MIS Review Vol. 20, No. 1, September (2014), pp. 45-58
DOI: 10.6131/MISR.2014.2001.03

A Discrete Formulation of Successive Software Releases Based on Imperfect Debugging

Jagvinder Singh[1], Adarsh Anand[2], Avneesh Kumar[3], Sunil Kumar Khatri[4]
[1]*Maharaja Agrasen College, University of Delhi, India*
[2]*Department of Operational Research, University of Delhi, India*
[3]*Integrated Academy of Managment Technology, MTU, India*
[4]*Amity Institute of Information Technology, Amity University Uttar Pradesh, India*

ABSTRACT: *Software reliability is the major dynamic attribute of the software quality, so gaining reliability of software product is a vital issue for software products. Due to intense competition the software companies are coming with multiple add-ons to survive in the pure competitive environment by keeping an eye on existing system i.e. system in operational phase. Software reliability engineering is focused on engineering techniques for timely add-ons/Up-gradations and maintaining software systems whose reliability can be quantitatively evaluated. In order to estimate as well as to predict the reliability of software systems, failure data need to be properly measured by various means during software development and operational phases. Although software reliability has remained an active research subject over the past 35 years, challenges and open questions still exist. This paper presents a discrete software reliability growth modeling framework for multi-up gradations including the concept of two types of imperfect debugging during software fault removal phenomenon. The Proposed model has been validated on real data set and provides fairly good results.*

KEYWORDS: *Software Reliability, Non-Homogeneous Poisson Process (NHPP), Software Testing, Successive Software Releases, Imperfect Debugging.*

1. Introduction

The Computer systems now pervade every aspect of our daily lives. While this has benefited society and increased our productivity, it has also made our lives more critically dependent on their correct functioning. Software reliability assessment is important to evaluate and predict the reliability and performance of software system. Several SRGMs have been developed in the literature to estimate the fault content and fault removal rate per fault in software. Goel and Okumoto (1979) have proposed NHPP based SRGM assuming that the failure intensity is proportional to the number of faults remaining in the software. The model is very simple and can describe exponential failure curves. Ohba (1984) refined the Goel-Okumoto model by assuming that the fault detection / removal

rate increases with time and that there are two types of faults in the software. SRGM proposed by Bittanti et al. (1988) and Kapur and Garg (1992) have similar forms as that of Ohba (1984) but are developed under different set of assumptions. Bittanti et al. (1988) proposed an SRGM exploiting the fault removal (exposure) rate during the initial and final time epochs of testing.

Kapur and Garg (1992) described a fault removal phenomenon, where they have assumed that during a removal process of a fault some of the remaining faults may also be removed. These models can describe both exponential and S-shaped growth curves and hence are termed as flexible models.

NHPP based SRGMs are generally classified into two groups. The first group contains models, which use the execution time (i.e., CPU time) or calendar time. Such models are called continuous time models. The second group contains models, which use the test cases as a unit of fault removal period. Such models are called discrete time models, since the unit of software fault removal period is countable (Kapur & Garg, 1999; Kapur et al., 2011; Musa, Iannino & Okumoto, 1987; Pham, 2006; Yamada, Ohba & Osaki, 1984). A test case can be a single computer test run executed in an hour, day, week or even month. Therefore, it includes the computer test run and length of time spent to visually inspect the software source code. A large number of models have been developed in the first group while fewer are there in the second group due to the difficulties in terms of mathematical complexity involved.

The utility of discrete reliability growth models cannot be under estimated. As the software failure data sets are discrete, these models many times provide better fit than their continuous time counterparts. Therefore, in spite of difficulties in terms of mathematical complexity involved, discrete models are proposed regularly. Most of discrete models discussed in the literature seldom differentiate between the failure observation and fault removal processes. In real software development scene, the number of failure observed can be less than or more than the number of error removed. Kapur and Garg (1992) has discussed the first case in their Error removal phenomenon flexible model which shows as the testing grows and testing team gain experience, additional number of faults are removed without them causing any failure. But if the number of failure observed is more than the number of error removed then we are having the case of imperfect debugging. Due to the complexity of the software system and the incomplete understanding of the software requirements, specifications and structure, the testing team may not be able to remove the fault perfectly on the detection of the failure and the original fault may remain or replaced by another fault. While the first phenomenon is known as imperfect debugging, the second is called fault generation (Kapur et al., 2011; Kapur, Singh, et al., 2010; Pham, 2006). In case of imperfect debugging the fault content of the software is not

changed, but because of incomplete understanding of the software, the detected fault is not removed completely. But in case of error generation the fault content increases as the testing progresses and removal results in introduction of new faults while removing old ones.

The concept of imperfect debugging was first introduced by Goel (1985). He introduced the probability of imperfect debugging in Jelinski and Moranda (1972). Kapur, Garg and Kumar (1999) and Kapur et al. (2011) introduced the imperfect debugging in Goel and Okumoto (1979). They assumed that the FRR per remaining faults is reduced due to imperfect debugging. Thus the number of failures observed by time infinity is more than the initial fault content. Although these two models describe the imperfect debugging phenomenon yet the software reliability growth curve of these models is always exponential. Moreover, they assume that the probability of imperfect debugging is independent of the testing time. Thus, they ignore the role of the learning process during the testing phase by not accounting for the experience gained with the progress of software testing. All these models are continuous time models. Kapur, Singh, et al. (2010) and Kapur, Tandon and Kaur (2010) have proposed three discrete models taking into account imperfect fault debugging and fault generation phenomena separately. But even that framework was restricted to single release of the software. Overcoming this, Kapur, Singh, et al. (2010) and Kapur, Tandon, et al. (2010) developed many multi release models but they were formulated in continuous time framework. In this paper, a general discrete SRGM for multi releases incorporating fault generation and imperfect debugging with learning has been proposed.

2. Multi up-gradation of software

The present software development environment is very competitive and advanced. Many independent and well established developing organizations are competing in the market with similar products and capabilities to attain the maximum market share and brand value. As such software delivered with full functionalities and very high reliability built over a period of time may turn out to be unsuccessful due to technological obsolescence. Therefore now a day's the software are rather developed in multiple releases where the latest releases might be developed by improving the existing functionality and revisions, increasing the functionality, a combination of both or improving the quality of the software in terms of reliability etc. (Kapur, Singh, et al., 2010; Kapur, Tandon, et al., 2010). For example we can see the various software in the market named as Windows 98, Windows 2000, Windows ME, Windows XP, Windows Vista, Windows 7 etc. For another illustration consider a development firm developing antivirus software. Such a firm can begin with releasing the product that detects and remove viruses and spywares from

the computer system. In their second release they can provide the feature of protecting the system from virus infected emails. Next, they can add the trait of blocking spyware automatically for the next release. Finally, the fourth release can provide the root kit protection along with removing hidden threats in the operating system.

This step by step release (base software with features enhancement) is advantageous for the developing firms in various contexts. Firstly, if a firm implements the complete characteristic capabilities in first release, than that would delay the product release in the market in the desired time window. Secondly, launching of new software product may bring the developing firm in limelight, but the stream of subsequent product releases is the source of their bread and butter. Moreover releasing different versions of the product lengthen the market life of product, protect competitive advantages and sustain crucial maintenance revenue streams.

Software products aren't static and each release has a limited existence. As soon as a software product reaches the market, a variety of factors begin creating demand for changes (Figure 1). Defects require repairs. Competitors offer software with added features and functions. Evolving technology requires upgrades to support new hardware and updated versions of operating software. Sales demands new features to win over prospects. Customers want new functionality to justify maintenance expenditures. These demands accumulate over time, eventually reaching the point when the software product must be upgraded with a new version to remain viable in the market. As soon as the new version is released, the cycle begins again.

For software developing organizations it is not an easy task to design software in isolation. Developing reliable software is one of the most difficult problems faced by them. Timely development, resource limitations, and unrealistic requirements can all negatively impact software reliability. Moreover, there is some interdependence between their developments. The interdependence between their developments exists in many

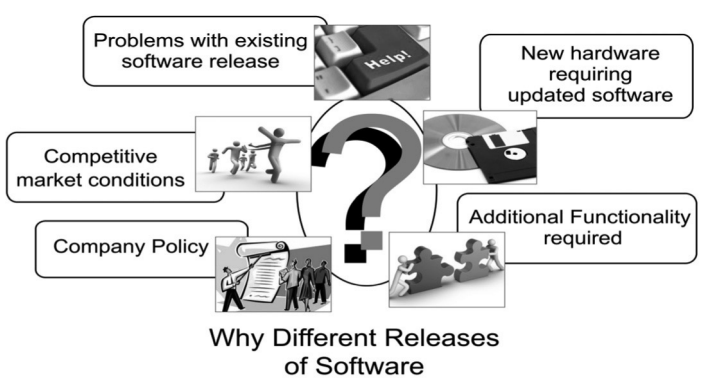

Figure 1 Need for Different Releases

ways, which also affects their reliability. A new release (an upgraded version of the base software) may come into existence even during its development, at the time of release or during its operation. The code and other documents related to a release may be some modification of the existing code and documents and/or addition of new modules and related modification in the documents. The dependence of the development process of successive releases necessitates considering this dependence in the reliability growth analysis.

Kapur, Tandon, et al. (2010) developed multi up-gradation reliability model. In this paper, the fault removal process for multiple releases is modeled with respect to testing time (CPU time). This is a continuous time model. But in real life situations most of the data is collected at discrete time instants so there arises a need for the modeling framework which is also discrete in nature. The discrete models relate fault removal process to either number of test cases executed or number of testing weeks etc. These models many a times provide a better fit as compared to continuous time models. Recently, Kapur, Aggarwal and Nijhwan (2014) have developed a modeling framework for multi up-gradations in discrete environment. But they have considered the fault removal process to be depending on all the previous releases. In the proposed model the fault removal process is related to the number of testing periods. The assumption of fault removal process to be depending on all the previous releases has been relaxed and we have considered the dependency on just previous release. Furthermore, 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. This paper develops a mathematical relationship between features enhancement and software faults removal process incorporating the aforesaid concepts of imperfect debugging. The model is developed for four 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 new 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 has been organized as follows: Section 3 provides the basic Structure for single release of software which is also the framework for modeling multiple releases; Section 4 contains the parameter estimation values along with the data description.

3. Software reliability modelling for single release: framework for multi-releases

3.1 Model development

During debugging process faults are identified and removed upon failures. In reality this may not be always true. The corrections may themselves introduce new faults or they may inadvertently create conditions, not previously experienced, that enable other faults to cause failures. This results in situations where the actual fault removals are less than the removal attempts. Therefore, the FRR is reduced by the probability of imperfect debugging. Besides, there is a good chance that some new faults might be introduced during the course of debugging (Kapur et al., 2011; Pham, 2006; Yamada et al., 1984).

3.2 Assumptions

The developed below is based upon the following basic assumptions:

(1) Failure observation / fault removal phenomenon is modeled by NHPP.

(2) Software is subject to failures during execution caused by faults remaining in the software.

(3) Each time a failure is observed, an immediate effort takes place to decide the cause of the failure in order to remove it.

(4) Failure rate is equally affected by faults remaining in the software.

(5) The debugging process is imperfect.

3.3 Notations

a : Initial Fault content of the software $\sum_{i=1}^{4} a_i = a$.

$a(n)$: Total fault content of the software dependent on the number of testing periods.

b_i : Proportionality constant.

$m_i(n)$: Mean number of faults removed by n number of testing periods.

F_i : Probability distribution function for the number of testing periods.

p_i : The probability of fault removal on a failure (i.e., the probability of perfect debugging).

α_i : The rate at which the faults may be introduced during the debugging process per detected fault.

In all the above notations, i = release 1 to 4.

3.4 Formulation

The software testing phase a software system is executed with a sample of test cases to detect and correct software faults, which cause failures. A discrete counting process $[N(n), n \geq 0]$, $(n = 0, 1, 2, ...)$ is said to be an NHPP with mean value function $m(n)$, if it satisfies the following conditions (Kapur et al., 2011):

There are no failures experienced at $n = 0$, that is, $N(0) = 0$.

The counting process has independent increments, that is, the number of failures experienced during $(n, n+1)^{th}$ testing period is independent of the history and implies that $m(n+1)$ of the process depends only on the present state $m(n)$ and is independent of its past state $m(n)$, for $x < n$.

In other words, for any collection of the numbers of testing periods $n_1, n_2, ..., n_k$ $(0 < n_1 < n_2 < ... < n_k)$ the k random variables $N(n_1), N(n_2) - N(n_1), ..., N(n_k) - N(n_{k-1})$ are statistically independent.

For any of two numbers of test cases n_i and n_j where $(0 \leq n_i \leq n_j)$, we have:

$$\Pr\{N(n_j) - N(n_i) = x\} = \frac{\{m(n_j) - m(n_i)\}^x}{x!} \exp\left[-\{m(n_j) - m(n_i)\}\right] \quad (1)$$

The mean value function $m(n)$ which is a non-decreasing in n represents the expected cumulative number of faults detected by n testing periods. Then the NHPP model with $m(n)$ is formulated by:

$$\Pr\{N(n) = x\} = \frac{\{m(n)\}^x}{x!} \exp\left[-\{m(n)\}\right] \quad (2)$$

Therefore, under the above assumptions, the expected cumulative number of faults removed between the n^{th} and $(n+1)^{th}$ testing period is proportional to the number of faults remaining after the execution n^{th} test run, satisfies the following difference equation

$$m(n+1) - m(n) = bp(a(n) - m(n)) \quad (3)$$

where an increasing a(n) implies an increasing total number of faults expressed as

$$a(n) = a + \alpha m(n) \quad (4)$$

Substituting Equation (4) in Equation (3) we have

$$m(n+1) - m(n) = bp(a + \alpha m(n) - m(n)) \quad (5)$$

Solving Equation (5) under the initial condition $m(n = 0) = 0$ we get

$$m(n) = \frac{a}{1-\alpha}\left[1 - \left(1 - bp(1-\alpha)\right)^n\right] \quad (6)$$

This Equation (6) can be rewritten as:

$$m(n) = a * F(n) \tag{7}$$

where,

$$\begin{cases} a* = \dfrac{a}{1-\alpha} \\ F(n) = 1-\left(1-bp(1-\alpha)\right)^n \end{cases} \tag{8}$$

Release 1:

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing starts once the code of software is written. Before the release of the software in the market the software testing team tests the software rigorously to make sure that they remove maximum number of bugs in the software. The first release is the foundation of the software so testing team are bound to give their best effort. Although it is not possible to remove all the bugs in the software practically. Therefore, when the software is tested by the testing team, there are chances that they may detect a finite number (less than the total content of the faults) of bugs in the code developed.

So finite numbers of bugs are then removed and mathematical equation for it is given as under:

$$m_1(n) = a_1 * F_1(n) \qquad 0 < n \le n_1 \tag{9}$$

where,

$$\begin{cases} a_1* = \dfrac{a_1}{1-\alpha_1} \\ F_1(n) = 1-\left(1-b_1 p_1(1-\alpha_1)\right)^n \end{cases} \tag{10}$$

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, $a_1 * (1 - F_1(n_1))$ is the leftover fault content of the first version which interacts with new portion of detected faults i.e.,

$F_2(n - n_1)$. In addition a fraction of faults generated due to enhancement of the features are removed with new rate. Here it may be noted that there is a change in the fault detection rate. This change in the fault detection may be due to change in time, change in the fault complexity due to new features or change in testing strategies etc. The mathematical equation of these finite numbers of faults removed can be given by:

$$m_2(n) = a_2 * F_2(n - n_1) + a_1 * (1 - F_1(n_1))F_2(n - n_1) \quad n_1 < n \leq n_2 \qquad (11)$$

where,

$$\begin{cases} a_2 * = \dfrac{a_2}{1-\alpha_2} \\ F_2(n-n_1) = 1-\left(1-b_2 p_2 (1-\alpha_2)\right)^{n-n_1} \end{cases} \qquad (12)$$

Release 3:

The modeling for release 3 is done on the basis of the arguments similar to given in second release along with taking into consideration the fact that the next release will not contain the remaining faults of all previous releases, rather it will be dependent on the just previous release. A proportion of faults get removed when the testing team tests the new code and these faults are removed with the detection proportion $F_3(n - n_2)$. During the testing of newly integrated code, apart from the faults lying in the new code, a number of bugs which have remained undetected i.e., $a_2 * (1 - F_2(n_2 - n_1))$ are also removed with the detection proportion $F_3(n - n_2)$. The resulting equation is as following:

$$m_3(n) = a_3 * F_3(n - n_2) + a_2 * (1 - F_2(n_2 - n_1))F_3(n - n_2) \quad n_2 < n \leq n_3 \qquad (13)$$

where,

$$\begin{cases} a_3 * = \dfrac{a_3}{1-\alpha_3} \\ F_3(n-n_2) = 1-\left(1-b_3 p_3 (1-\alpha_3)\right)^{n-n_2} \end{cases} \qquad (14)$$

Similarly for release 4, the corresponding mathematical expression can be given by:

$$m_4(n) = a_4 * F_4(n - n_3) + a_3 * (1 - F_3(n_3 - n_2))F_4(n - n_3) \quad n_3 < n \leq n_4 \qquad (15)$$

where, $a_4 *$ and $F_4(n - n_3)$ can be defined as done in previous steps.

4. Parameter analysis

Parameters estimation is of primary concern in software reliability prediction. For this, the failure data is collected and is recorded in either of the following two formats-

the first approach is to record the time between successive failures while second way is to collect the number of failures experienced at regular testing intervals. The mean number of faults detected/removed by testing periods $m(n)$ is mostly described by the non-linear functions and once the analytical solution is known for a given model, the parameters in the solution are required to be determined. Parameter estimation is done by Non-linear Least Square (NLLS). For this nonlinear regression (NLR) module of SPSS has been used.

4.1 Model validation

To check the validity of the proposed model to describe the software reliability growth, it has been tested on Tandem Computers (Wood, 1996). The data set includes the failure data from 4 major releases of the software at Tandem Computers. In the First Release, 100 faults were detected after testing for 20 weeks. The Second Release was done after detecting 120 faults for 19 weeks. The Third and Forth Release were done after testing for 12 and 19 weeks, removing 61 and 42 faults respectively. Table 1 gives the value of the parameters and Table 2 provides the comparison criteria's.

The performance of SRGM is judged by their ability to fit the past software failure occurrence / fault removal data and to predict satisfactorily the future behavior of the software failure occurrence / fault removal process. Figures 2 ~ 5 give the Goodness of Fit curves for the four releases.

Table 1 Parameter Estimates

Release 1		Release 2		Release 3		Release 4	
a_1	109.24	a_2	118.58	a_3	62.187	a_4	43.316
b_1	0.3286	b_2	0.2777	b_3	0.3332	b_4	0.0374
p_1	0.7119	p_2	0.7028	p_3	0.6691	p_4	0.6728
α_1	0.1899	α_2	0.1838	α_3	0.1739	α_4	0.0097

Table 2 Comparison Criteria

Criteria	R2	Bias	Variation	MSE
Release 1	.990	0.403	2.81	7.71
Release 2	.995	0.214	2.159	7.065
Release 3	.995	0.050	1.490	1.909
Release 4	.992	0.075	1.106	1.163

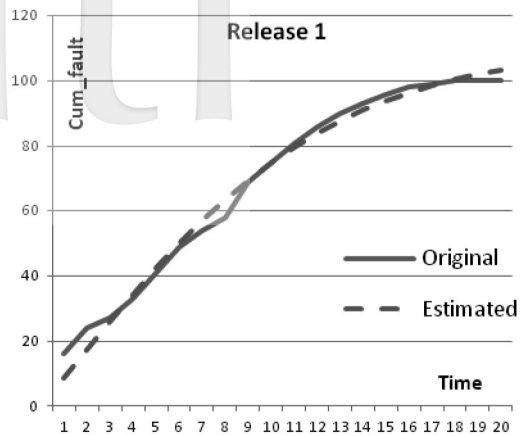

Figure 2 Goodness of Fit for Release 1

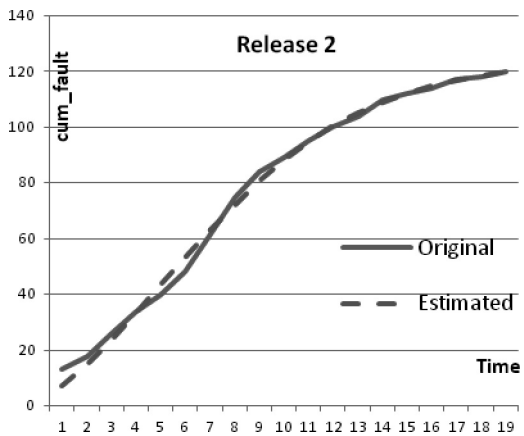

Figure 3 Goodness of Fit for Release 2

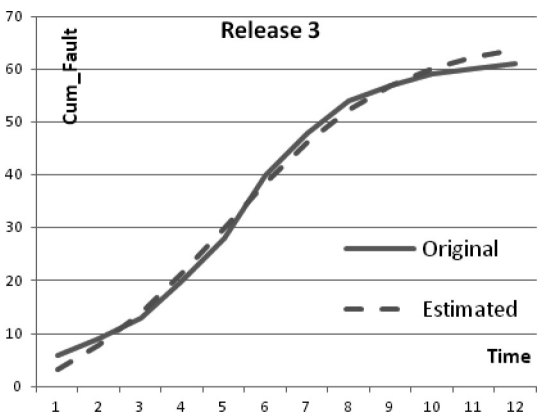

Figure 4 Goodness of Fit for Release 3

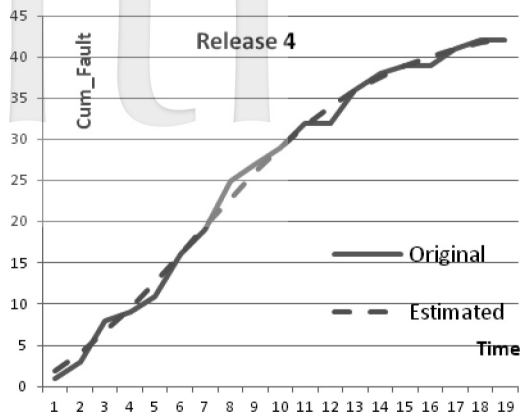

Figure 5 Goodness of Fit for Release 4

5. Conclusion

Making reliable software is the need of an hour. Every customer needs a more efficient and bug free software. Software products in general face a fierce competition in the market and therefore have to come up with upgraded versions of the software. But the matter of the fact is that up-gradation is a complex and difficult process. The add-ons can result in distorting the fault removal rate and can cause change in the number of fault contents. The software reliability multi up-gradation model developed in this paper incorporates this concept of Imperfect Debugging and is based on the assumption that the overall fault removal of the new release depends on the faults generated in that release and on the leftover faults of just previous release (for each release). This helps us in removing more and more faults in the software and produces highly reliable software. The proposed multi up gradation model is estimated on a four release data set.

Acknowledgements

The work done in this paper is supported by grants to the second author via grant No DRCH/R&D/2013-14/4155 and RC/2014/6820 from University of Delhi, India. Further, we would like to acknowledge that this paper is an extended version of a paper titled “A Discrete Formulation of Successive Software Releases Based on Imperfect Debugging” presented at an International Conference on Reliability, Infocom Technologies and Optimization, January 29 ~ 31, 2013

References

Bittanti, S., Bolzern, P., Pedrotti, E. and Scattolini, R. (1988), 'A flexible modelling approach for software reliability growth', in Goos, G. and Hartmanis, J. (Eds.), *Software Reliability Modelling and Identification*, Springer Verlag, Berlin, Germany, pp. 101-140.

Goel, A.L. (1985), 'Software reliability models: assumptions, limitations and applicability', *IEEE Transactions on Software Engineering*, Vol. SE-11, pp. 1411-1423.

Goel, A.L. and Okumoto, K. (1979), 'Time-dependent error-detection rate model for software reliability and other performance measures', *IEEE Transactions on Reliability*, Vol. 28, No. 3, pp. 206-211.

Jelinski, Z. and Moranda, P.B. (1972), 'Software reliability research', in Freiberger, W. (Ed.), *Statistical Computer Performance Evaluation*, Academic Press, New York, NY, pp. 465-497.

Kapur, P.K., Aggarwal, A.G. and Nijhawan, N. (2014), 'A discrete SRGM for multi release software system', *International Journal Industrial and System Engineering*, Vol. 16, No. 2, pp. 143-155.

Kapur, P.K. and Garg, R.B. (1992), 'A software reliability growth model for an fault removal phenomenon', *Software Engineering Journal*, Vol. 7, pp. 291-294.

Kapur, P.K., Garg, R.B. and Kumar, S. (1999), *Contributions to Hardware and Software Reliability*, World Scientific, Singapore.

Kapur, P.K., Pham, H., Gupta, A. and Jha, P.C. (2011), *Software Reliability Assessment with OR Applications*, Springer, London, UK.

Kapur, P.K., Singh, O., Garmabaki, A.S. and Singh, J. (2010), 'Multi up-gradation software reliability growth model with imperfect debugging', *International Journal of Systems Assurance Engineering and Management*, Vol. 1, pp. 299-306.

Kapur, P.K., Tandon, A. and Kaur, G. (2010) 'Multi up-gradation software reliability model', *Proceedings of the 2nd IEEE International Conference on Reliability, Safety & Hazard*, Mumbai, Indian, pp. 468-474.

Musa, J.D., Iannino, A. and Okumoto, K. (1987), *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, NY.

Ohba, M. (1984), 'Software reliability analysis models', *IBM Journal of Research and Development*, Vol. 28, No. 4, pp. 428-443.

Pham, H. (2006), *System Software Reliability*, Springer, New York, NY.

Wood, A. (1996), 'Predicting software reliability', *Computer*, Vol. 29, pp. 69-77.

Yamada, S., Ohba, M. and Osaki, S. (1984), 'S-shaped software reliability growth models and their applications', *IEEE Transactions on Reliability*, Vol. 33, No. 4, pp. 289-292.

About the **authors**

Jagvinder Singh is working as Assistant Professor in Maharaja Agrasen College, University of Delhi. He received his Doctoral Thesis in 2012 from Department of Operational research, University of Delhi. His area of research is Software Reliability Modeling. He has published several papers in International/National Journals and Proceedings. He is a lifetime member of the Society for Reliability Engineering, Quality and Operations Management (SREQOM). E-mail address: jagvinder.singh@gmail.com

Adarsh Anand is doing research in the area of Innovation Diffusion Modeling and Software Reliability Assessment. Presently he is working as an Assistant Professor in the Department of Operational Research, University of Delhi (INDIA). He did his PhD. And M Phil in Operational Research in 2013 and 2010 respectively. He has published several papers in International/National Journals and Proceedings. He is a lifetime member of the Society for Reliability Engineering, Quality and Operations Management (SREQOM).
Corresponding author. Room No. 208, 2nd Floor, Department of Operational Research, University of Delhi, Delhi 110007, India. Tel: 011-27666960. E-mail address: adarsh.anand86@gmail.com

Avneesh Kumar is pursuing his PhD from Jiwaji University, MP. Currently he holds the position of a Lecturer in INMANTEC, UP. He has been active member of the Society for Reliability Engineering, Quality and Operations Management (SREQOM). E-mail address: avn119@rediffmail.com

Sunil Kumar Khatri is working as Director in Amity Institute of Information Technology, Amity University, Noida, India. He is a Fellow of IETE, Sr. Member of IACSIT and of Computer Society of India and Member of IEEE. He has been conferred "IT Innovation & Excellence Award for Contribution in the field of IT and Computer Science Education" by *Knowledge Resource Development & Welfare Group* Dec, 2012 and the award for "Exceptional Leadership and Dedication in Research" during the *4th International Conference on Quality, Reliability and Infocom Technology* in the year 2009. He has edited three books, two special issues and published several papers in international and national journals and proceedings. His areas of research are Software Reliability, Modeling and Optimization, Data Mining and Warehousing, Network Security, Soft Computing and Pattern Recognition. E-mail address: sunilkkhatri@gmail.com