Patching: A Requirement for Complete Software Testing

Adarsh Anand [1], Subhrata Das [2], Ompal Singh [3]

(1) Department of Operational Research.University of Delhi (India)
E-mail: adarsh.anand@gmail.Com

(2) Department of Operational Research.University of Delhi (India)
E-mail: shus.das@gmail.Com

(3) Department of Operational Research.University of Delhi (India)
E-mail: drompalsingh1@gmail.Com

ABSTRACT

If any of the essentials for software testing (test strategy, testing plan, test cases, and test environment) is missing or inadequate, testing effort is most likely to fall short of what could have otherwise been achieved. Every product release in today's time shall meet a desired level of quality, and release processes undergo continual fine-tuning. One such aspect of this tuning comes from the concept of patching. Patching helps in no roll back policy for the broken or distorted releases and thus lays a helping hand in monitoring the quality of software. These software update ideology has undoubtedly helped in effectively improving usability and performance for software system. With this concept come two groups; tester and user who detect the defect (if any) in the software while it is in the operational stage. Inculcating this varied aspect, we propose an approach based on differing performance of tester and users during pre and post release of the software. The model has been validated on software failure data sets.

Keywords: Patching, Reliability, Software Testing, User.

1- INTRODUCTION

The function and market of software in the present era is escalating where merchandise gets incorporated and firms utilize networks for business manoeuver and infringement of security is major issue. For bug free operation software development and testing teams act pivotal in determining reliability and quality of the same. During the course of development of software rigorous testing plays important role in quality check. Logically only bug free software should be released in the marketplace. The synchronization of the actual bugs in the product and how the developers put efforts to debug after release is the end quality experience. The end product is the result of various phases in the development of the software. With the advancement of technology and the growth of internet, the role of software has apprehended a foremost shape of business activities as well as in our daily life. As software becomes a crucial part of our very high quality life, its failures are even more severe. Tremendous media reports are available on software failures. The following are some

well-known examples in recent years: in 2001, Nike Inc. suffered a loss of $100 million (USD) contributed by the supply chain management software system; in 2004, Ford Motor Co. lost approximately $400 million (USD) due to the software purchasing system and Hewlett-Packard Co. lost $160 million (USD) with a flaw in its ERP system [1]. The brutal competition between software firms imposes pressure on software developers to come up with their product in a short duration. Therefore firms are releasing their product earlier in the marketplace. Because of this reason, some known and unknown bugs remain at the time of software release. Thus, software testing holds utmost importance to improve the quality of the product.

The five core elements of testing are: test strategy, testing plan, test cases, test data and test environment. They help the firms in enhancing the effectiveness and efficiency of testing. The objective of test strategy indicates what kind of testing will work best towards making bug free software. Testing plan is simply that part of the project which deals with the testing tasks. It explains about who will do which tasks, when to start, when to end, how much efforts will be required and finding out other tasks on which it depends. Test cases are developed based on prioritized needs and acceptance criteria of the software, keeping in mind the customer's prominence on quality and risk assessment. Test data development and test case development are done simultaneously. In order to execute test cases testing team need to judiciously and systematically use the test data. Choosing the right environment is a part of testing plan and so should be selected beforehand [2].

One of the main questions to be answered on every software development is – how long testing should be done in order to ensure the software is reliable? It is too often that the testing phase is rushed and the software gets released with an unacceptable level of faults. On the other hand, it is likely to have diminishing yields while prolonging the testing past a certain point. In general, there may be a time point in life of the software where the profit earned in releasing it is greater than the value earned by continuing testing activities. With the aim of attaining more profit most commercial software are shipped early and are known to contain faults. Normally a firm tries to improve the quality of the software after its release by means of patching or updates. The final call on quality experienced by the end users is the proportion of the faults in the software when it get released and the frequent provision of patching and updates from the vender's end.

Patching is nothing but a piece of code in which designed software would enable updating or fixing and improvement of computer program. Patches also known as bug fixers which include fixing of security susceptibility and other bugs and also contribute to the performance. It is considered to be a rapid-repair job of a tester [3]. Bug-fixer is considered to be an instant solution maker available to the consumer; sometimes it can be downloaded from the website of software maker. From this, product developers find out an enhanced quality of their product. Recent trend has been seen in smart phones and antivirus products, from time to time the applications ask for updates. These up-

dates are provided by developers to counter effect the lurking faults.

In this paper, we intend to study the concept of patching; for which the users are acting as a supportive hand. Firstly, the testing team investigates and removes the bugs throughout the planning horizon so as to fulfill user's need of satisfactory product. At the same time the firms wish to provide maintenance of the software even after its release and also provide users with updates. As soon as any bug is encountered by tester or user, errors are debugged and updates are provided by the developers. The remainder of this article is structured as follows: The background and theoretical framework is discussed in section 2. Notations are presented in section 3 whereas the modeling framework is employed in section 4 which provides fault removal process incorporating the concept of patching. Data analysis is presented in section 5. Conclusion and acknowledgement are supplemented in last sections.

2- BACKGROUND AND THEORITICAL FRAMEWORK

Numerous software reliability growth models have been proposed in the literature under particular set of assumptions based on Non-homogeneous Poisson Process (NHPP) [4, 5, 6]. Also there are various type of models based on the concept of perfect debugging environment, which means whenever developing team is removing detected bugs, it is removed perfectly and no new bugs are generated. The parsimonious model in the field of software reliability was proposed by Goel and Okumoto in the year 1979 [7]. Further, Yamada et al. [8] developed a software reliability growth model as two stage process incorporating the time lag between failure observation and correction i.e. the delayed S-shaped model. There are some other models based on this concept such as: Ohba model [9], Bittanti model [10] and Kapur & Garg model [11]. But in reality, it is not possible to remove each and every fault perfectly due to complexities in the software system. Thereafter, most of the researchers have extended their work by using some realistic issues such as imperfect debugging, error generation, testing effort, testing coverage, change point, learning phenomena of software developers and fault severity etc. [4, 5, 6, 12, 13]. Also, to satisfy customer's need and meet the competitive edge, enhancing the features of the existing products by the developers is mandate. So, in later years, Kapur et al. [14] have contributed to the literature by proposing a multi up-gradation software reliability model. Working on the same line Singh et al. [15] proposed a successive release model in which fault removal phenomena of new release depends upon just previous release. A release time problem based on multi attribute theory was proposed by Singh et al. [16, 17]. Further the work has been extended to different attributes by Singh et al. [18]. Some studies have focused on multi up-gradation models considering the impact of fault severity, imperfect debugging environment, uncertainty, and release time problem etc. [19, 20, 21, 22, 23]. Recently, Anand et al. [24] formulated the generalized framework for fault removal phenomenon using different distributions based on the concept of up-gradation of the features and undetected faults from operational phase of preceding releases. Moreover, this concept has been extended using severity of faults in operational phase by Anand et al. [25].

In the entire list of conventional models, researchers have talked about testing phase of the software before release. If we test the software after the release, usability and performance of software will improve and it will also extend the time period of testing. Keeping the software under testing environment even after the release of software is termed as patching. The study by Jiang and Sarkar has focused on a software release time problem incorporating the concept of patching [26]. Recent study by Das et al. [27] defined a mathematical approach for fault removal phenomena considering the idea of patching and used convolution probability function to study the joint effort of tester and user. Further, they depicted the economic significance of the model. Anand et al. [28] developed a scheduling policy for a software product using the concept of patching that makes the system more cost effective. Considering the above mention aspects of patching, here we have studied the time of release and the time at which the testing is stopped. We have proposed a fault removal phenomenon which is able to compute the bugs which are debugged in In-house testing as well as in the field testing.

3-NOTATIONS

$F(t)$	Probability distributions function for fault removal process.
$f(t)$	Probability density function
$m(t)$	Expected number of faults removed by time t.
a	Total number of faults in the software.
b_1	Fault detection/correction rate of tester before release of software.
b_2	Fault detection/correction rate of tester after release of software.
b_3	Fault detection/correction rate of user after release of software.
τ	Software release Time.
T	Testing stop time.

4- MODELING FRAMEWORK

The role of software testers is integral in the creation of the software and are involved in the quality assurance stage. Generally they conduct manual testing and perform different combinations of test cases so as to insure that it is bug free. The major aim of testing process is to check the software for its quality in terms of highest possible level of reliability. In general, as soon as firms release the product they start upgrading in terms of launching its multiple releases. But recently a different trend has been visualized like in the case of anti-viruses, firms keep on providing the user significant updates of the software in terms of patches to make it adaptable for newly developed malicious activities or overcome weak area of the software. The weak areas can be the code of software containing some bugs or the zones from which the software security can be bypassed.

Before explaining the mathematical modeling framework incorporating patching, an important concept called unification is essential to discuss. This methodology presents a generic approach of taking different forms of $F(t)$, to

define different fault removal phenomena that exist in literature [4]. The mean value function of the software reliability growth model can be represented as [4]:

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

where $\frac{f(t)}{1-F(t)}$ is the failure correction rate per remaining faults or hazard rate function. This function shows the picture of failure changes over the systems [4].

Solving the (1) using boundary conditions $m(t=0)=0$, we obtain [28]

$$m(t) = a.F(t) \ , \ 0 < t < \tau \tag{2}$$

Equation (2) represents the cumulative number of faults removed by time t. The expression written in (2) has wide range of acceptance in software reliability literature. As stated earlier; the concept of unification approach [28] has been utilized for deducing the fault count in the system.

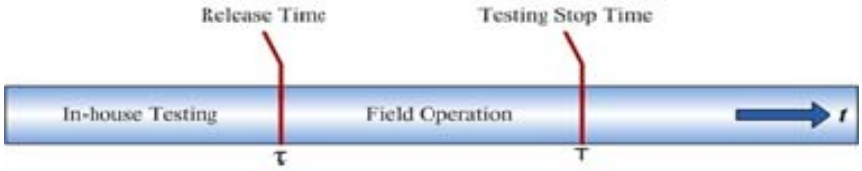

Figure 1. Testing Process [27]

Let us assume that the release time of the software as 'τ' is different from the time at which software firm wish to stop the testing of the software at 't'. That is, we have considered that software is release in field prior to the time till which the debugging activity is performed. We have divide the testing period into two time intervals $[0,\tau)$ and $[\tau,T]$ depicting the time the software was in testing process before and after its release respectively. In the period of release, the testers are the only debuggers who are directly or indirectly involved in finding and fixing the bugs while, after release the users are continuously using the software and it might be the case that they encountered some flaws and report to the firms. As soon as bug is reported by users, debugging team may remove the faults and simultaneously provide patches. Here again comes the role of testers in debugging the reported bugs (see fig. 1). Patching is the practice of making corrections in the source code which fix the bugs encountered while using the software. It is a solution to common problems that may exist in the real time environment which have led to increased usage.

Since $m(t)$ follows NHPP; we assume that the new removal function which is actually combination of time intervals; also follows NHPP criteria. Further, as a part of assumption, we assume that the removal function behaves exponentially, in order to state that as and when the bugs are detected, they are perfectly removed [4]. So making use of (2) the mean value function in $[0,\tau]$ can be given as follows:

$$m_1(\tau) = aF_1(\tau) = a\left(1 - e^{-b_1\tau}\right) \tag{3}$$

After time point τ, the software is released in the field and users start using the product and there might be case when users come across some loop holes. Thus, users are actively involved in the debugging process; although testers are available throughout the testing process. For this very reason, we have assumed that tester and user are in debugging process from τ onwards.

Assuming that the rate of fault detection/ correction by user as $F_U(t-\tau)$, the number of removal during a small time interval $[t-\varepsilon, t]$ can be expressed as:

$$A.\left[F_T(t) - F_T(t-\varepsilon)\right].F_U(t-\tau) \tag{4}$$

where $A = a.\left[1 - F(\tau)\right]$ represents the count of fault latent in the software after its release.

Hence, the instantaneous rate of debugging by the user at time t, denoted by $u(t)$ can be given by

$$\begin{aligned} u(t) &= \lim_{\varepsilon \to 0} \frac{A.\left[F_T(t) - F_T(t-\tau)\right]}{\varepsilon} F_U(t-\tau) \\ &= A.f_T(t).F_U(t-\tau) \end{aligned} \tag{5}$$

where $f_T(t)$ is the derivative of $F_T(t)$ and represents the removal rate of testers at time t. Hence, the cumulative number of faults removed by time T can be given by

$$U(T) = A\int_{\tau}^{T} f_T(t).F_U(t-\tau)dt \tag{6}$$

It is to note that, along with detection from users, testers are also involved in defect detection process. Therefore, number of faults that will be removed by testers in $[\tau, T]$ is some proportion of faults which were leftover by testers in the period $[0,\tau]$ and faults not reported by the users in the planning

horizon $[\tau, T]$. The novelty lies here that when user encounter some faults it reports back to the testers which in turn work on it and patches are issued to overcome such situations. The problems left unnoticed by the users are handeled by the testing fraterinity who regularly keep a track on the software even post release.

Inculcating the concept of firms coming up with patch post their software release, the cumulative number of remaining faults becomes $[A - U(t)]$ instead of A (to be debugged by tester). Here it is to be noted that $U(t)$ stands for number of faults identified by users at any time point $'t'$. Upon further analysis, it may be concluded that the efficiency of tester is always higher than that of the user during the fault detection process. So we define a multiplier for defining the tester's efficiency being dependent on user's detection rate. Note that, $F_U(t-\tau)$ represents the removal rate of users after time $'\tau'$, who are actively participating in defect detection activity. The multiplier is designed in such a manner that tester's efficiency is comparatively higher than that of the user. Thus number of faults removed by tester at some time point $'t'$ can be modeled as given by:

$$\begin{aligned} w(t) &= [A - U(t)] \frac{A.F_T(t)}{[A - U(t)]} . f_U(t-\tau) \\ &= A.F_T(t) . f_U(t-\tau) \end{aligned} \tag{7}$$

And its cumulative form is as follows:

$$W(t) = A \int_{\tau}^{T} F_T(t) . f_U(t-\tau) dt \tag{8}$$

Here, our analysis is more focused in modeling the defects detected under the post release testing of the software and it is a part of assumption that as soon as defects are detected corresponding patches are issued and fixed by the firms. Thus, cumulative number of defects detected by testers and users after the release of the software is given:

$$\begin{aligned} & U(T) + W(T) \\ &= \int_{\tau}^{T} A.[f_T(t).F_U(t-\tau) + F_T(t).f_U(t-\tau)] dt \end{aligned} \tag{9}$$

Equation (9) represents the number of bugs fixed during the period in which the firm provide patches according to users report and result of testers' detection process.

Therefore, total number of faults during the planning horizon $[0,\tau) and [\tau,T]$ which comprises of faults removed prior to release and left over faults that are dealt by testers and users together can be mathematically described as follows:

$$m(T)=\begin{cases} a.F_T(\tau) & ;T \le \tau \\ \int_{\tau}^{T} A.\left[f_T(t).F_U(t-\tau)+F_T(t).f_U(t-\tau)\right]dt & ;T > \tau \end{cases} \quad (10)$$

Equation (10) can be rewritten as:

$$m(T)=\begin{cases} a.F_T(\tau) & ;T \le \tau \\ a.\left(1-F_T(\tau)\right).F_T(T).F_U(T-\tau) & ;\ T > \tau \end{cases} \quad (11)$$

Equation (11) presents a methodical approach to quantify the number of faults that are debugged before and after its release in the field, where after the release of the software efforts applied by users also facilitate in testing process.

5- DATA ANALYSIS

The proposed methodology is examined taking into consideration the concept of patching. To validate the proposed methodology, we have made use of two data sets, first data set (DS-I) collected from Tandem Computers [29] whereas second data set was presented by Musa et al. [30] and estimated the parameters using non linear least square by software package SAS. The estimated value of total number of faults for DS-I i.e. *a=103.0139* (actual number of total faults are 100), the rate by which tester debug the software before release i.e.b_1*=0.105654*, the rate by which testers debug the software after release i.e.b_2*=0.231635* and the rate by which users are rigorously involved in debugging process of the software after release i.e.b_3*=0.274044*. Similarly for DS-II, the estimated value of total number of faults i.e.*a=179.552* (actual faults are 136), the rate of fault removal before release of the software i.e.b_1*=0.01* and the rate of fault removal after release of the software by tester and user i.e.b_2*=0.1385* and b_3*=0.1662*.

Table 1. Comparison Criteria for DS-I & DS-II

	SSE	MSE	RMSE	R-Square	Adj.R-Square
DS-I	65.8041	4.1128	2.028	0.996	0.9952
DS-II	468.6	26.0342	5.1024	0.9906	0.9895

From table 1, it clearly indicates that this model fits well to the given data set as the values of SSE, MSE, and RMSE are significantly low and the value of

co-efficient of determination $\left(R^2\right)$ and $Adj\ R^2$ is closer to 1. The goodness of fit curve shows the little discrepancy between the actual and observed values and the fault removal phenomenon is captured appropriately in the developed methodology as shown in fig. (2) and fig. (3).

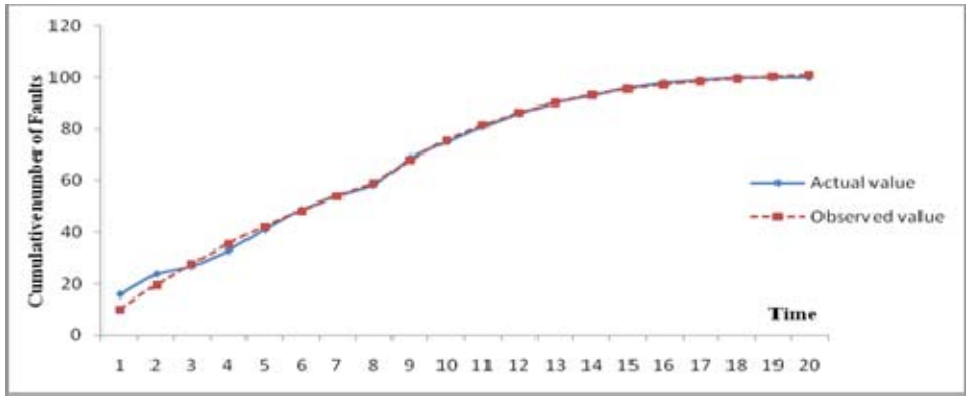

Figure 2. Goodness fit curve for the proposed methodology for DS-I

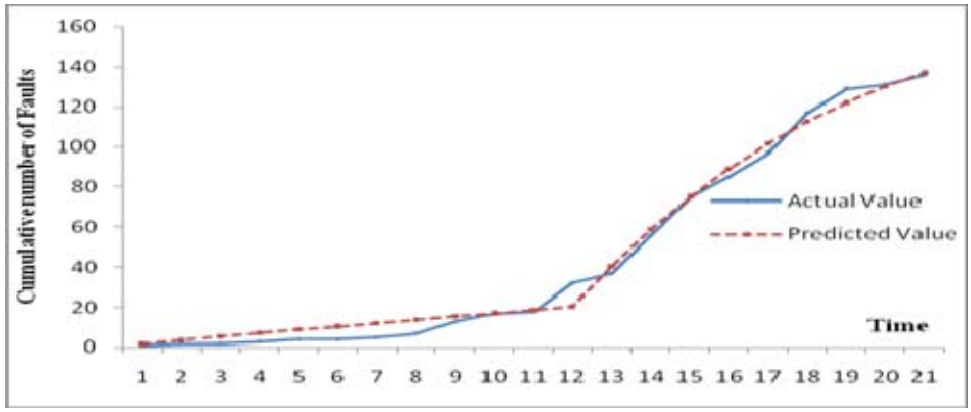

Figure 3. Goodness fit curve for the proposed methodology for DS-II

6- CONCLUSION

Due to the globalized nature of the market, the software industry is becoming competitive; also the repute of any software firm is related to its quality. Thus software engineers are keenly considering and studying the requisites of various software testing right from planning, strategizing, testing cases to testing environment, from the very beginning. The testing endeavour would seem to fall short in case any of the five rudiments (test strategy, testing plan, test cases, test data and test environment [2]) is missed or insufficient. Providing extended testing is a measure of quality which can be attained by providing patches to the

users. In the present article, we have incorporated the concept of patching; which has led to the creation of new aspect in the testing of entire arena of software system. This concept increases the total fault removal rate as both testers and users are using and debugging the latent faults present in the software. Thus a software reliability growth model based on their differing performance is formulated during testing and operational phase of software development life cycle. The results obtained are very promising.

ACKNOWLEDGMENT

The research work presented in this paper is supported by grants to the first author from University of Delhi R&D Grant No. RC/2015/9677, Delhi, India and third author from DST via DST PURSE phase II, India. Further the authors would like to thank the reviewers for suggesting qualitative changes in the paper.

REFERENCES

[1] R. N. Charette, “Why Software Fails”, http://spectrum.ieee.org/, July, 2015.

[2] D. L. Brown, “Five Essentials for Software Testing”, www.isixsigma.com, December 20, 2015.

[3] J. Dadzie, “Understanding Software Patching”, ACM Queue, 3(2), March 2005.

[4] P. K. Kapur, H. Pham, A. Gupta, and P. C. Jha, “Software Reliability Assessment with OR Application”, Springer, Berlin, 2011.

[5] P. K. Kapur, R. B. Garg, and S. Kumar, “Contributions to Hardware and Software reliability”, World Scientific Publishing Co. Ltd: Singapore, 1999.

[6] H. Pham, “System Software Reliability”, Springer-Verlag, 2006

[7] A. L. Goel, and K. Okumoto, “Time Dependent Error Detection Rate Model for Software Reliability and other Performance Measures”, IEEE Transaction Reliability, R-28(3), 206-211, 1979.

[8] S. Yamada, M. Ohba, and S. Osaki, “S-shaped Software Reliability Growth Models and their Applications,” IEEE Trans. on Reliability, 1984, Vol. 33, no. 4, pp. 289–292, 1984.

[9] M. Ohba, “Inflection S-shaped Software Reliability Growth Models”, in: S. Osaki, Y. Hatoyama (Eds.), Stochastic Models in Reliability Theory, Springer, Berlin, 144-162, 1984.

[10] S. Bittanti, P. Bolzern, E. Pedrotti, N. Pozzi, and R. Scattolini, “A Flexible Modelling Approach for Software Reliability Growth”, In: Goos G., Harmanis J (eds), Software reliability modelling and identification, Springer,

Berlin, 101-140, 1988.

[11] P. K. Kapur, and R. B. Garg, “A Software Reliability Growth Model for an Error Removal Phenomenon”, Software Engineering Journal, (7), 291-294, 1992.

[12] M. Trachtenberg, "A General Theory of Software-Reliability Modeling."Reliability, IEEE Transactions on Reliability, 39(1), 92-96, 1990.

[13] S. Yamada, J. Hishitani, and S. Osaki, “Software Reliability Growth Model with Weibull Testing Effort: A model and Application”, IEEE Transaction Reliability, 42, pp. 100-105, 1993.

[14] P. K. Kapur, A. Tandon, and G. Kaur, “Multi Up-gradations Software Reliability Model”, ICRESH, 468-474, 2010.

[15] O. Singh, P. K. Kapur, S. K. Khatri, and J. N. P. Singh, “Software Reliability Growth Modeling for Successive Releases”, proceeding of 4th International Conference on Quality, Reliability and Infocom Technology (ICQRIT), PP 77-87, 2012.

[16] O. Singh, D. Aggrawal, and P. K. Kapur, “Reliability Analysis and Optimal Release Time for a Software using Multi Attribute Utility Theory”, Communications in Dependability and Quality Management-An International Journal, Serbia, Vol. 5, No. 1, pp. 50-64, 2012.

[17] O. Singh, P. K. Kapur, and A. Anand, “A Multi Attribute Approach for Release Time and Reliability Trend Analysis of a Software”, international Journal of System Assurance and Engineering Management (IJSAEM), Vol. 3, Issue 3, pp. 246-254, 2012.

[18] O. Singh, J. N. P. Singh, A. Anand, and P. K. Kapur, “Optimal Release Time of Software: An integrated approach”, proceeding of 4th International DQM conference on life cycle engineering and management, Serbia, pp-148-161, 2013.

[19] P. K. Kapur, A. Anand, and O. Singh, “Modeling Successive Software Up-Gradations with Faults of Different Severity”, Proceedings of the 5th National Conference on Computing For Nation Development, ISSN 0973-7529 ISBN 978-93-80544-00-7, 2011.

[20] P. K. Kapur, O. Singh, A. Garmabaki and J. Singh, “Multi up-gradation Software Reliability Growth Model with Imperfect Debugging”, International Journal of systems Assurance Engineering and Management, 1(4),299-306, 2010.

[21] O. Singh, A. Anand, and D. Aggrawal, and L. Papic, “Uncertainty Based Fault Removal Phenomenon and Successive Software Releases Planning” Communications in Dependability and Quality Management-An International Journal, Serbia, Vol. 17, Issue 1, pp 5-17, 2014.

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

[23] O. Singh, P. K. Kapur, and A. Anand, “A Stochastic Formulation of Successive Software Releases with Fault Severity” Industrial Engineering and Engineering Management, 136-140, 2011.

[24] A. Anand, A. Singh, P.K. Kapur, and S. Das, “Modeling Conjoint Effect of Faults Testified from Operational Phase for Successive Software Releases”, Proceedings of the 5th International Conference on Life Cycle Engineering and Management (ICDQM), PP 83-94, 2014.

[25] A. Anand, O. Singh, and S. Das, “Fault Severity based Multi up-gradation Modeling Considering Testing and Operational Profile”, International Journal of Computer Applications (0975 – 8887), Volume 124 – No.4, 2015.

[26] S. Jiang and S. Sarkar, “Optimal Software Release Time with Patching Considered”, in Proc. 13th Annual Workshop Information technologies and Systems, Seattle, 61-66, 2003.

[27] S. Das, A. Anand, O. Singh, and J. Singh, “Influence of Patching on Optimal Planning for Software Release & Testing Time”, CDQM- An International Journal, Volume 18, Number 4, pp. 81-92, 2015.

[28] A. Anand, M. Agarwal, Y. Tamura, and S. Yamada, “Economic Impact of Software Patching and Optimal Release Scheduling”, Quality Reliability Engineering International, DOI: 10.1002/qre.1997, 2016,

[29] A. Wood, “Predicting Software Reliability”, IEEE Computer (11), pp. 69-77, 1996.

[30] J. D. Musa, A. Iannino A., and K. Okumoto, “Software Reliability: Measurement, Prediction, Application”, McGraw-Hill, New York, ISBN-0-07-044093-X, 1987.