1))$ interact with the new fault detection/correction rate $F_3(t - t_2)$. Besides this the testing team removes the new faults with this new fault detection rate for the existing system. The mathematical equation of these finite numbers of faults removed can be given by:

$$m_3^*(t) = (a_3 + a_2(1 - F_2(t_2 - t_1)).F_3(t - t_2) \ , t_2 \le t \le t_3 \qquad (4)$$

where,

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

The process of adding new functionalities is an ongoing process. These add-ons keep on happening till software is there in the market. On similar basis as earlier, the mathematical equation for Release 4 can be given as:

$$m_4^*(t) = (a_4 + a_3(1 - F_3(t_3 - t_2)).F_4(t - t_3) \ , t_3 \le t \le t_4 \qquad (5)$$

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

4. MODEL VALIDATION, DATA SET AND DATA ANALYSIS

Figure 1-4 shows the estimated and the actual values of the number of faults removed for four releases. To check the validity of the proposed model and to describe the software reliability growth, it has been tested on tandem computer four release data set. Also we have used non linear least square technique in SPSS software for estimation of parameters. Estimated value of parameters of each releases are given in Table 1. Table 2 shows the comparison criterion of the four software releases. Based on data available given in Table1, the performance analysis of proposed model is measured by the four common criteria, Bias, Variation, RMSPE, MSE.

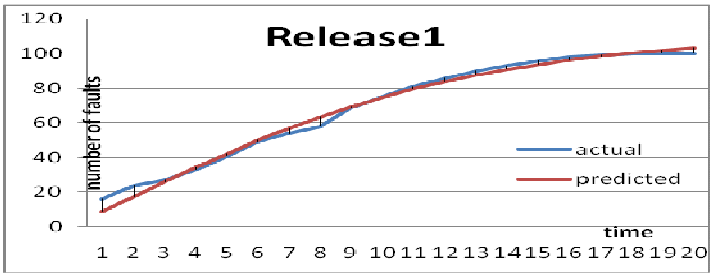

Figure 1. Goodness of fit curve for Release 1

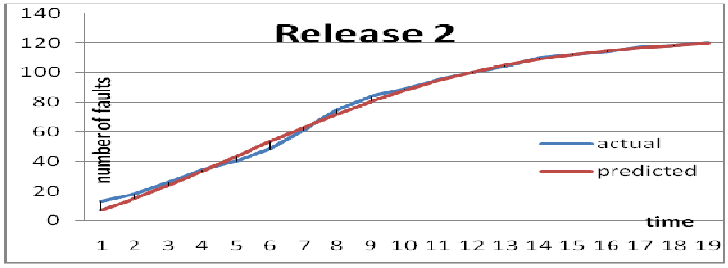

Figure 2. Goodness of fit curve for Release 2

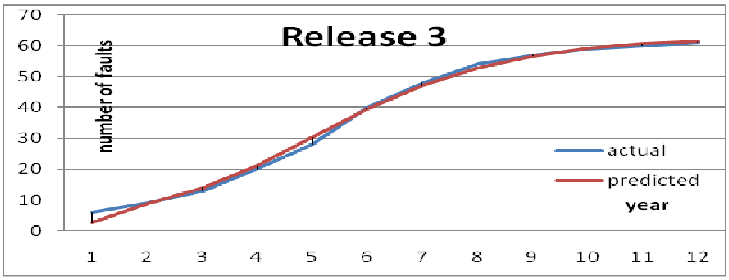

Figure 3. Goodness of fit curve for Release 3

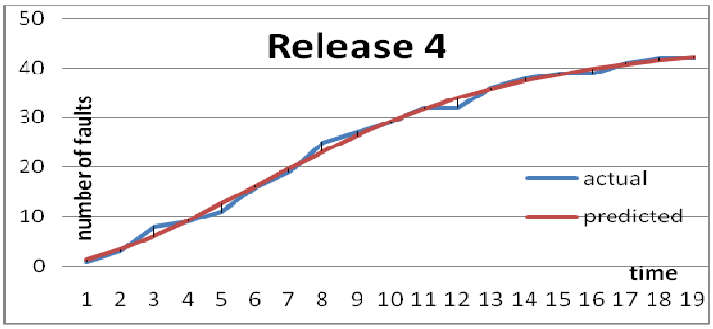

Figure 4. Goodness of fit curve for Release 4

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

Comparison	Release 1	Release 2	Release 3	Release 4
R^2	0.989	0.995	0.996	0.994
Bias	0.4352	0.3400	0.0762	-0.0509
MSE	8.9742	6.0013	1.7849	1.0711
Variation	3.0417	2.4925	1.3931	1.0620
RMSPE	3.0727	2.5156	1.3952	1.0632

5. OPTIMAL RELEASE PLANNING PROBLEM AND SOLUTION METHOD

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

5.1 Modeling Cost Function

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

$$C_n(t)=C_{n1}m_n(t)+C_{n2}\left(\left(a_n+a_{n-1}\left(1-F_{n-1}(t_{n-1})\right)\right)-m_n(t)\right)+C_{n3}t; \tag{6}$$

Equation (6) can be re-writen as:

$$\begin{aligned} C_n(t)= & C_{n1}a_n\left[1-\left(\frac{(1+\beta_n)}{1+\beta_n e^{-b_n t}}\right)e^{-b_n t+\frac{1}{2}\sigma^2 t}\right] \\ & +C_{n2}\left(\begin{array}{l}\left(a_n+a_{n-1}\left(1-(1-\left(\frac{(1+\beta_{n-1})}{1+\beta_{n-1}e^{-b_{n-1}(t-t_{n-2})}}\right)e^{-b_4(t-t_{n-2})+\frac{1}{2}\sigma^2(t-t_{n-2})}\right)\right)\right) \\ -a_n\left\{1-\left(\frac{(1+\beta_n)}{1+\beta_n e^{-b_n t}}\right)e^{-b_n t+\frac{1}{2}\sigma^2 t}\right\}\end{array}\right) \\ & +C_{n3}t; \end{aligned} \tag{7}$$

5.2 Reliability Evaluation

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

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

5.3 Modeling Release Time Problem

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

$$\begin{aligned} &\max\ R(x/T) \\ &\min\ C(T) \\ &s.t \\ &C(T) \leq C_B \\ &R(x/T) \geq R_0 \\ &T \geq 0,\ R_0 < 1 \end{aligned} \tag{8}$$

Alternately, we may write:

$$\begin{aligned} &\max \log R(x/T) \\ &\min \overline{C}(T) \\ &s.t \\ &\overline{C}(T) \leq 1 \\ &R(x/T) \geq R_0 \\ &T \geq 0, 0 < R_o < 1 \end{aligned} \tag{9}$$

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

Thus the previously stated formula is further reformulated as:

$$\begin{aligned} &\min K(T) = \lambda_1 \overline{C}(T) - \lambda_2 \log R(x/T) \\ &s.t \\ &\overline{C}(T) \leq 1 \\ &R(x/T) \geq R_0 \\ &T \geq 0, 0 < R_0 < 1 \end{aligned} \tag{10}$$

5.4 Genetic Algorithm

The GA algorithm steps for solving the release planning problem is as follows (D E Goldberg 1989):

Step 1: Start.

Step 2: Generate random population of chromosomes:- A chromosome comprises genes where each gene represents a specific attribute of the solution. The task of designing an appropriate chromosome representation of solutions of the problem is extremely crucial for the proper and

successful functioning of GA. Here a chromosome is a set of modules resources consumed as part of the total testing effort. It is initialized to random values within the limits of each variable.
Step 3: Evaluate the fitness of each chromosome in the population:-In our release planning problem, the fitness function is the objective of optimization problem along with the penalties of the constraint that is not met.
Step 4: Create a new population by repeating following steps until the new population is complete:

- [Selection] Select two parent chromosomes from a population according to their fitness (We use Tournament selection without replacement).
- [Crossover] With a crossover probability, cross over the parents to form new offspring (children). If no crossover is performed, offspring is the exact copy of parents.
- [Mutation] With a mutation probability, mutate offspring at each locus (position in chromosome).
- [Accepting] Place new offspring in the new population.
- [Replace] Use new generated population for further part of the algorithm.
- [Test] If the end condition is satisfied, stop and return the best solution in the current population.
- [Loop] Go to step 3 for fitness evaluation.

6. NUMERICAL EXAMPLE

As an example here we choose the same data set of four releases taken in section 4. In this data set first, second and third release have already been into market. The problem formulated in section 5 determines when to stop testing the fourth release of the software such that the cost of testing is minimized and reliability is maximized.

In order to determine the optimal release time and optimal resource consumption for the fourth release we make use of the estimated values of the parameters of third and fourth release given in Table 1. With these parameter values we solved the following problem using genetic algorithm method given in section 6. Further we assume $C_1 = 100$, $C_2 = 150$, $C_3 = 50$ and it is desired that at least 0.95 proportion of faults should be removed from 4^{th} release. The budget is assumed to be 10000 units and x in the reliability evaluation is taken as 7 days (i.e. 1 week). The problem is solved using Matlab software under VC++ (6.0) compiler.

Min:

$$C_4(t) = C_{4_1} a_4 \cdot \left(1 - \left(\frac{(1+\beta_4)}{1+\beta_4 e^{-b_4(t-t_3)}}\right) e^{-b_4(t-t_3)+\frac{1}{2}\sigma^2(t-t_3)}\right) + C_{4_2} \begin{pmatrix} \left(a_4 + a_3\left(1-(1-\left(\frac{(1+\beta_3)}{1+\beta_3 e^{-b_3(t-t_2)}}\right) e^{-b_3(t-t_2)+\frac{1}{2}\sigma^2(t-t_2)})\right)\right) \\ -a_4 \cdot \left(1 - \left(\frac{(1+\beta_4)}{1+\beta_4 e^{-b_4(t-t_3)}}\right) e^{-b_4(t-t_3)+\frac{1}{2}\sigma^2(t-t_3)}\right) \end{pmatrix} + C_{4_3} t \quad (11)$$

Max:

$$R(x/T) = e^{-(m_4(t+x)-m_4(t))}$$

subject to (11)

$$C_4(T) \le 10000$$
$$R(x/T) \ge 0.85$$

Writing an alternate form for equation (11) we have:

$$\begin{aligned} &\min \overline{C}_4(T) \\ &\max \log R(x/T) \\ &s.t \\ &\overline{C}_4(T) \le 1 \\ &R(x/T) \ge 0.85 \end{aligned} \tag{12}$$

Where, $\overline{C}_4(T) = \frac{C_4(T)}{10000}$ Assuming that both the objectives carry equal importance i.e. $\lambda_1 = 0.5$ *and* $\lambda_2 = 0.5$ we have the problem as:

$$\begin{aligned} &\min K(T) = 0.5\overline{C}(T) - 0.5 \log R(x/T) \\ &s.t \\ &\overline{C}(T) \le 1000 \\ &R(x/T) \ge .85 \end{aligned}$$

The above problem is solved using GA. The parameters used in GA evaluation are given in Table 3. The crossover method taken is simulated binary crossover (SBX), and selection criterion is tournament selection without replacement.

Table 3. Parameters of GA

Parameter	Population Size	Number of Generations	Crossover Probability	Mutation Probability
Value	50	25	0.9	0.1

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

7. CONCLUSION

In this paper we have proposed an SRGM for the successive release of the software using stochastic differential equations. While modeling the successive generations of the software we have assumed the interaction between the faults left in the just previous release and the new release. The stochastic process can be thought of as capturing uncertainties generated by stochastic fluctuations due to environmental conditions or so. The proposed multi release model is estimated on the real data set of four releases. Then a release planning problem is formulated and 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. A numerical illustration is also given for the developed optimal release planning problem. The model can be extended by classifying the severity of faults, lying in the software. Some faults are easy to remove and some take more time to leave the system. This classification can be worked on in the future.

ACKNOWLEDGEMENT

The research work presented in this paper is supported by grants to the first and second author from University of Delhi, R&D Grant No- DRCH/R&D/2013-14/4155, and Delhi, India.

REFERENCES

[1] C. H. Lee, YT Kim and DH Park, S-Shaped Software Reliability Growth Models derived from Stochastic Differential Equations, IIE Transactions, Vol. 36, pp. 1193-1199, 2004.

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

[3] H. Pham. System Software Reliability. Springer-Verlag. 2006.

[4] H. Pham, X. Zhang, NHPP Software reliability and Cost models with Testing Coverage European Journal of Operational Research, 145: 443-454, 2003.

[5] K. Khataneh, T. Mustafa, Software reliability modeling using soft computing technique, 1: 154-160, 2009.

[6] L. Goel, K. Okumoto, Time-dependent error-detection rate model for software reliability and other performance measures, IEEE Trans. on Reliability, 28(3): 206–211, 1979.

[7] P. K. Kapur, R. B. Garg, Software reliability growth model for an error-removal phenomenon, Software Engineering Journal, 7(4): 291–294, 1992.

[8] P. K. Kapur, R. B. Garg, S. Kumar, Contributions to Hardware and Software Reliability. World Scientific, Singapore, 1999.

[9] P. K. Kapur, O. Singh, J. Singh, An Irregular Fluctuation Based Multi up- gradation Model, Proceeding of international Conference on Reliability, Infocom Technology and Optimization (Trends and Future Directions), Lingaya's University, Faridabad, 734-741, 2010a.

[10] P. K. Kapur, A. Tandon, G. Kaur, Multi Up- gradation Software reliability Model, 2nd international conference on reliability, safety and hazard, ICRESH, 468-474, 2010b.

[11] P. K. Kapur, S. Anand, V. S. S. Yadavalli, F Beichelt, A Generalized Software Growth Model using Stochastic Differential Equations, Communication in Dependability and Quality Management – An International Journal, Serbia, Vol. 10, No. 3, pp. 82-96, 2007.

[12] P. K. Kapur, P. C. Jha, A. K. Bardhan, Optimal allocation of testing resource for a modular software, Asia-pacific journal of operational research, 24 (2), 1-22, 2004.

[13] S. Yamada, M. Ohba, S. Osaki, S-shaped Software reliability growth models and their applications, IEEE Trans. on Reliability, 33(4): 289–292, 1984.

[14] S. Yamada, M. Obha, S. Osaki, S-shaped reliability growth modeling for software error detection, IEEE Trans. on Reliability, R-32,475-478, 1983.

[15] S Yamada, Nishigaki, Kimura, A stochastic Differential Equation Model for software Reliability assessment and its Goodness of Fit, IJRA, 4(1): 1-11, 2003.

[16] S. Yamada, Y. Tamura, A Flexible Stochastic Differential Equation Model in Distributed Development Environment, European Journal of Operational Research, Vol. 168, pp. 143-152, 2006.

[17] Williams M, Software glitch halts Tokyo Stock Exchange. InfoWorld. http://www.infoworld.com/article/05/11/01HNtokyoexchange_1.html?APPLICATION%20PERFORMANCE%20MANAGEMENT. Retrieved 2008-07-30. Associated Press (2006-04-20). Official: Software glitch, not bomb, shut airport. MSNBC. http://www.msnbc.msn.com/id/12411853/. Retrieved 2008-07-30, 2005.

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