Fault Severity based Multi Up-gradation Modeling considering Testing and Operational Profile

Adarsh Anand
Department of Operational Research
University of Delhi, Delhi 110007

Ompal Singh
Department of Operational Research
University of Delhi, Delhi 110007

Subhrata Das
Department of Operational Research
University of Delhi, Delhi 110007

ABSTRACT
For sustaining in the era of globalization, frequent up-gradation in the software is demanded. In order to maintain the competitive edge firms come up with highly reliable versions. For providing high reliability firms have to test software rigorously that requires debugging the software in the testing as well as in the operational phase. Upgrading the software leads to the enlarged complexity in the system which often results in the increase number of faults content. In this paper, a framework for successive releases is modeled which incorporates the fault of the present release and faults from the just previous release informed by the users. The approach provides a more realistic perspective of fault evaluation by taking into consideration both; the faults from the testing of new releases and the reported faults from the operational phase of the preceding releases. We have examined the case when there exists two types of faults in the software; simple and hard faults during testing and operational phase. Further we have compared our proposition with the previous developed models in the field of multi up-gradation.

Keywords
Fault Severity**,** Multi Release, Operational Phase, Testing Phase.

1. INTRODUCTION
Today, Software is used across the world. It's in everything we do and use. It's in all the sectors like: accounting, banking, education, health care, pharmaceuticals, telecommunication etc. We as consumers interact and buy software more than ever, often without even knowing it. For example, Amazon, Flip kart and many other changed the book selling – and everything selling industry – with software. Airbnb is another example, it is a trusted community marketplace for people to list, discover, and book unique accommodations around the world. Established industries are also evolving to adapt software to improve performance. The automotive market, for example, used more or less no software around 2001. Now we have software working all over our vehicles, from sensors and cameras to navigation. In the financial industry, it's becoming less and less important for consumers to have face to face interactions at their bank because of mobile banking, deposits using your phone, applications that allow you to budget and transfer money.

In the 21st century we seldom see any industry or service organization working without the help of an embedded software system. Such a dependence of mankind on software system has made it necessary to produce the highly reliable software [10]. There are many real life examples when failures in computer systems of safety critical systems have caused spectacular failure resulting in calamitous loss to life and economy. For example, News reports of Asia in July of 2011 reported that software bugs in a national computerized testing and grading system resulted in incorrect test results for tens of thousands of high school students [2, 6]. Due to this very reason the software firms keep on up-grading the software. A better up-gradation can enhance the reliability and characteristics of the system; but at the same time a risky up-gradation can cause errors in the system. For example: In March 2014, automotive manufacturer Tesla addressed a known fire risk in its car by providing a software update to existing vehicles. This helped reduce the risk without owners needing to visit dealerships or service centers. Also, in October of 2013 the U.S. federal government opened a new health insurance exchange web site that, during its first few months of operation, generated major national and worldwide press coverage of its many reported problems. The problems were attributed to, among other things, inadequate time allowed for system testing [2, 6].

In present times, firms are developing software's in multiple releases by improving the existing functionality and revisions, increasing the functionality, or a combination of both. One such example is versions of Android like: Alpha (1.0), Beta (1.1), Cupcake (1.5), Donut (1.6), Eclair (2.0–2.1), Froyo (2.2–2.2.3), Ginger bread (2.3–2.3.7), Honey comb (3.0–3.2.6), Ice Cream Sandwich (4.0–4.0.4), Jelly Bean (4.1–4.3.1), Kit Kat (4.4–4.4.4) and Lollipop (5.0–5.0.1). Thus, up-gradation is a process of adding new features, defects fixes and patches to an application in the form of installer or additions or patch. Additional functionalities may cause fault generation in the system. It is essential to know the content of faults in the software before debugging them. Many researchers have worked on modeling the concept of multi up-gradation including different scenarios that may occur in the system.

Researchers like: Kapur et al. [7] developed a multi up-gradation software reliability model, considering that cumulative faults removed in a particular release depend on all previous releases. Singh et al. [15] assumed 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 functionality to the existing software system. They developed two SRGM's using Logistic distribution and Normal distribution. Kapur et al. [9] proposed a multi release software reliability growth model in which they identified the faults left in the software when it is in operational phase during the testing of the new code incorporating that the software includes different types of faults. Anand et al. [2] incorporated the generalised framework for faults in new release due to up-gradation of the features and undetected faults from operational phase of preceding releases and different distributions have used for fault removal phenomenon.

In the proposed model, we have modeled successive release of software which incorporates the fault of the present release and faults from the just previous release informed by the users. Here we have accentuated on the fact that the software has two types of faults i.e. simple and hard faults. Simple faults are those which require less effort to remove from the system. Whereas, hard faults require more time to remove from the system.The simple faults interact with new portion of simple detected faults in both testing and operational phase. The hard faults too interact with the new portion of the hard detected faults in both testing and operational phase. It has been observed that a large number of simple faults are easily detected at the beginning of testing; on the other hand fault removal becomes a tedious task in the later stages.

Therefore in this article, we consider two different fault detection rates, we assume that the simple faults are removed exponentially [5] and hard faults are removed using a two stage fault removal phenomenon i.e. using Yamada function [16]. Further we compare our proposed model with work done by other researchers in the field of multi up-gradation such as: Singh et al. [15], Kapur et al. [9]. Outline of the paper is organized as follows: Section 2 indicates notations and Section 3 presents an overview of software development life cycle. Section 4 shows the modeling framework. Finally data analysis, comparison of models and conclusions are supplemented in Section 5 and Section 6 respectively.

2. NOTATIONS

$m(t)$	Number of faults removed by time 't'
$F(t)$	Probability distributions function for fault removal phenomena.
a	Total number of faults in the software.
a_n	Initial fault content for nth release (n=1 to 4).
b_i	Fault detection rate function (i=1 to 4).
k	Shape parameter.
λ	Proportion of simple faults in the software.
$1-\lambda$	Proportion of hard faults in the software.
γ	Proportion of undetected faults removed in testing phase.
$1-\gamma$	Proportion of undetected faults removed in operational phase.

3. IMPORTANCE OF OPERATIONAL PHASE IN SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Software development life cycle (SDLC) is an important factor in the development of software. It is a ruminate, structured and methodological way to pursue the enhancement of software system, considering each stage of the life cycle from beginning of the idea to release of the final system. SDLC has five following phases: analysis, design, coding, testing and operational phase. Analysis phase includes the investigation of the requirements of user and scope of the product which is generally done by the software developers and management team through market research. The next step is to create a high level design document containing time frame to create project, required number of resources, and technology and language details, by the technical team. Coding phase reviews the analyzed document and designed document is done by the software developers. After the review, coding is done on the decided technologies to obtain the project structure. Then the solution is submitted to the testing team for further analysis.

Testing phase is a noteworthy phase of a software development life cycle. The main purpose of this phase is to remove faults with minimum cost in order to achieve certain desired level of reliability. As mentioned in the article [2], Software testing is an exploration conducted to provide developers with information about the quality of the product. Testing of software starts, when the code has been generated by the programmers. Testing is generally focused on validation and verification of the system. Validation confirms that the product actually meets the user's needs. Verification confirms that whether the product has been built according to design specification and requirements. Once the product is tested then it is ready to release in the market. Operational phase provides continuous support for the system in the form of troubleshooting, creating performance reviews and assessments, and ensuring that the solution's security remains intact [2]. The operational phase starts after verification and acceptance of the system by the client. In this phase, any change made to software, either to correct a deficiency in its performance to compensate for environmental change, or to enhance its operation [11]. If any fault has been identified from developer side, it get removed immediately without requiring any extra resources but if it has been found from customer's side it took extra cost, efforts and time. Sometime it also affects the goodwill of the software companies. Software is called to be in operational phase when it is delivered to the customer for their usage. When the software is at the user end, it does not means that it is not being tested rather the firms are adding new functionalities to existing code and rigorously testing it.

4. MODELING FRAMEWORK

In order to survive in the market with globalized competition, software companies assure to come up with innovative functionalities/features in its each new release of their merchandise. The addition of new functionalities often leads to increase in the content of bugs. These bugs are reported by the users in the current release of the operational phase and this is how information is transferred to the companies and they get to know about it. Based on the information received, firms introduce new functions and fix the defects in order to increase the effectiveness of the product. There may be chances that some of the bugs in the previous release are removed directly by the testing team of existing release and some are removed in the operational phase. On the basis of severity, faults are distinguished into two types: simple and hard faults. In this article, we use exponential distribution for simple faults [5] and two stage yamada distributions for hard faults in testing phase [16]. Here, we discuss a mathematical expression for successive releases of software reliability growth model. Let us assume that the software's first release is done at $t = \tau_1$.

Due to complexity, all faults cannot be removed in the first release of the software and therefore, some faults remain in code [9]. It is important to know that how many faults exist in the software at any time, so that different testing strategy and testing effort can be applied to remove those faults [4]. The mathematical expression of total faults in first release of software is given as:

$$m_1(t) = \lambda_1 a_1 . F_{11}(t) + (1-\lambda_1) . a_1 . F_{12}(t); (\tau_1 = 0) < t < \tau_2 \qquad (1)$$

In competitive scenario the major concern of the firm is to find out the new functionalities that are to be added to increase their popularity among the customers. The firms add new functionalities on the basis of customer's feedback and the left over faults which are not removed before the release but are reported from the users in the operational phase. Now the

simple faults interact with new portion of simple detected faults in both testing and operational phase. And also the hard faults interact with the new portion of the hard detected faults in both testing and operational phase. When the firms are ready to launch the second version of software, they have to keep in consideration some aspects. In the existing release, firm has to remove some leftover faults and new generated faults due to improvement of the software.

Modeling for second release is given as:

$$\begin{aligned} m_2(t) &= \lambda_2.a_2.F_{21}^T(t-\tau_2)+(1-\lambda_2).a_2.F_{22}^T(t-\tau_2) \\ &+\gamma_1.\lambda_1.a_1\left(1-F_{11}(\tau_2)\right).F_{21}^T(t-\tau_2) \\ &+(1-\gamma_1).\lambda_1.a_1\left(1-F_{11}(\tau_2)\right).F_{11}^O(t-\tau_2) \\ &+\gamma_2.(1-\lambda_1).a_1\left(1-F_{12}(\tau_2)\right).F_{22}^T(t-\tau_2) \\ &+(1-\gamma_2).(1-\lambda_1).a_1\left(1-F_{12}(\tau_2)\right).F_{12}^O(t-\tau_2);\ \tau_2<t<\tau_3 \end{aligned} \quad (2)$$

where $F_{21}^T=\left(1-e^{-b_1 t}\right)$

$$F_{22}^T=\left(1-(1+b_2 t)e^{-b_2 t}\right)$$

$$F_{11}^O=\left(1-e^{-b_3 t^{k_1}}\right), F_{12}^O=\left(1-e^{-b_4 t^{k_2}}\right)$$

Now for both simple and hard faults in operational phase, we use weibull distribution. Here $\gamma_1.\lambda_1.a_1.\left(1-F_{11}(\tau_2)\right)$ and $\gamma_2.(1-\lambda_1).a_1.\left(1-F_{12}(\tau_2)\right)$ represent the undetected simple and hard faults of first release during testing phase which interacts with new portion of code of second release. Leftover faults are removed by new detection rate of second release, which are $F_{21}^T(t-\tau_2)$ and $F_{22}^T(t-\tau_2)$. $(1-\gamma_1).\lambda_1.a_1\left(1-F_{11}(\tau_2)\right)$ and $(1-\gamma_2).(1-\lambda_1).a_1\left(1-F_{12}(\tau_2)\right)$ demonstrate fault content of simple and hard faults during operational phase of first release, which interact with new detection rate i.e. $F_{11}^O(t-\tau_2)$ and $F_{12}^O(t-\tau_2)$.

Similarly for n^{th} release, we can write:

$$\begin{aligned} m_n(t) &= \lambda_n.a_n.F_{n1}^T(t-\tau_n)+(1-\lambda_n).a_2.F_{n2}^T(t-\tau_n) \\ &+\gamma_n.\lambda_{n-1}.a_{n-1}\left(1-F_{n-1,1}(\tau_n-\tau_{n-1})\right).F_{n1}^T(t-\tau_n) \\ &+(1-\gamma_n).\lambda_{n-1}.a_{n-1}\left(1-F_{n-1,1}(\tau_n-\tau_{n-1})\right).F_{n-1,1}^O(t-\tau_n) \\ &+\gamma_{n+1}.(1-\lambda_{n-1}).a_{n-1}\left(1-F_{n-1,2}(\tau_n-\tau_{n-1})\right).F_{n,2}^T(t-\tau_n) \\ &+(1-\gamma_{n+1}).(1-\lambda_{n-1}).a_{n-1}\left(1-F_{n-1,2}(\tau_n-\tau_{n-1})\right). \\ &F_{n-1,2}^O(t-\tau_n) \end{aligned} \quad (3)$$

In earlier time the analysis for multi up-gradation in software reliability was concerned in determining the faults based on the impact of all previous releases of the software [7]. In later hours, researchers have extended the concept of multi up-gradation to account for the number of faults from just previous release [15]. This methodology of modeling was based on the concept that the faults of first release which were not removed in second release would not cause any system failure in the later releases and also they will be quite less in number. There are certain conditions in which the testing team was not able to fix the bugs perfectly. To account for this kind of situations, Kapur et al [8] have extended the concept of fault removal process under the just previous release criteria to consider the case of imperfect debugging. The various factors that influence the testing progress are testing effort expenditure, testing efficiency and skills, which may not be deterministic in nature [10, 13]. In order to capture the uncertainty of the testing process, Singh et al. [13] proposed a multi up-gradation software reliability model incorporating stochastic differential equations. Further the classification of faults was incorporated in which faults were categorized into simple and hard faults based on the time they require to be removed. Kapur et al. [9] has also proposed a concept of fault severity into multi up-gradation software reliability model under the just previous release criteria. Few researchers have incorporated the concept of imperfect debugging and stochastic differential equation into fault severity; to model more realistic scenarios occurring in the field [1, 14]. Recently Garmabaki et al. [4] developed a successive software release model in which they focus on bugs reported from operational phase. The consideration of fault severity into the modeling of faults reported from operational profile has not been given importance in determining the fault content. To capture this process we have developed a modeling framework for incorporating the fault severity in testing as well as in operational phase of SDLC.

There are different scenarios for determining the count of faults that were removed in each successive release. For this very reason we compare the proposed approach with few established approaches and to have a deeper insight about the proposed methodology which considers both the faults from testing and operational phase. Table 1 shows how our analysis is significant over the research done earlier in the same field.

Table 1: Comparison between proposed approach and prior research

	Proposed approach	**Singh et al [15]**	**Kapur et al [9]**	**Garmabaki et al. [4]**	**Kapur et al[8]**	**Singh et al [13]**
Current release depending upon just previous release	Yes	Yes	Yes	Yes	Yes	Yes
Impact of operational phase	Yes	No	No	Yes	No	No
Impact of fault severity	Yes	No	Yes	No	No	No

5. DATA ANALYSIS AND COMPARISON CRITERIA

The performance of the model has been analyzed by using real software failure data. The data set presented the failure data for four major releases of software product on Tandem computers [17]. Table 2 illustrates the estimated values of parameters of each of the four releases. Table 3 interprets the comparison criteria of the four software releases. Figure 1 to 4 shows the estimated and actual values of the number of faults removed in different releases. For the applicability of the methodology we have compared proposed model with Singh et al. [15] as SRGM-I and, Kapur et al. [9] as SRGM-II.

Singh et al. [15] developed a model in which there is confusion in the performance of release 3 and release 4, which is given in Table 4. If we see the value of MSE for release 3 and release 4 are 1.8529 and 1.0077, which shows that release 4 gives better value. On the other side BIAS value of release 3 is -0.1011 and release 4 is -0.11854, here release 3 gives better value. The approach of Kapur et al. [9] shows that release 3 performs better, which is given in Table 5. Then question arises why there is a need to upgrade to next version. As the name suggests multi up-gradation means to upgrade the software to capture the user's requirement, add new functionalities, improve the already existing version. It is also important that upgraded version is able to fulfill the expectations that are arising at the customers end. The earlier models in this pasture were not able to provide a clear picture about the concept of up-gradation. There are different methodologies such as distance based approach [12], weighted criteria approach [3] etc. available to the software developing firms; on the basis of which they can optimally select the best version. Without using any specific criteria, the approach presented in this article is indeed able to predict the upgraded version performs better. This approach is able to satisfy the clear meaning of multi up-gradation. Table 2 demonstrates the comparison criteria of the proposed model, in which it can be clearly seen that release 4 gives the lower value of MSE, BIAS, VARIATION, and RMSPE as compared to other releases. It means that up-gradation results into better version of the software.

Table 2: Parameter Estimation

Parameters	a	b_1	b_2	b_3	b_4	λ	γ_1	γ_2	k_1	k_2
Release 1	120.022	0.101	0.187	-	-	0.785	-	-	-	-
Release 2	136.096	0.068	0.225	0.0192	0.023	0.621	0.45	0.512	1.996	1.913
Release 3	65	0.043	0.3	0.002	0.002	0.424	0.52	0.53	3.516	3.516
Release 4	45.454	0.025	0.207	0.001	0.031	0.552	0.542	0.681	3.389	1.162

Table 3: Comparison criteria for proposed approach

Comparisons	Release 1	Release 2	Release 3	Release 4
R^2	0.988	0.992	0.990	0.996
MSE	10.15732	10.45623	4.366086	0.832707
BIAS	0.256244	0.099926	0.057641	0.002596
VARIATION	3.259263	3.320632	2.182431	0.937529
RMSPE	3.269321	3.322135	2.183192	0.937532913

Table 4: Comparison criteria for just preceding release SRGM-I

Comparisons	Release 1	Release 2	Release 3	Release 4
R^2	0.989	0.995	0.993	0.995
MSE	3.0471	2.4925	1.8529	1.0077
BIAS	0.435	0.340	-0.1011	-0.11854
VARIATION	8.979	6.001	3.1547	0.9774
RMSPE	3.0727	2.5156	1.85569	1.0146

Table 5: Comparison criteria for just preceding release SRGM-II

Comparisons	Release 1	Release 2	Release 3	Release 4
R^2	0.996	0.997	0.999	0.995
MSE	2.658	3.209	0.532	0.992
BIAS	0.190	0.129	0.158	-0.059
VARIATION	1.666	1.835	0.744	0.984
RMSPE	1.677	1.840	0.760	0.986

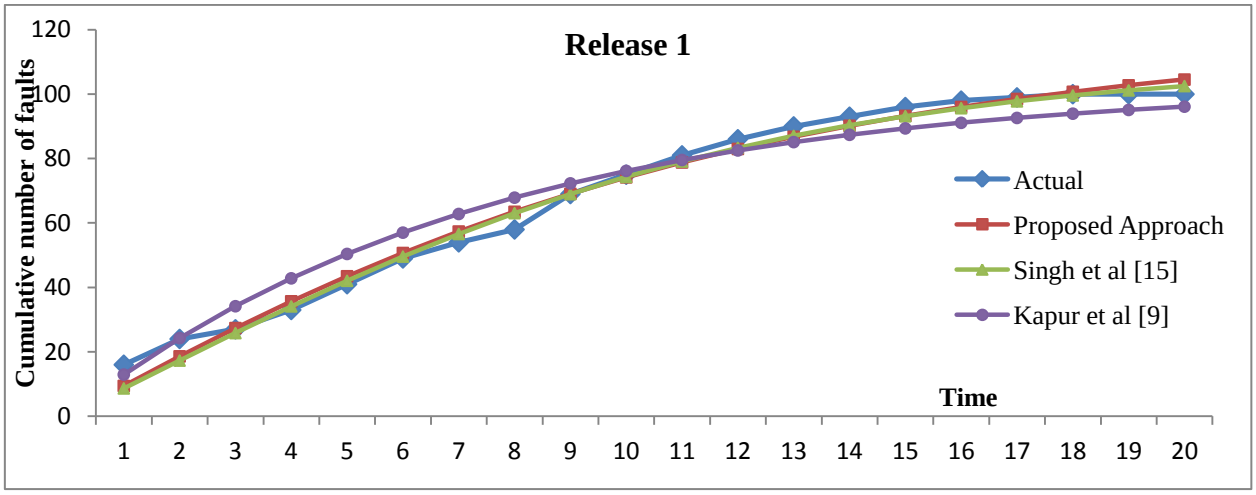

Fig 1: Goodness of Fit curve for Release 1

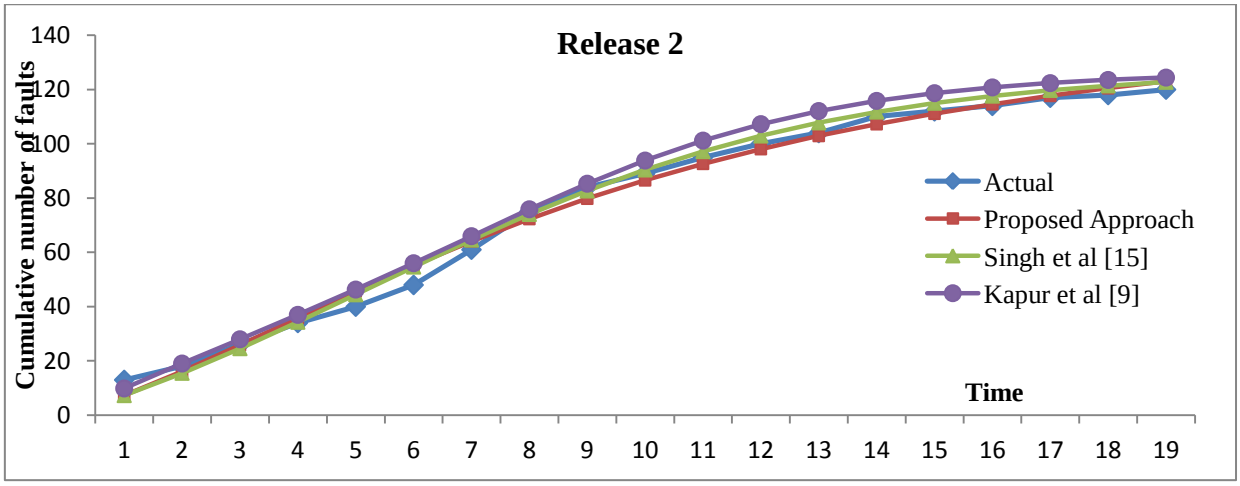

Fig 2: Goodness of Fit curve for Release 2

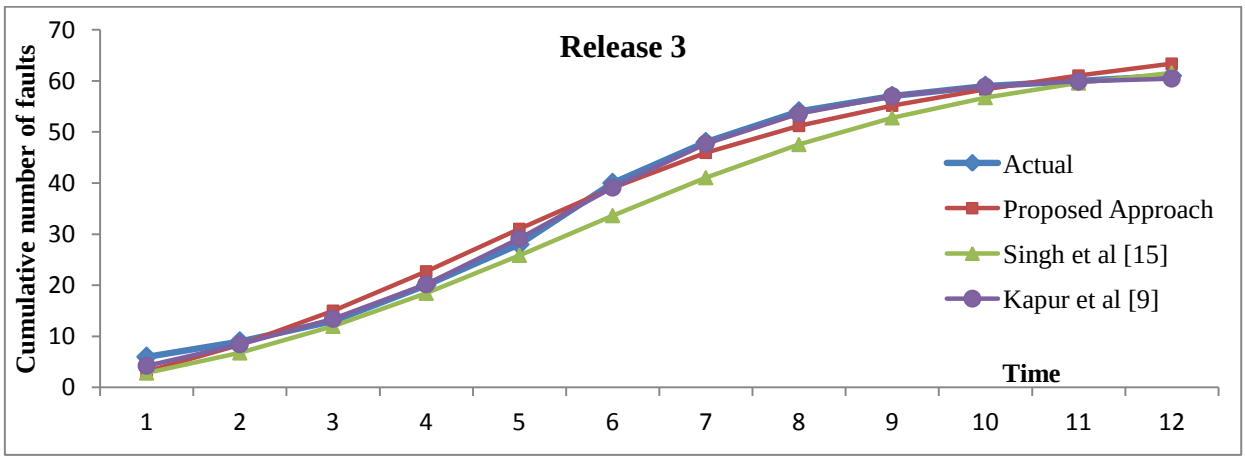

Fig 3: Goodness of Fit curve for Release 3

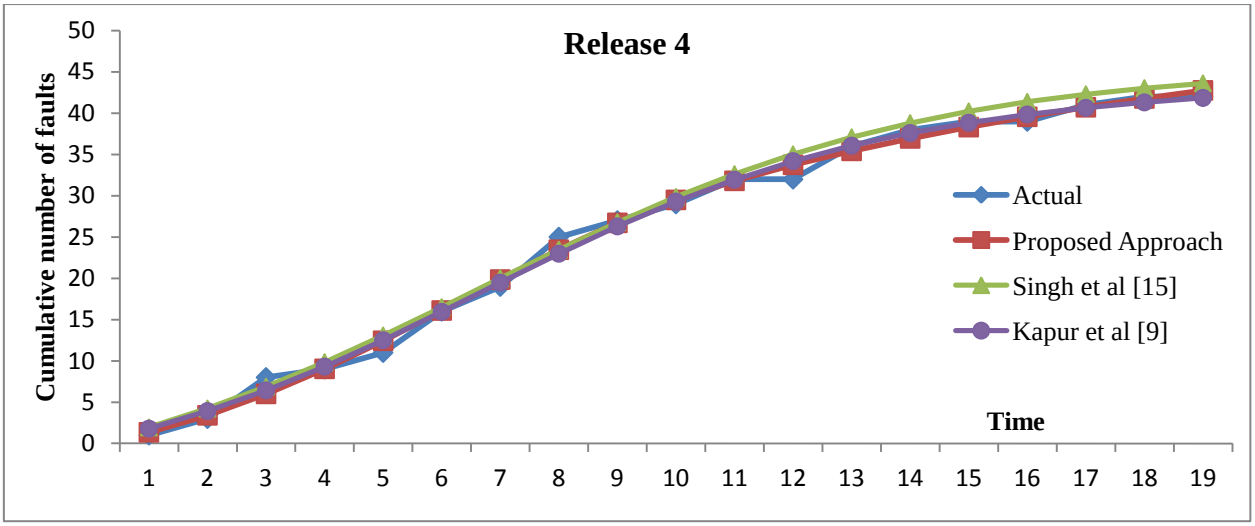

Fig 4: Goodness of Fit curve for Release 4

Figure 1 to 4 represent the goodness of fit curve between the proposed approach; and the approach given by Singh et al [15] and Kapur et al [9]. It shows that the deviation between the actual and predicted values have a decreasing trend from release 1 to 4; and the methodology presented here is able to capture the fault removal phenomenon in a more appropriate manner compared to the framework of Singh et [15] and Kapur et al [9]. Hence we can also interpret with the help of these representations that Figure 4 (release 4) has more significant goodness of fit curve for the proposed methodology.

6. CONCLUSION

In the present time, the firms are competing with each other and like any other product; software products too face a tough competition. Quality of the software is an important attribute to survive in this cut-throat competition. In this process of deciding about the quality, reliability is the pivotal factor. Providing high quality leads to successive releases of the previous version of the software. The proposed model is based on the fact that total fault elimination in the software of the new release is judged on the basis of the reported bugs of the previous release and it also takes into consideration the faults generated due to additional features. Also, we have discussed a multi up-gradation software model which incorporates severity of faults, in which the simple faults interact with new portion of detected simple faults in both testing and operational phase, whereas, the hard faults interact with the new portion of the detected hard faults in both testing and operational phase. Further, we have compared our proposed model with the previous work done by researchers and it conclude that the successive version provides better quality as can be seen from the results supplemented in Table-3. Also further the incorporation of testing effort, imperfect debugging and error generation can be studied.

7. ACKNOWLEDGMENTS

The research work presented in this paper is supported by grants to the first and second author from Department of Science and Technology (DST) via DST PURSE phase II grant, India.

8. REFERENCES

[1] Aggarwal, A.G., Kapur, P.K., and Garmabaki, A.S. 2011. Imperfect Debugging Software Reliability Growth Model for Multiple Releases, Proceedings of the 5th National Conference on Computing for Nation Development-INDIACOM, New Delhi.

[2] Anand, A., Singh, A., Kapur, P. K., and Das, S. 2014. 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.

[3] Cuong, N. H., Thang, H. Q., and Trieu, L. H. 2014. Different Ranking of NHPP Software Reliability Growth Models with Generalised Measure and Predictability, International Journal of Applied Information Systems, 7(11).

[4] Garmabaki A. H. S., Aggarwal A.G., Kapur P. K., and Yadavali V. S. S., "The Impact of Bugs Reported from Operational Phase on Successive Software Releases", International Journal of Productivity and Quality Management, 2014, volume 14, number 4, pp 423-440.

[5] Goel, A.L., and Okumoto, K. 1979. Time-dependent Error Detection Rate Model for Software Reliability and other Performance Measures, IEEE Trans. on Reliability, vol. 28, no. 3, pp. 206–211.

[6] Hower, R. What are some recent major computer system failures caused by software bugs? http://www.softwareqatest.com/qatfaq1.html, (February 2015).

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

[8] Kapur P. K., Singh O., Garmabaki A. and Singh J., "Multi up-gradation software reliability growth model with imperfect debugging", International Journal of systems Assurance Engineering and Management, 2010, 1(4), 299-306.

[9] Kapur, P. K., Anand, A., and Singh, O. 2011. 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.

[10] Kapur, P.K., Pham, H., Gupta, A., and Jha, P.C. 2011. Software reliability Assessment with OR application, Springer London.

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

[12] Sharma, K., Garg, R., Nagpal, C. K. and Garg, R. K. 2010. Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach, IEEE Transactions on Reliability, vol. 59(2).

[13] Singh, O., Kapur, P.K., Anand, A. and Singh, J. 2009. Stochastic Differential Equation based Modeling for Multiple Generations of Software, Proceedings of Fourth International Conference on Quality, Reliability and Infocom Technology (ICQRIT), Trends and Future Directions, Narosa Publications, pp. 122-131.

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

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

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

[17] Wood, A. 1996. Predicting Software Reliability. IEEE Computer (11) 69-77.