A Flexible Reliability Growth Model for various Releases of Software under the Influence of Testing Resources

Jagvinder Singh[1], Ompal Singh[2,*], Deepti Aggrawal[3], Adarsh Anand[4] and Indarpal Singh[5]
[1]Maharaja Agrasen College, University of Delhi, India
[2,3,4]Department of Operational Research, University of Delhi, Delhi, India
[5]DNPG College, Gulaothi, Bulandshahar, UP, India

The life of software is very short in the environment of perfect competition market. Therefore the software developers have to come up with multiple releases in order to survive in the market. But, enhancing the product and coming up with multiple releases puts a constant pressure on even the best organized engineering organizations. The reason being, up-grading a software application is a complex process. Up-grades of software introduce the risk that the new release will contain a bug, causing the program to fail. Therefore, to capture the effect of faults generated in the software with multiple release, we have developed a multi release software reliability model in this paper. The model uniquely take into account the faults of the current release and the remaining faults of just previous release. The multi release software reliability growth model treats the faults removal rate as a function of testing resources consumed. In addition, most of the debugging process in real life is not perfect. Due to complexity and incomplete understanding of the software, the testing team may not be able to remove/correct the fault perfectly on observation/ detection of a failure and the original fault may remain resulting in the phenomenon known as imperfect debugging, or get replaced by another fault causing error generation. The effects of both type of imperfect debugging during testing phase are incorporated in our proposed multi-release model. The model developed is validated on real data set with software which has been released in the market with new features four times

Keywords: Flexible Reliability Growth Models, Various releases, Testing resources.

1. INTRODUCTION

Long-lived software systems evolve through new product releases. This evolution into market with multiple releases offers lot of advantages to the software development firms. It helps firm to uphold crucial maintenance revenue streams, to cope with rapidly changing technology which otherwise could make their product obsolete, protect competitive advantages and also provide bread and butter to the software companies.

In general, when software reaches a level of operational reliability desired by the software development firm, a new release is introduced in the market and the software gets upgraded. The term upgrade refers to the replacement of a product with a newer version or release of the same product. Apart from this, the need to advance the existing release of the software may arise due to following different reasons:

(a) **Problems with existing software release:** It may happen that the existing release of the software may not satisfy the targeted customers to their full expectation. In such case firm may bring a new release into the market.
(b) **Additional Functionality required:** Adding new features basically add life to the product. If the software development company doesn't add new features, than the customers might get bored with the same features which can lead to obsolescence of the product. This factor generally plays a major role in bringing of new release.
(c) **New hardware requiring updated software:** In today's hi-tech world the technology changes with the blink of an eye. This forces the developing firms to come out with new releases.
(d) **Competitive market conditions:** Nowadays software business is getting big portion of market trade and even big software companies like Microsoft, Yahoo, Google, Apple, Linux, Adobe, Macintosh are competing to survive in this competitive market scenario. Therefore, in such a competitive environment, if the developing firm doesn't bring releases of their product then they can easily loose their customers.

Although, launching software in different releases offers various advantages to development firms, but the constant pressure to market software releases, pricks the nerves of even the most well organized engineering organizations. The reason being, up-grading a software application is a complex process. The new and the old component in the different releases may vary in the functionality, interface, and performance. Even if, only the selected components of the software is changed or new featured are added into it, while the other parts of the application continue to function; but this process leads to an increase in the fault contents. Upgrades of software introduce the risk that the new release can contain a bug, causing the program to break down. There have been incidents in the history where upgrades caused the software to malfunction. 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 [1]. Similar blunders have occurred: from important government systems [2] to freeware on the internet. Thus, all the above advantages and risks involved with multiple releases of software brings into limelight the utmost need of modeling multi-release reliability growth models in software engineering. In this paper we develop modeling framework for such multi releases failure process, which is relatively an unexplored area in the field of software reliability.

Software reliability modeling has gained a lot of importance in the past years. Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment (ANSI definition). Abundance of software reliability models, considering only a single release of software, has been developed in the history of this subject. A Software Reliability Growth Model (SRGM) gives a mathematical relationship between time span of testing or using the software and the cumulative number of faults detected. It is used to evaluate the reliability of the software during testing and operational phases. Software testing involves running the software and checking for unexpected behavior in software output. The successful test can be considered to be one, which reveals the presence of the latent faults. The process of locating the faults and designing the procedures to detect them is called the debugging process. The software reliability model is the tool, which can be used to evaluate the software quantitatively, develop test status, schedule status and monitor the change in reliability performance. Several SRGMs that related fault detection process to the time spent in testing, with different mean value functions have been proposed in literature[3, 4,5]. These earlier SRGMs were developed to fit an exponential reliability growth curve and they are known as exponential SRGMs. In the real life software development projects, the non-uniform testing is more popular and hence the S-shaped growth curve has been observed in many software development projects. The cause of S-shapedness has been attributed to different reasons. Ohba & Yamada [6] and Yamada, Ohba & Osaki [7] attributed it to time delay between the fault removal and the initial failure observation which is result of the unskilledness of the testing team at the early stages of the test. There exists another category of models, classified as flexible models. These models can depict both exponential and S-shaped failure growth phenomenon depending upon the values of the parameters. Model due to Bittanti et al. [8], Kapur & Garg [9] are examples of flexible models.

Though these SRGMs related fault detection to the testing time, but the detection of faults can be more closely related to the amount of resource expended on testing and not just the amount of testing time elapsed. Testing resources are the resources that govern the pace of testing that include manpower and computing time. In the recent years, incorporating testing resources into SRGM's has received a lot of interest because such models relate to real world more closely [10,11,5,12]. This paper too treats the fault removal rate as a function of testing resources consumed.

In addition, most of the debugging process in real life is not perfect. During the fault removal process two possibilities can occur. It may happen that the fault, which was considered to be perfectly fixed, had been improperly repaired and resulted in same type of failure again. There is also a fine chance that some new faults might get introduced during the course of correcting. This situation is much dangerous than the former one, because in the first case the total fault content is not altered, whereas in latter, error generation resulted in increased fault content. The effects of both type of imperfect debugging during testing phase are incorporated in our proposed multi-release model.

This paper develops the relationship between features enhancement (i.e. the new release which is made by up-grading the just previous release) and software faults removal to develop a mathematical model. The mathematical structure identifies the behavior of faults removed during the testing of the software. The model is developed for N software releases in the software. It assumes that when the software is upgraded for the first time, some new functionality is added to the software. The new code written for the software enhancement leads to some faults in the software which are detected during the testing of the software. During the testing of the newly developed code, there is a possibility that the certain faults were lying dormant in the software which were not removed or detected in the previously released software version. The testing team also removes these faults before releasing the up graded version of software in the market.

The paper is organized as follows: section 2 provides a review of single release resource dependent SRGM with imperfect debugging. In section 3, a novel modeling framework for software with multiple releases is proposed. The parameter estimation results and data validation of the formulated model is shown in section 4. Finally conclusions and scope for future research is penned in section 5.

2. SINGLE RELEASE RESOURCE DEPENDENT FLEXIBLE SOFTWARE RELIABILITY GROWTH MODEL WITH IMPERFECT DEBUGGING

The resource dependent flexible SRGM presented in this section was proposed by Kapur et al. [13].

2.1. Notations

m(t)/m(W(t))	Expected number of faults removed in the time interval (0,t].
W(t)	Cumulative testing effort in the interval (0.t].
w(t)	Current testing-effort expenditure rate at testing time t . $\frac{d}{dt}W(t)=w(t)$
b	Constant fault detection rate.
p	Probability of perfect debugging of a fault.(0<p<1)
α	Constant rate of error generation.
a	Constant, representing the number of faults lying dormant in the software at the beginning of testing.
β	Constant in logistic learning function
γ	Parameter of Weibull distribution.
λ	Amount of testing-effort eventually consumed.

2.2. Assumptions

The software reliability growth model is based on Non Homogeneous Poisson Process (NHPP). The assumption that governs these models is, 'the software failure occurs at random times during testing caused by faults lying dormant in software.' And, for modeling the software fault detection phenomenon, counting process $\{N(t);t\geq 0\}$ is defined which represents the cumulative number of software faults detected by testing time t. The SRGM based on NHPP is formulated as:

$$Pr\{N(t)=n\}=\frac{(m(t))^n\cdot\exp\left(-m(t)\right)}{n!},\quad n=0,1,2,... \tag{1}$$

m(t)is the mean value function of the counting process N(t).

The other assumptions of the model are:

1. Software is subject to failures during execution caused by faults remaining in the software.
2. Each time a failure is observed, an immediate effort takes place to decide the cause of the failure in order to remove it.

3. Reliability growth during the testing phase is dependent upon the testing resources spend during testing.
4. During the fault removal process, following may occur:
   (a) Fault content is reduced by one with probability p.
   (b) Fault content remains unchanged with probability 1-p.
5. During the fault removal process, new faults can be generated and the fault generation rate is proportional to the rate of fault removal.
6. Fault removal rate per remaining fault is assumed to be non-decreasing inflection S-shaped logistic function.

2.3. Modeling Testing Resource Function

The developed SRGM takes into account the time dependent variation in testing resource consumption. The function that is used in this paper to explain the testing effort is Weibull function. It can be derived from the assumption that, 'The testing effort rate is proportional to the testing resources available'.

i.e. $$\frac{dW(t)}{dt}=V(t)\left[\lambda-W(t)\right] \tag{2}$$

Where v(t) is the time dependent rate at which testing resources are consumed, with respect to remaining available resources.

If v(t)= $v.\gamma\cdot t.^{\gamma-1}$, we get Weibull function as:

$$W(t)=\lambda\left(1-e^{-v\,t^{\gamma}}\right) \tag{3}$$

2.4. Model Development

Under the above assumptions, the removal phenomenon can be described with respect to time as follows:

$$\frac{dm(t)}{dt}=pb\big(W(t)\big)\big(a(t)-m(t)\big)\frac{dW(t)}{dt} \tag{4}$$

Where, $$b\big(W(t)\big)=\frac{b}{1+\beta e^{-bW(t)}} \quad and \quad a(t)=\big(a+\alpha m(t)\big) \tag{5}$$

Using (2.3) in (2.2) we have

$$\frac{dm(t)}{dt}=\frac{pb}{1+\beta e^{-bW(t)}}\big(a+\alpha m(t)-m(t)\big)\frac{dW(t)}{dt} \tag{6}$$

Solving Eq. (6) under the initial condition m(t=0) = 0 and W(t=0) = 0 we get

$$m(W(t)) = \frac{a}{1-\alpha}\left[1-\left(\frac{(1+\beta)\,e^{-bW(t)}}{1+\beta e^{-bW(t)}}\right)^{p(1-\alpha)}\right] \tag{7}$$

The above model is flexible SRGM. If the value of $\beta = 0$, the above model reduces to an exponential model with imperfect debugging.

3. RESOURCE DEPENDENT MULTI-RELEASE SOFTWARE RELIABILITY GROWTH MODEL

In modeling the proposed multi-release model apart from the assumptions stated in section 2, we assume that while modeling the mean value function for the next release the faults of the current release and the remaining faults of just previous release are considered. Also, we rewrite model equation (7) in the following form:

$$m\big(W(t)\big) = a^{*}.F(W(t)) \tag{8}$$

Where $a^{*} = \dfrac{a}{1-\alpha}$; $\qquad F(W(t)) = \left[1-\left(\dfrac{(1+\beta)\,e^{-bW(t)}}{1+\beta e^{-bW(t)}}\right)^{p(1-\alpha)}\right]$ (9)

First Release

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. In multi release model first release is the major step that can bring the software developing firm in limelight. Hence company pays an extraordinary attention on it. Therefore to bang into the market, the firm tests the software rigorously with an attempt to remove maximum number of faults lying dormant in the software. But, because of constraints on resources available, 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. The faults that get removed during the testing phase of the first release in the imperfect debugging environment can be represented mathematically by the following equation:

$$m_1(W_1(t)) = a_1^{*}.F_1(W_1(t))\,, \quad 0 \le t \le t_1 \tag{10}$$

Second Release

In order to sustain in the market company adds some new functionality into the software. Adding new feature implies adding more lines of code in the software coding which in turn results in increase in the complexity of software and thus leads to an increase in the fault content. Further, at the same time the firm has to take into consideration the errors that get reported in operational phase of the first release.

Therefore, in this period, when there are two versions of the software; $a_1^*\left(1-F_1\left(W_1(t_1)\right)\right)$ are the left over fault content of the first version which interacts with new detection/ correction rate of the testing phase of second release. In addition, a fraction of faults that are generated due to the enhancement of the features also are removed during the testing of current release, with new detection rate i.e. $F_2\left(W_2(t-t_1)\right)$. This change in the fault detection is due to change in time, change in the complexity due to new features, change in testing strategies etc. Therefore, the mathematical equation representing the cumulative number of faults removed in second release is given by:

$$m_2\left(W_2(t)\right)=(a_2^*+a_1^*\left(1-F_1\left(W_1(t_1)\right)\right)F_2\left(W_2(t-t_1)\right) \qquad t_1<t<t_2 \tag{11}$$

Next Releases

Under the stated postulation that the mean value function for the subsequent release is affected only by the faults of the current release and the remaining faults of just previous release we get the algebraic equations for the next releases in the environment of imperfect debugging as:

$$m_3\left(W_3(t)\right)=(a_3^*+a_2^*\left(1-F_2\left(W_2(t_2)\right)\right)F_3\left(W_3(t-t_2)\right) \qquad t_2<t<t_3 \tag{12}$$

$$m_4\left(W_4(t)\right)=(a_4^*+a_3^*\left(1-F_3\left(W_3(t_3)\right)\right)F_4\left(W_4(t-t_3)\right) \qquad t_3<t<t_4 \tag{13}$$

$$m_5\left(W_5(t)\right)=(a_5^*+a_4^*\left(1-F_4\left(W_4(t_4)\right)\right)F_5\left(W_4(t-t_4)\right) \qquad t_4<t<t_5 \tag{14}$$

.

.

.

$$m_N\left(W_N(t)\right)=(a_N^*+a_{N-1}^*\left(1-F_{N-1}\left(W_{N-1}(t_{N-1})\right)\right)F_N\left(W_N(t-t_{N-1})\right) \qquad t_{N-1}<t<t_N \tag{15}$$

4. ESTIMATION OF PARAMETERS MODEL VALIDATION AND COMPARISON CRITERIA

Parameter estimation is of prime significance in software reliability prediction. Parameter estimation is achieved by extensively used estimation techniques for non-linear models by the method of Non-linear Least Square (NLLS). In most of the cases solving this system of equations is difficult and contrary to the linear model fitting, as we cannot express the solution of this optimization problem analytically. Moreover, it requires numerical methods and huge computation time to solve the problem, which is not favored by the management and software engineering practitioners. Statistical software packages such as SPSS, SAS, and Mathematics etc. help to overcome this problem in which we can use the inbuilt software functions to solve these kinds of optimization problems to find the estimates of nonlinear models. In our study we have used the Statistical Package for Social Sciences (SPSS). SPSS is a comprehensive and flexible statistical analysis and data management system. It can take data from almost any type of file and use them to generate tabulated reports, charts, and plots of distributions and trends, descriptive statistics, and conduct complex statistical analysis. SPSS Regression Models enables the user to apply more sophisticated models to the data using its wide range of nonlinear regression models. For the estimation of the parameters of the proposed model, nonlinear regression (NLR) modules of SPSS have been used. The modules use the iterative estimation algorithms namely sequential quadratic programming and Levenberg-Marquardt method to find the least square estimates of the parameters.

Goodness of Fit Measures

The parameter estimated should be a “good” point estimate. To check the performance of these estimates we describe the following comparison criteria:

(a) Coefficient of Multiple Determination (R^2): This coefficient is defined as the ratio of the sum of squares resulting from the trend model to that from constant model subtracted from 1. It measures the percentage of total variation about the mean accounted for the fitted curve. It ranges in value from 0 to 1. Small values indicate that the model do not fit the data well. The larger R^2 , the better the model.

(b) Bias: The Bias is defined as:

$$\text{Bias} = \frac{1}{n}\sum_{k=1}^{n} \text{(Actual Value}(m(t_k)) - \text{Estimated Value}(m_k))$$

The difference between the observation and prediction of number of failures at any instant of time i is known as PE_i (prediction error). The average of PEs id known as bias lower the value of Bias better is the goodness of fit. A smaller Bias indicates a smaller fitting error and better performance.

(c) Variance : It is defined as :

$$Variance = \sqrt{\frac{1}{n-1}(Actual\ Value - Estimated\,Value - Bias)^2}$$

A smaller variance indicates a small fitting error and hence better performance.

(d) The Mean Square Error (MSE): It is defined as:

$$\frac{1}{n}\sum_{k=1}^{n}(Actual\ Value - Estimated\ Value)^2$$

Small values of MSE imply a small estimated error and thus better the model.

5. DATA SET AND DATA ANALYSIS

We test the proposed multi-release model using software data collected from Tandem computers [14]. 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 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.

In the process of estimation, firstly, the testing resource was estimated using Weibull distribution function and then the parameters of software reliability growth model were estimated for each release using the proposed multi-release modeling framework given in section 3. The parameter values of the proposed model obtained for the four releases are shown in Table 1. The goodness of fit curves of four releases is shown graphically in Figures 1 to 4 respectively. The goodness of fit measures of four releases is shown in Table 2.

Table 1: Parameter Estimates.

	λ	v	γ	a	b	β	p	α
Release 1	11710.76	0.02352	1.46015	103	0.04624	0.96217	0.00408	0.23918
Release 2	11415.79	0.01855	1.66335	121	0.024571	0.841302	0.00620	0.35856
Release 3	5796.178	0.01316	2.068394	63	0.001	1.06711	0.527109	0.09929
Release 4	12011.12	0.00441	2.26179	43	0.007094	0.001	0.02155	0.164741

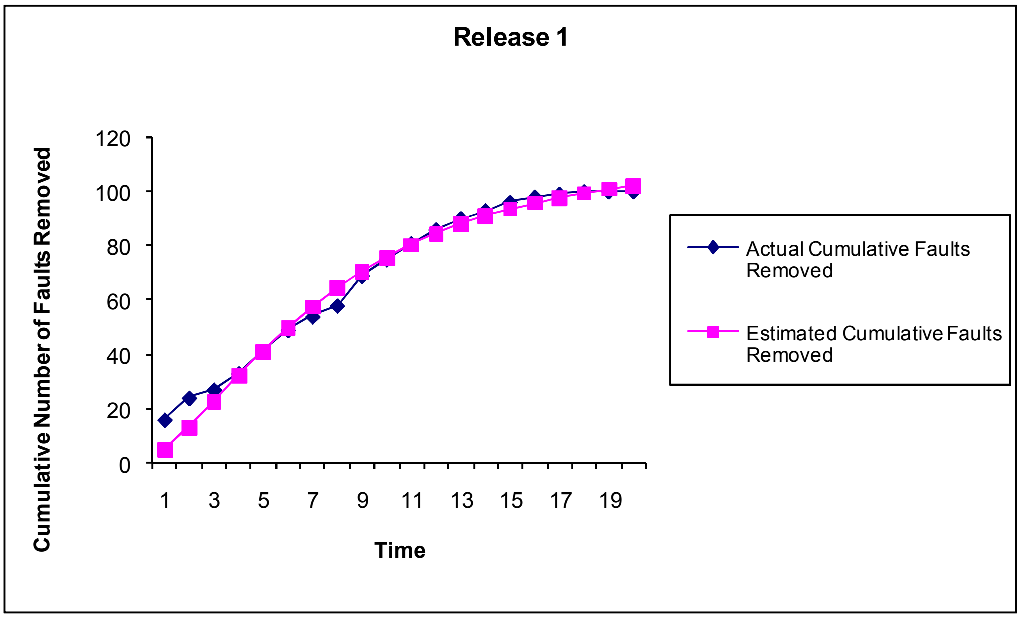

Fig. 1: Goodness of Fit Curve for Release 1.

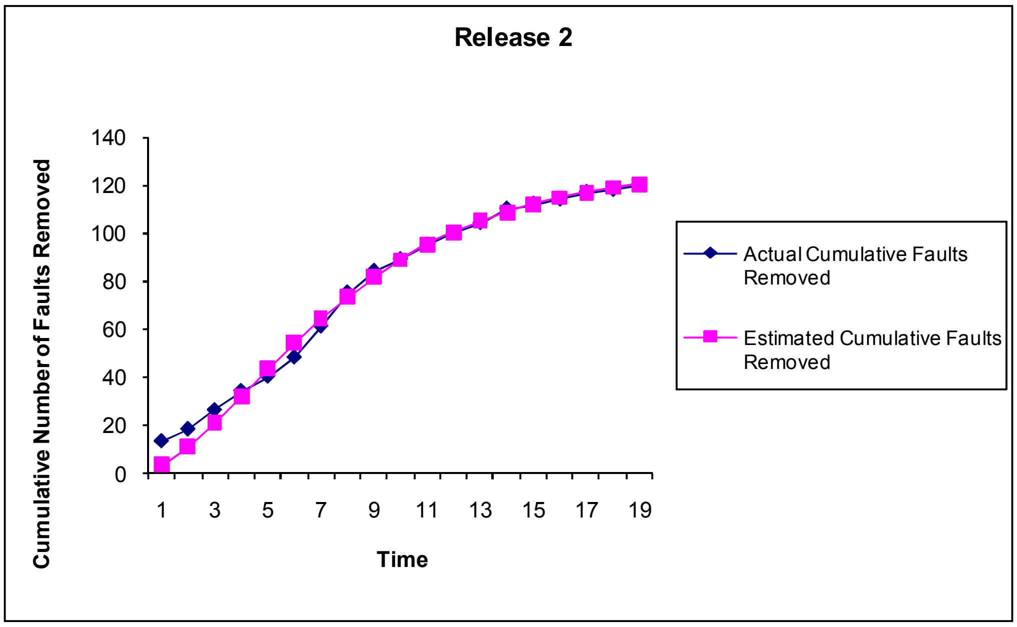

Fig. 2: Goodness of Fit Curve for Release 2.

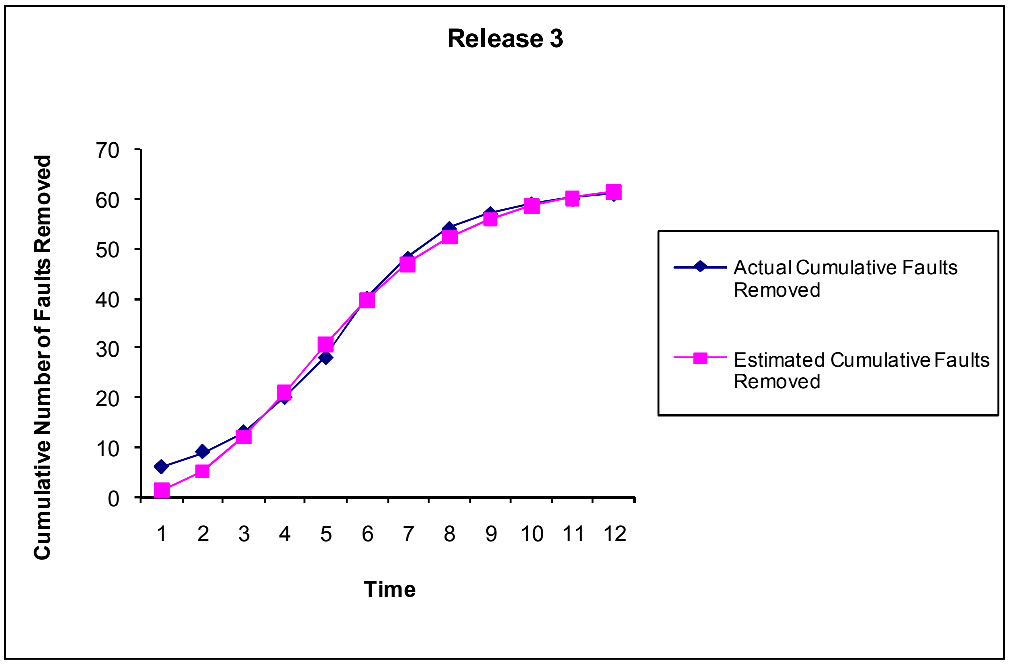

Fig. 3: Goodness of Fit Curve for Release 3.

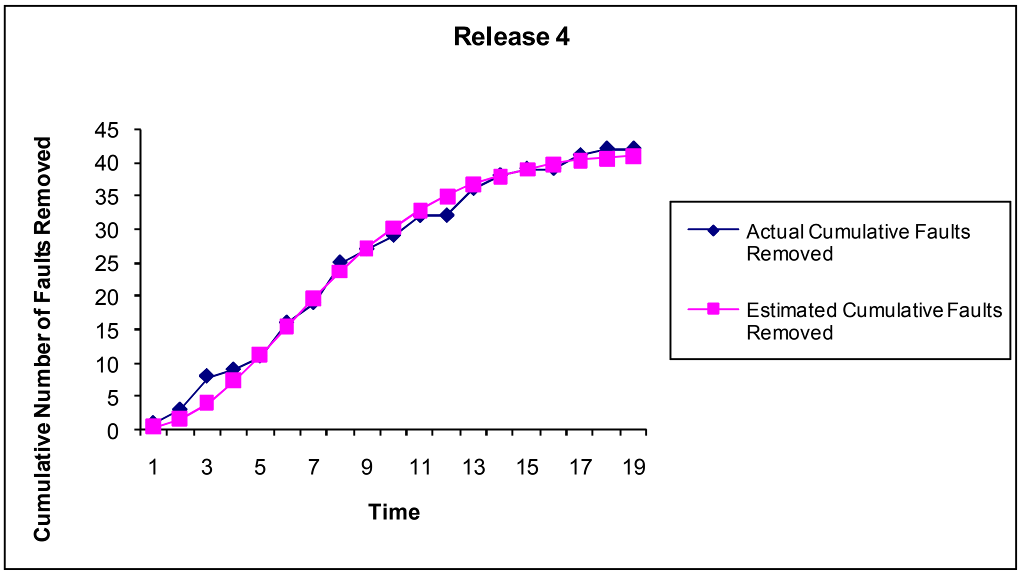

Fig. 4: Goodness of Fit Curve for Release 4.

Table 2: Goodness of Fit Measures.

	Release 1	**Release 2**	**Release 3**	**Release 4**
R Square	0.98	0.991	0.99	0.989
Bias	1.005755	0.465545	0.75837	0.30044
MSE	20.23086	12.6021	4.341648	2.059861
Variance	4.119035	3.615722	2.027051	1.44188

6. CONCLUSION

Nowadays, few software products are launched into the market with their full version during the initial release. Most products start their life cycle with an initial version that includes enough features to make it useful for the customers. Later, they progress as new features are supplemented and existing features are enhanced through a series of releases. Releasing different versions of the product lengthen the market life of product, protect competitive advantages and sustain crucial maintenance revenue streams. But, the matter of fact is that up-gradation of software application is a difficult process. Upgrades in software amplify the fault content. Hence, there is an utmost need to model multi-releases in software development process. In this paper we have proposed a multi release software reliability growth model for N software releases. The proposed model is developed incorporates the effect of testing resource function. The SRGM is flexible in nature in each release as it can describe both exponential and S-shaped curves as demonstrated in this paper. This model also includes the effects of imperfect debugging and error generation.

In this article we have discussed the modeling framework with respect to testing resource function. In future, we can also explore the possibility of including multi dimensional multi release software reliability growth modeling that can take into consideration the effect of not only testing resource but also other testing factors like testing coverage, testing time/number of test cases on the fault removal process simultaneously.

REFERENCES

[1] Martyn Williams; "Software glitch halts Tokyo Stock Exchange", 01 Nov 2005. And "Official: Software glitch, not bomb, shut airport", MSNBC, 20 April 2006.

[2] Lea Rush; "Windows Vista patch ready for download", IT News Digest, 7 Aug, 2007.

[3] A.L. Goel and Kazu Okumoto; “Time-dependent error-detection rate model for software reliability and other performance measures”, IEEE Transactions on Reliability, Vol. R-28(3), pp. 206–211, Aug 1979.

[4] P.K. Kapur, S. Kumar, and R.B. Garg; “Contributions to Hardware and Software Reliability”, World Scientific Publishing Pte. Co. Ltd.: Singapore 1999. ISBN 981-02-3751-0.

[5] H. Pham; “Software Reliability“, Springer-Verlag, Singapore, 2000. DOI: 10.1002/047134608X.W6952

[6] M. Ohba and S. Yamada; “S-shaped Software Reliability Growth Model”, Proceedings of the 4th International Conference on Reliability and Maintainability, pp. 430-436, 1984.

[7] S.Yamada, M. Ohba and S. Osaki; “S-Shaped Software Reliability Growth Modelling for Software Error Detection” IEEE Trans. On Reliability, Vol. R-32(5): pp. 475-484, 1983.

[8] S. Bittanti, P. Bolzern, E. Pedrotti, M. Pozzi and R. Scattolini; “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. DOI: 10.1007/BFb0034288,

[9] P.K. Kapur and R.B. Garg; “A Software Reliability Growth Model for an Error Removal Phenomenon”, Software Engineering Journal, Vol. 7(4), pp. 291-294, 1992.

[10] C.Y. Huang, S.K. Kuo and M.R. Lyu; “An assessment of testing-effort dependent software reliability growth models”, IEEE Trans. Reliability., Vol. 56(2), pp. 198-211, 2007.

[11] P.K. Kapur, D.N. Goswami, A. Bardhan and O. Singh; "Flexible software reliability growth model with testing effort dependent learning process", Applied Math. Modelling, Vol. 32(7), pp. 1298-1307, 2008.

[12] S. Yamada, H. Ohtera and H. Narihisa; "Software Reliability Growth Models with Testing-Effort", IEEE Trans. on Reliability, Vol. 35(1), pp. 19-23, April 1986.

[13] P.K. Kapur, Anu G. Aggarwal, Kanica Kapoor and Gurjeet Kaur; "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(6), pp. 495–508, 2009.

[14] A. Wood; "Predicting Software Reliability", IEEE Computers, Vol. 29(11), pp. 69-77, 1996.