Certain Reliability Growth Models for Debugging in Software Systems

JAHIR PASHA, S.Ranjitha, Dr. H. N. Suresh

Abstract — **A large number of software reliability growth models (SRGMs) have been proposed during the past thirty years to estimate software reliability measures such as the number of residual faults, software failure rate, and software reliability. Selection of optimal SRGM for use in a particular case has been an area of interest for researchers in the field of software reliability. Tools and techniques for software reliability model selection found in the literature cannot provide high level of confidence as they use a limited number of model selection criteria. There is therefore a need for evolving more efficient techniques. An effort has been made in this paper to review some of the well known techniques of this area and the possibility of developing a more efficient technique. A number of analytical models have been proposed during the past three decades for assessing the reliability of the software system. In this paper we will summarize some existing Software Reliability Growth Models (SRGMs), provide a critical analysis of the underlying assumptions, and assess the applicability of these models during the software development cycle . There is therefore a need for evolving more efficient techniques. An effort has been made in this paper to review some of the well known techniques of this area and the possibility of developing a more efficient technique.**

Keywords — **Software Reliability, Software Reliability Growth Models (SRGMs), Fault, Failure, Imperfect Debugging, S-Shaped Model.**

I. INTRODUCTION

Towards moving 21's century, software becomes a coercer for everything from elementary education to genetic engineering. Dependency and requirements on computer increases the difficulties and failures. Due to increase in addiction, the size & complexity of the system have grown. To avoid the failures & faults, reliability of software needs to be studied during the development of software so as to come up with reliable software [1]. Testing is the major quality control used during software development [2]. The quality of the software system has many attributes such as maintainability, portability, usability, security, reliability, availability, etc. Software reliability is the most dynamic attribute, which can measure and predict the operational quality of the product. Software Reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. The aim of software reliability engineers is to increase the probability that a designed program will work as intended in the hands of the customers [2,8].

A commonly accepted metric for quantifying a product's reliability is the number of faults one can expect to find within a certain time. Failures are the result of a fault in the software code, and several failures can be the result of one fault. The process of finding and removing faults to improve the software reliability can be described by a mathematical relationship called a Software Reliability Growth Model (SRGM) [3]. SRGM is a mathematical model of how the software reliability improves as faults are detected and repaired. SRGM can be used to predict when a particular level of reliability is likely to be attained. Thus, SRGM is used to determine when to stop testing to attain a given reliability level [4,9]. There are essentially two types of software reliability models - those that attempt to predict software reliability from design parameters and those that attempt to predict software reliability from test data [5].

Various SRGM have been developed during the last three decades and they can provide very useful information about how to improve the reliability. One can easily determine some important metrics like time period, number of remaining faults, Mean Time Between Failures (MTBF), and Mean Time To Failure (MTTF) through SRGM. For the past several decades, various statistical models have been proposed to access the software reliability. The most common approach for developing software reliability model is the probabilistic approach. The probabilistic model represents the failure occurrences and the fault removals as probabilistic events. They are classified into various groups, including error seeding models, failure rate models, curve fitting models, reliability growth models, Markov structure models, and Non Homogenous Poisson Process (NHPP) models [6,7].

A. *Statement of Problem*

Software reliability represents the probability of failure-free software operation for a certain period of time in a particular environment. Over the past 30 years, lots of SRGMs have been developed and most SRGMs presuppose that detected faults are corrected at once. In literature, many software reliability growth models were utilized and the parameters related to software reliability were estimated. Besides the other, the most recently developed SRGM for distributed environment incorporating two types of imperfect debugging method estimates the software reliability in two types of software sub systems. The software sub-systems are reused system with simple faults, and newly developed system with hard faults and complex faults, respectively. The Mean Value Function (MVF) of all these systems were computed separately and summed to estimate the software reliability. However, this technique has drawbacks in fault consideration process. The software systems basically have independent and dependent faults in the debugging process. The existing SRGM has focused only on the fault severity. It will not find

Manuscript received April 20, 2014.

JAHIR PASHA, Research scholar, Dept. Of CS, Jain University, Bangalore, Karnataka, India.
S.Ranjitha, BE(ECE), Bangalore Institute of Technology, vv puram ,Bangalore-04
Dr .H. N. Suresh, Professor & Research coordinator, Bangalore Institute of Technology, Dept. of IT, VV puram, Bangalore

whether the faults are independent or dependent to the system. The lack of such fault consideration in imperfect debugging process does not produce accurate estimation results. If the aforesaid drawbacks in the literary works are solved, then the SRGM imperfect debugging process will be improved with high accurate results. Hence, the lack of solution for such drawbacks has motivated to do the research work in this area.

II. LITERATURE SURVEY

Sharma *et al.* [10] have developed a deterministic quantitative model based on a Distance Based Approach (DBA) method, then applied it for evaluation, optimal selection, and ranking of SRGMs. DBA recognizes the need for relative importance of criteria for a given application, without which inter-criterion comparison could not be accomplished. It requires a set of model selection criteria, along with a set of SRGMs, and their level of criteria for optimal selection; and it successfully presents the results in terms of a merit value, which has been used to rank the SRGMs. They have used two distinct, real data sets for demonstration of the DBA method. The results were the ranking of SRGMs based on the Euclidean composite distance of each alternative to the designated optimal SRGM.

P. Bubnov *et al.* [11] have proposed a generalized software reliability model on the basis of non-stationary Markovian service system. Approximation by Coxian distribution allows investigating software reliability growth for any kinds of distribution (for example, Weibull, Gamma) of time between the moments of program errors detection and fixing. The model enables to forecast important software reliability characteristics, such as number of corrected errors, number of errors to be fixed, required debugging time, etc. The diagram of transitions between states of the generalized model and differential equations system has been presented. The example of calculation with use of the offered model has been considered, research of influence of Coxian distributions variation coefficients of duration of intervals between the error detection moments and error correction time distributions on values of look-ahead characteristics has been executed.

Kapur *et al.* [12] have proposed two general frameworks for deriving several software reliability growth models based on a Non-Homogeneous Poisson Process (NHPP) in the presence of imperfect debugging and error generation. The proposed models were initially formulated: for the case when there was no differentiation between failure observation and fault removal testing processes, and then extended for the case when there was a clear differentiation between failure observation and fault removal testing processes. During the last three decades, many SRGMs have been developed to describe software failures as a random process, and can be used to evaluate development status during testing. With SRGM, software engineers can easily measure (or forecast) the software reliability (or quality), and plot software reliability growth charts. It is not easy to select the best model from a plethora of models available. In real software development environments, the number of failures observed need not to be same as the number of faults removed. Due to the complexity of software systems and an incomplete understanding of software, the testing team may not be able to remove the fault perfectly on observation of a failure, and the original fault may remain, resulting in a phenomenon known as imperfect debugging, or get replaced by another fault causing error generation. In the case of imperfect debugging, the fault content of the software remains the same; while in the case of error generation, the fault content increases as the testing progresses. Removal of observed faults may result in the introduction of new faults.

Software Reliability is defined as the probability of free-failure operation for a specified period of time in a specified environment. Software Reliability Growth models (SRGM) have been developed to estimate software reliability measures such as number of remaining faults, software failure rate and software reliability. Imperfect debugging models have been considered in these models. However, most SRGM assume that faults will eventually be removed. Purnaiah *et al.* [13] have aimed to incorporate the fault removal efficiency in software reliability growth modeling. Here, imperfect debugging has been considered in the sense that new faults can be introduced into the software during debugging and the detected faults may not be removed completely.

Khatri *et al.* [14] have discussed a discrete software reliability growth model for distributed system considering imperfect debugging that faults are not always corrected/removed when they are detected and fault generation. Their proposed model assumes that the software system consists of a finite number of reused and newly developed sub-systems. The reused sub-systems do not involve the effect of severity of the faults on the software reliability growth phenomenon because they stabilize over a period of time i.e. the growth is uniform whereas, the newly developed subsystem does involve. For newly developed component, it has been assumed that removal process follows logistic growth curve due to the fact that learning of removal team grows as testing progresses. The fault removal phenomena for reused and newly developed sub-systems have been modeled separately and were summed to obtain the total fault removal phenomenon of the software system. The model has been validated on two software data sets and it has been shown that the proposed model fairs comparatively better than the existing one.

III. SOFTWARERELIABILTY GROWTH MODELS

A Software Reliability Growth Model is one of the fundamental techniques to assess software reliability quantitatively [26], [38]. The Software Reliability Growth Model required having a good performance in terms of goodness-of-fit, predictability, and so forth. 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. Any software required to operate reliably must still undergo extensive testing and debugging. This can be a costly and time consuming process, and managers require accurate information about how software reliability grows as a result of this process in order to effectively manage their budgets and projects. The effects of this process, by which it is hoped software is made more reliable, can be modeled through the use of Software Reliability Growth Models, hereafter referred to as SRGMs. Research efforts in software reliability engineering have been conducted over the past three decades and many software reliability growth models (SRGMs) have been proposed. SRGMs can estimate the number of initial faults, the software reliability, the failure intensity, the mean time-interval between failures, etc. Ideally, these models provide a means of characterizing the development process and enable software reliability practitioners to make predictions about the expected future reliability of software under development. Such techniques allow managers to accurately allocate time, money, and human resources to a project, and assess when a piece of software has reached a point where it can be released with some level of confidence in its reliability. Unfortunately, these models are often inaccurate. A comparative study of Software Reliability Growth Models allows to determine with models is suited well and under what conditions. A number of analytical models have been proposed to address the problem of software reliability measurement. These approaches are based mainly on the failure history of software and can be classified according to the nature of the failure process studied as indicated below.

A. Times between Failures Models

In this class of models, the process under study is the time between failures. The most common approach is to assume that the time between, say, the (i 1 st −) and the ith failures, follows a distribution whose parameters depend on the number of faults remaining in the program during this interval. Estimates of the parameters are obtained from the observed values of times between failures and estimates of software reliability, mean time to next failure, etc., are then obtained from the fitted model. Another approach is to treat the failure times as realizations of a stochastic process and use an appropriate time-series model to describe the underlying failure process. b. Failure Count Models. The interest of this class of models is in the number of faults or failures in specified time intervals rather than in times between failures. The failure counts are assumed to follow a known stochastic process with a time dependent discrete or continuous failure rate. Parameters of the failure rate can be estimated from the observed values of failure counts or from failure times. Estimates of software reliability mean time to next failure, etc., can again be obtained from the relevant equations.

*B. **Fault Seeding Models***

The basic approach in this class of models is to "seed" a known number of faults in a program which is assumed to have an unknown number of indigenous faults. The program is tested and the observed numbers of seeded and indigenous faults are counted. From these, an estimate of the fault content of the program prior to seeding is obtained and used to assess software reliability and other relevant measures.

C. Input Domain Based Models

The basic approach taken here is to generate a set of test cases from an input distribution which is assumed to be representative of the operational usage of the program. Because of the difficulty in obtaining this distribution, the input domain is partitioned into a set of equivalence classes, each of which is usually associated with a program path. An estimate of program reliability is obtained from the failures observed during physical or symbolic execution of the test cases sampled from the input domain .

IV. FAULT COUNT MODEL

This class of models is concerned with modeling the number of failures seen or faults detected in given testing intervals. As faults are removed, from the system, it is expected that the observed number of failures per unit time will decrease. If this is so, then the -cumulative number of failures versus time curve will eventually level off. Note that time here can be calendar time, CPU time, number of test cases run or some other relevant metric. In this setup, the time intervals may be fixed a priori and the observed number of failures in each interval is treated as a random variable. Several models have been suggested to describe such failure phenomena. The basic idea behind most of these models is that of a Poisson distribution whose parameter takes different forms for different models. It should be noted that Poisson distribution has been found to be an excellent model in many fields of application where interest is in the number of occurrences. One of the earliest models in this category was proposed by Shooman [3]. Taking a somewhat similar approach, Musa [12] later proposed another failure count model based on execution time. Schneidewind took a different approach and studied the fault counts over a series of time intervals. Goel and Okumoto [7] introduced a time dependent failure rate of the underlying Poisson process and developed the necessary analytical details of the models. A generalization of this model was proposed by Goel [2]. These and some other models in this class are described below.

A. Goel-Okumoto Nonhomogeneous Poission Process Model

In this model Goel and Okumoto [6] assumed that a software system is subject to failures at random times caused by faults present in the system. Letting N(t) be the cumulative number of failures observed by time t, they proposed that N(t) can be modeled as a no homogeneous Poisson process, i.e., as a Poisson process with a time dependent failure rate.

B. Goel Generalized Non homogeneous Poisson Process Model

Most of the times between failures and failure count models assume that a software system exhibits a decreasing failure rate pattern during testing. In other words, they assume that software quality continues to improve as testing progresses. In practice, it has been observed that in many testing situations, the failure rate first increases and then decreases. In order to model this increasing/decreasing failure rate process, Goel [2], [3] proposed the Goel-Okumoto NHPP model.

C. Musa Execution Time Model

In this model Musa [13] makes assumptions that are similar to those of the JM model except that the process modeled is the number of failures in specified execution time intervals.

D. Shooman Exponential Model :

This model is essentially similar to the JM model.

E. Generalized Poisson Model :

This is a variation of the NHPP model of Goel and Okumotoand assumes a mean value function is the total number of faults removed up to the end of the (i - 1) st debugging interval, Ø is a constant of proportionality, and a is a constant used to rescale timed .

F. IBM Binomial and Poisson Models :

In these models Brooks and Motley [4] consider the fault detection process during software testing to be a discrete process, following a binomial or a Poisson distribution. The software system is assumed to be developed and tested incrementally. They claim that both models can be applied at the module or the system level.

G. Musa-Komodo Logarithmic Poisson Execution Time Model

In this model [10] the observed number of failures by some time T is assumed to be a NHPP, similar to the Goel-Okumoto model, but with a mean value function which is a function of τ.

V. APPLICABILITY OF SOFTWARE RELIABILITY MODELS

In this section we consider the four classes of Software Reliability Models and assess their applicability during the design, unit testing, integration testing, and operational phases of the software development process.

A. Design Phase

During the design phase, -faults may be detected visually or by other formal or informal procedures. Existing software reliability models are not applicable during this phase because the test cases needed to expose faults as required by fault seeding and input domain based model do not exist, and the failure history required by time dependent models is not available.

B. Unit Testing

The typical environment during module coding and unit testing phase is such that the test cases generated from the module input domain do not form a representative sample of the operational usage distribution. Further, times between exposures of module faults are not random since the test strategy employed may not be random testing. In fact, test cases are usually executed in a deterministic fashion. Given these conditions, it seems that the fault seeding models are applicable provided it can be assumed that the indigenous and seeded faults have equal probabilities of being detected. However, a difficulty could arise if the programmer is also the tester in this phase. The input domain based models seem to be applicable, except that matching the test profile to operational usage distribution could be difficult. Due to these difficulties, such models, although applicable, may not be usable. The time dependent models, especially the time between failures models, do not seem to be applicable in this environment since the independent times between failures assumption is seriously violated.

C. Integration Testing

A typical environment during integration testing is that the modules are integrated into partial or whole systems and test cases are generated to verify the correctness of the integrated system. Test cases for this purpose may be generated randomly from an input distribution or may be generated deterministically using a reliable test strategy, the latter being probably more effective. The exposed faults are corrected and there is a strong possibility that the removal of exposed faults may introduce new faults. Under such testing conditions, fault seeding models are theoretically applicable since we still have the luxury of seeding faults into the system. Input domain based models based on an explicit test profile distribution are also applicable. The difficulty in applying them at this point is the very large number of logic paths generated by the whole system. If deterministic testing (e.g., boundary value analysis, path testing) is used, times between failures models may not be appropriate because of the violation of the independence of inter failure times assumption. Fault count models may be applicable if sets of test cases are independent of each other, even if the tests within a set are chosen deterministically. This is so because in such models the system failure rate is assumed to decrease as a result of executing a set of test cases and not at every failure. If random testing is performed according to an assumed input profile distribution, then most of the existing software reliability models are applicable. Input domain based models, if used, should utilize a test profile distribution which is statistically equivalent to the operational profile distribution. Fault seeding models are applicable likewise, since faults can be seeded and the equal probability of fault detection assumption may not be seriously violated. Thesis due to the random nature of the test generation process. Times between failures and failure count models are most applicable with random testing. The next question could be about choosing a specific model from a given class. Some people prefer to try a couple of these model sons the same failure history and then choose one. However, because of different underlying assumptions of these models, there are subtle distinctions among them. Therefore, as far as possible, the choice of a specific model should be based on the development environment considerations.

D. Acceptance Testing

During acceptance testing, inputs based on operational usage are generated to verify software acceptability. In this phase, seeding of faults is not practical and the exposed faults are not usually immediately corrected. The fault seeding and times between failures models are thus not applicable. Many other considerations here are similar to those of integration testing so that the fault count and input domain based models are generally applicable.

E. Operational Phase

When the reliability of the software as perceived by the developer or the "friendly users" is already acceptable, the software is released for operational use. During the operational phase, the user inputs may not be random. This is because the user may use the same software function or path on a routine basis. Inputs may also be correlated(e.g., in real-time systems), thus losing their randomness. Furthermore, faults are not always immediately corrected. In this environment, fault-count models are likely to be most applicable and could be used for monitoring software failure rate or for determining the optimum time for installing a new release.

VI. OBJECTIVES

The main objective of this study is to provide an efficient SRGM method for estimating accurate imperfect debugging in software systems. The objectives of this study are highlighted as follows:

- To develop a SRGM by estimating imperfect debugging in the software systems.
- To reduce fault consideration process by considering the systems independent and dependent faults.
- To develop Yamada Weibull-Type Testing-Effort Function Model for software reliability estimation.
- A selection function, which chooses the best candidate to be added to the solution
- A feasibility function, that is used to determine if a candidate can be used to contribute to a solution
- An objective function, which assigns a value to a solution, or a partial solution,
- A solution function, which will indicate when we have discovered a complete

VII. PROPOSED METHODOLOGY

The main aim of this research is to provide a better SRGM by solving the drawbacks that currently exist in the literary works. Thus, it is intended to propose a SRGM by estimating the imperfect debugging in the software systems. In the proposed technique, initially the basic assumptions about software faults will be discussed. The proposed SRGM will be based on the Non Homogeneous Poisson Process (NHPP). The software faults MVF will be computed for all faults presented in the software systems. Subsequently, the proposed technique will develop a Yamada Weibull-Type Testing-Effort Function Model for the software reliability estimation process. The software reliability estimated by our proposed technique will be more accurate than the existing technique. The technique will be implemented in the JAVA platform and the results will be analyzed to demonstrate the performance of the proposed SRGM technique.

VIII. POSSIBLE OUTCOME

By using the proposed SRGM, the software systems reliability will be computed by considering the faults in the software systems. The performance of our proposed technique will be tested by two data sets, and the results from both the datasets will be compared with the conventional SRGM. Overall, the proposed technique will accurately find the reliability of the software by considering the faults presented in the software systems

REFERENCES

[1] Tariq Hussain Sheakh and VijayPal Singh, "Taxonomical study of Software Reliability Growth Models", International Journal of Scientific and Research Publications, Vol. 2, No. 5, pp. 1-3, May 2012
[2] Anni Princy and Sridhar, "An Efficient Software Reliability Growth Models with Two Types of Imperfect Debugging", European Journal of Scientific Research, Vol. 72, No. 4, pp. 490-503, 2012
[3] Vincent Almering, Michiel van Genuchten, Ger Cloudt and Peter J.M. Sonnemans, "Using Software Reliability Growth Models in Practice", IEEE Computer Society, pp. 82-88, 2010
[4] Quadri and Ahmad, "Software Reliability Growth Modeling with New Modified Weibull Testing–effort and Optimal Release Policy", International Journal of Computer Applications, Vol. 6, No.12, pp. 1-10, September 2010
[5] Preethi and Aasha, "A Quality Oriented Approach for Developing an Enhanced Software Reliability Growth Model", International Journal of Advanced Research in Technology, Vol. 2, No. 2, pp. 39-44, March 2012
[6] Geetha Rani Neppala, Satya Prasad and Kantam, "Software Reliability Growth Model using Interval Domain Data", International Journal of Computer Applications, Vol. 34, No. 9, pp. 5-8, November 2011
[7] Khaled M. S. Faqih, "What is Hampering the Performance of Software Reliability Models? A literature review", In Proceedings of the International Multi Conference of Engineers and Computer Scientists, Hong Kong, Vol. I, 2009
[8] Chin-Yu Huang, Sy-yen Kuo and Lyu, "An Assessment of Testing-Effort Dependent Software Reliability Growth Models", IEEE Transactions on Reliability, Vol. 56, No. 2, pp. 198-211, 2007
[9] Quadri, Ahmad and Peer, "Software Optimal Release Policy and Reliability Growth Modeling", In Proceedings of the 2nd National Conference on Computing for Nation Development, New Delhi, India, pp. 423-431, 2008
[10] Sharma, Garg, Nagpal and Garg, "Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach", IEEE Transactions on Reliability, Vol. 59, No. 2, pp. 266-276, 2010
[11] Vladimir P. Bubnov, Alexey V. Tyrva and Anatoly D. Khomonenko, "Model of Reliability of the Software with Coxian Distribution of Length of Intervals between the Moments of Detection of Errors", In Proceedings of IEEE Conference on Annual Computer Software and Applications, Munich, pp. 310-314, 2011s
[12] Kapur, Pham, Anand and Yadav, "A Unified Approach for Developing Software Reliability Growth Models in the Presence of Imperfect Debugging and Error Generation", IEEE Transactions on Reliability, Vol. 60, No. 1, pp. 331-340, 2011
[13] Purnaiah, Rama Krishna and Bala Venkata Kishore, "Fault Removal Efficiency in Software Reliability Growth Model", Advances in Computational Research, Vol. 4, No. 1, pp.-74-77, 2012
[14] Sunil Kumar Khatri, P.K. Kapur and Prashant Johri, "Flexible Discrete Software Reliability Growth Model for Distributed Environment Incorporating Two Types of Imperfect Debugging", In Proceedings of the International Conference on Advanced Computing & Communication Technologies, Rohtak, Haryana, pp. 57-63, 2012

M.Jahir Pasha was born in Kurnool, Andhra Pradesh, India in 1988.He is currently pursuing Doctors degree at department of computer science and engineering, Jain University, Bangalore, India. He has obtained his Bachelor of Technology and Master of Technology in Information Technology and Reliability Engineering from Jawaharlal Nehru technological university Hyderabad, Andhra Pradesh, India, and Jawaharlal Nehru technological university, Anantapur, Andhra Pradesh, India in 2009 and 2011 respectively. He is currently working as assistant professor in the department of computer science and engineering, Modugula kalvathamma institute of technology for women, Rajampet, kadapa, Andhra Pradesh, India. His current research is focused on software reliability.

Dr. H.N. Suresh received his BE (E&C) from P.E.S College of Engineering, Mysore University, Karnataka, India, in the year 1989 and completed his M.Tech (Bio Medical Instrumentation) from SJCE Mysore affiliated to University of Mysore., in the year of 1996 and since then he is actively involved in teaching and research and has Twenty six years of experience in teaching. He obtained his PhD (ECE) from Anna university of Technology.He worked at various capacities in affiliated University Engineering Colleges. For Visveswaraya Technical University and Bangalore University he worked as a Chairman for Board of Examiners, and member of Board of Studies etc. At present he is working as Professor and BIT research coordinator in Bangalore Institute of Technology, Bangalore Affiliated to Visveswaraya Technical University. He has good exposure in the field of signal processing, Wavelet Transforms, Neural Networks,Pattern recognition,Bio Medical Signal Processing, Netwoking and Adaptive Neural network systems. He has published more than 30 research papers in the refereed international journals and presented contributed research papers in refereed international and national conferences. He is a member of IEEE, Bio Medical Society of India, ISTE,IMAPS & Fellow member of IETE.