Proceeding of the National Conference; INDIACom-2007
Computing for Nation Development, February 23 –24, 2007
Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi

A Software Reliability Growth Model With Two Types of Imperfect Debugging For Project and Product Software During Operational Phase

P. C. Jha Anshu Gupta P. K. Kapur
Department of Operational Research, University of Delhi, Delhi – 110007
{jhapc@yahoo.com, anshu_or@yahoo.com, pkkapur@gmail.com}

ABSTRACT
Several Software Reliability Growth Models (SRGM) have been proposed in the literature for modeling the software reliability growth during the testing phase. In field the software is subject to an environment, which is different from that of testing. Therefore a SRGM developed for the testing phase is not suitable for estimating the reliability growth during the testing phase. In this paper, we propose a generalized Software Reliability Growth Model, which can be used to estimate number of faults both in the testing and operational phase. During the testing phase it is appropriate to estimate the reliability growth with respect to the amount of testing resources spend on testing while during the operation phase the number of failure detected and hence the reliability depends on the usage of software. Appropriate usage functions are linked to both project and product type software. To describe the fault removal phenomenon, imperfect debugging environment is incorporated into the model building. Study related to this paper highlights an interdisciplinary modeling approach in Software Reliability Engineering and Marketing. The proposed model is validated for both the phases by supplying the data sets obtained from different sources. Results are encouraging.

KEYWORDS
Software Reliability Growth Model, Imperfect Debugging, Error Generation, Testing Phase, Operation Phase, Usage Function, Testing Efforts, Project Software, Product Software.

1. INTRODUCTION

Advancement of Information Technology along with Globalization and free trade during the last two decades has changed the outlook and working of business and industry completely. The global marketplace has become fiercely competitive where schedule, quality and cost are parameters with which competence is measured. The situation calls for planning, controlling and scientific decision making for the proper functioning of an organization and tradeoffs between many conflicting objectives under system constraints. Operational Research is a scientific disciple used to model complex systems and to optimize their performance. One of the fields where modeling, particularly stochastic modeling and optimization have vastly been applied is reliability. The subject has traditionally been attached to hardware systems. But with ever increasing use of computers in present times software reliability has also emerged as a discipline of its own where operational researchers can meaningfully contribute.

The current scenario is that computers and computer-based systems have invaded every sphere of human activity. Due to ease of use and faster performance more and more systems are being automated. Dependence of mankind on computers is rapidly increasing. A mere postponement of a function can led to big losses in terms of money and time. High precision and safety of software component is desired for a smooth operation of a system. Therefore software developers lay special emphasis on testing their software.

During the testing phase test cases designed on the users specification are executed on the software and if any departure from specifications or requirements occurs, is termed as a failure. An immediate effort is made to remove the cause of that failure (a fault in the software). Testing goes on until the release time set by management or a desired reliability objective is achieved. No software can be tested exhaustively before release due to constraints on time and cost. This is the reason, why we often hear about failures of software in operational phase and sometimes even for safety critical systems. Hence it is important to study how the reliability of software grows both during testing and operational phase. Such a study also helps management in deciding when to stop testing and release the software during the testing phase. Like any other product software developers also give warranty on their products to their licensed users. If a failure occurs in users environment, it is reported to the developer. The developing team isolates and removes the fault that has caused the failure. Once the fault is removed the developer update the licensed users code with no additional cost during the warranty period. An early estimation of the failure\removal phenomenon during the testing phase can assists management in decision making related to the warranty they can offer on their product. SRMG can help in developing such a quality metric.

Many Software Reliability Growth Models (SRGMs) have proposed in literature to estimate the reliability of software during testing. Many authors have tried to extend SRGMs to represent the failure phenomenon during the operational phase, typically used in release time problem of software [10]. But this approach is not justified since during the operation profile the software is subject to a different environment as compared to the testing phase for most of the commercial software products. During testing phase testing is done under controlled environment. Testing resources such as manpower and computer time consumed can be measured and extended further to the user phase [10,14,21]. Mathematical models have been proposed for testing effort but they are not suitable for measuring usage of software. The intensity with which failures would manifest during the operational use is dependent upon the number of times the software is used which can be well described by a usage function, which depends on the number of executions of the software in field.

Before we describe how to model the reliability growth firstly we classify the software into two categories - Project type software and Product type software. Project type software is designed for specific applications for known operational environment as specified by the user. However multiple usage of the software is possible with in that user environment. The developer does not market such software. Product type softwares are developed for the general purpose and are marketed in the open market. Many distinct users may buy a licensed copy of such software and use it for their own purpose. During the testing phase the type of software under testing does not affect the reliability growth since in this phase the testing environment doesn't depends on the type of software. However during the reliability growth is greatly influenced by the type of software. The usage function of project type software is different form that of product type software. However in literature no distinction is made between the two types of software.

Kenny [12] has proposed a model to estimate the number of faults remaining in the software during its operational use. He has assumed a power function to represent the usage rate of the software. The author assumes that the rate at which the commercial software is used is dependent upon the number of its users; but the model proposed by him fails to capture the growth in number of users of the software. Kapur et al [6,11] proposed a model in this series where a marketing model (Bass model [1]) describing number of adoption of a product over time is used to model the user growth. However both of these models considers product type software and a perfect debugging environment. In this paper we develop an SRGM for testing phase, which can be extended to the operational phase, thus providing a unique approach to modeling both testing and operational phase under imperfect debugging environment. An attempt has been made to model the reliability growth of both type of software during the operation phase linking to an appropriate usage function.

In real life situations, most of the debugging processes or the fault removal efficiency is not perfect. The fault removal team may not be able to remove the fault perfectly on the detection of a failure and the original fault may remain or replaced by another fault. When a failure occurs, the cause of the failure is identified and removed. To ensure that the cause is perfectly fixed, the software is tested for the same input and if a failure occurs again, the code is checked again. Two possibilities occur. The fault, which was thought to be perfectly fixed, has been imperfectly repaired and caused same type of failure again when checked on the same input. However, it may also happen some other kind of failure occurs which might be due to the fact that the fault was perfectly removed but some other fault was generated while removing the cause of the failure. This is called error generation, which can be known only during the removal phase. Imperfect fault debugging causes more failures as compared to removals by time infinity but the fault content remains the same. However, when a fault is generated, the number of failures increases because the fault content has increased. Some models have been developed in literature to incorporate the effect imperfect debugging in modeling software reliability [9,16,18,23]. In the proposed model we have incorporated the effect of both type of imperfect debugging on the removal process.

This paper is organized as follows: First in section 2 a general description of a NHPP based SRGM is given. Then a general framework of the model is developed in section 2.2. In section 3 we have discussed about modeling the testing effort function. Further in section 4 we model the usage function for both product and project type software. The parameter estimation also constitutes an important part of model building. Parameter estimation of the models proposed in the paper have been discussed and illustrated on software failure data sets cited in literature [3,13] in section 5. Finally conclusions are drawn in section 6.

2. FRAMEWORK FOR MODELING

2.1 NHPP SRGM BASED – AGENERAL DESCRIPTION

Several SRGM have been proposed in literature to measure the reliability of software during the testing phase. Many of these can be classified under the title of Non Homogeneous Poisson Process (NHPP) models. These NHPP models are based on the assumption that 'Software failure occurs at random times during testing caused by faults lying dormant in the software'. Hence NHPP can be used to describe the failure phenomenon during both these phases. The counting process $\{N(t), t \geq 0\}$ of an NHPP process is given as follows.

$$\text{Pr. } \{N(t) = k\} = \frac{\{m(t)\}^k}{k!} e^{-m(t)}, \; k = 0,1,2, \qquad \ldots(2.1)$$

and $$m(t) = \int_0^t \lambda(x)dx$$

The intensity function $\lambda(x)$ (or the mean value function $m(t)$) is the basic building block of all the NHPP models existing in the software reliability engineering literature. These models assume diverse testing environments like distinction between failure and removal processes, learning of the testing personnel, possibility of imperfect debugging and error generation etc.

2.2 MODEL DEVELOPMENT

In this section we develop the general framework of the proposed model. In models proposed by Yamada et al. [21] and Trachtenberg [19], the effect of intensity of testing effort on the failure phenomenon has been studied. During the testing phase test cases stimulating the users environment are executed, and if a failure occurs the corresponding fault is identified and removed. In the testing phase the most dominating factors affecting the failure phenomenon are the test cases, testing environment and the testing efforts spend during testing. Testing efforts include the manpower and the computing time. Efficiency and skill of the testing\fault removal team greatly influence the debugging process. While in operational phase the most dominating factor affecting the reliability growth is the rate at which failures would occur which depends upon its usage. The number of executions in the operational profile describes usage function. Hence SRGM should incorporate the effect of testing effort in the testing phase and the usage function in the operation phase. A number of functions exist in the literature that can be used to describe the testing effort or the usage function with time.

Imperfect debugging greatly influence the reliability growth both during testing and operational phase. The fault removal rate per remaining fault reduces due to imperfect fault debugging. We assume that fault removal rate per remaining fault is a function of both time and perfect debugging probability p. Whereas due to fault generation the initial fault content of the software increases as the testing progresses. We have assumed a constant error generation rate. Using the basic building blocks of this framework SRGM for both testing and operational phases can be developed with ease. The proposed model is based upon the following basic notations and assumptions.

Notations:

$m(t)$	: Expected number of faults identified in the time interval $(0,t]$
$\lambda(t)$	: Failure rate, $\lambda(t) = dm(t)/dt$.
$W(t)$	: Cumulative testing effort\Usage in the time interval $(0,t]$ and $\frac{d}{dt}W(t) = w(t)$
a	: Constant, representing the number of faults lying dormant in the software at the beginning of testing.
$a(t)$	: S-expected fault content at time t, $a>0$.
p	: Probability of perfect debugging of a fault.
α	: Constant rate of error generation.
β,μ,	: Constants
c, k, r, s	: Constants
$\bar{W}$	: Constant, representing the saturation point for the testing effort of software.
$\bar{N}$	: Constant, representing the saturation point for the user growth of software.
F(t)	: is the fraction of ultimate potential adopters of the software.

Assumptions

1. The Non-homogeneous Poisson Process (NHPP) can describe software failure phenomenon.
2. As soon as a failure occurs the fault causing that failure is immediately identified.
3. Software is subject to failures during execution caused by faults remaining in the software.
4. Reliability growth during the testing phase is dependent upon the testing efforts spend on testing.
5. The number of failures during operation phase is dependent upon the usage function.
6. Usage function\testing effort function is a function of time and usage function depends on the number of executions of the software in field.
7. When a software failure occurs, an instantaneous repair effort starts and the following may occur:
   (a) Fault content is reduced by one with probability p
   (b) Fault content remains unchanged with probability 1-p.
8. During the fault removal process, whether the fault is removed successfully or not, new faults are generated with a constant probability α.
9. Fault removal rate per remaining fault is assumed to be non-decreasing inflection S-shaped logistic function.

Using the assumptions 4, 5 and 6 the removal phenomenon can be described with respect to time as follows:

$$\frac{dm(t)}{dt} = \frac{dm(t)}{dW(t)}\frac{dW(t)}{dt} \quad \ldots(2.2)$$

Further using assumptions 7, 8 and 9 equation (2.2) can be expanded as

$$\frac{dm(t)}{dt} = b(p, W(t))(a(t) - m(t))\frac{dW(t)}{dt} \quad \ldots(2.3)$$

Where

$$b(p,W(t)) = \frac{bp}{1+\beta e^{-bpW(t)}} \quad \text{and} \quad a(t) = (a + \alpha m(t)) \quad \ldots(2.4)$$

Equation (2.3) using (2.4) can be further written as

$$\frac{dm(t)}{dt} = \frac{bp}{1+\beta e^{-bpW(t)}}(a + \alpha m(t) - m(t))\frac{dW(t)}{dt} \quad \ldots(2.5)$$

Solving equation (2.5) under the initial condition $m(0) = 0$ and $W(0) = 0$ we get

$$m_r(t) = \frac{a}{1-\alpha}\left[1 - \left(\frac{(1+\beta)e^{-bpW(t)}}{1+\beta e^{-bpW(t)}}\right)^{(1-\alpha)}\right] \quad \ldots(2.6)$$

In the next two sections we discuss how to model the testing effort and usage functions.

3. MODELING TESTING EFFORT

The resources that govern the pace of testing for almost all software projects [14] are

1. Manpower, which includes
   (a) Testing\Failure identification personnel
   (b) Programmers\Failure correction personnel
2. Computer time

Various testing effort functions have been discussed in literature. Three forms viz. Exponential, Rayleigh and Weibull functions can be derived under the assumption that, "the testing effort rate is proportional to the testing resource available at that time" differential equation describing the testing effort expenditure rate is given by

$$\frac{dW(t)}{dt} = c(t)\left[\bar{W} - W(t)\right] \qquad \ldots(3.1)$$

Where *c(t)* is the time dependent rate at which testing resources are consumed, with respect to remaining available resources. Solving equation (3.1) under the initial condition *W(t=0)=0*, we get

$$W(t) = \bar{W}\left[1 - \exp\left\{\int_0^t c(x)dx\right\}\right] \qquad \ldots(3.2)$$

If *c(t)=c* (a constant) an exponential curve is obtained

$$W(t) = \bar{W}\left(1 - e^{-ct}\right) \qquad \ldots(3.3)$$

If *c(t)=ct*, (3.2) gives Rayleigh type curve

$$W(t) = \bar{W}\left(1 - e^{-ct^2/2}\right) \qquad \ldots(3.4)$$

If $c(t) = c\mu t^{\mu-1}$, a more flexible and general testing effort function is obtained given by a Weibull function and the cumulative testing effort consumed in the interval (0,t] has the following form

$$W(t) = \bar{W}\left(1 - e^{-ct^{\mu}}\right) \qquad \ldots(3.5)$$

Exponential and Rayleigh curves become special cases of the Weibull curve for $\mu = 1$ and $\mu = 2$ respectively.

Huang et al. [7] developed an SRGM, based upon NHPP with a logistic testing effort function. The cumulative testing effort consumed in the interval $(0,t]$ has the following form

$$W(t) = \frac{\bar{W}}{1 + \mu e^{-ct}} \qquad \ldots(3.6)$$

SRGM with logistic testing effort function provides better result on some failure data sets. To study the testing effort process, one of the above functions can be chosen depending upon the testing process.

4. MODELING THE USAGE FUNCTION IN THE OPERATIONAL PHASE

In the operation phase the rate at which failure would occur is dependent upon the usage of software. The mathematical form of usage function is dependent upon the number of executions of the software in field and is a function of time. However the usage function of project type software is different from that of product type software. In literature no distinction is made between the two type of software, but the models due to Kenny and Kapur et al address to the product type software.

4.1 USAGE FUNCTION FOR A PROJECT TYPE SOFTWARE

Project type software is owned by a specific organization running it for their specific use, for example computerized banking system. With in the organization many users may be assessing it either at a single location or at different locations. We propose an exponential function given by equation (4.1) to model the usage function for such software.

$$W(t) = r + s(1 - e^{-ct}) \qquad \ldots(4.1)$$

Where r represent the initial usage of the software when it is implemented in the user environment. As the time progress the usage of software grows within the organization until it reaches the saturation level r+s. Although some other functional form can also be used depending upon the user environment, number of people assessing the software and the usage of the software at each terminal.

4.2 Usage function for a Product Type Software

Product type softwares are the general-purpose software and are marketed in the open market. Many different customers may buy a licensed copy of the software for different purpose. At any licensed buyer end many uses might be assessing it. For example an educational institution buy software and many students and\or faculty members might be assessing it. Number of executions of product software depends on the total number of user of the software using it for their own specific purpose. Although commercial software products are there in the market for the last two decades identifying the target customers with certainty is impossible. However product software also come into the category of technological products and as such behaves as a new product or an innovation when released in the market.
Kenny [12] used the power function given as

$$W(t) = \frac{t^{(k+1)}}{(k+1)} \qquad \ldots(4.2)$$

to describe the growth in the user population of software in operational phase. The function can correctly describe the users growth in terms of

a) A slow start but gain in growth rate
b) A constant addition of users
c) A big beginning and tail off in the usage rate.

But in the Marketing literature power function is seldom used for the purpose as described above. One of the reasons can be that the parameters of the function are not amenable to interpretations.

Kapur et al[11] used Bass model for innovation diffusion [1] in marketing for the dynamic market of software products for predicting the successive growth in the number of adopters of a product over time. The parameter of the model explicitly categorizes the adopters into innovators and imitators. Innovators have independent decision making abilities whereas imitators make the purchase decisions after getting first hand opinion from a user. The model can adequately describe the users growth in terms of the factors stated above. Adopters (or users) of software report a failure caused by some fault remaining in the software to the developer. Once the number of users of the software is known, the rate at which instructions in the software are executed can be estimated. For applying the Bass model it is assumed that there exists a finite population of prospective users who with time increasingly become actual users of the software (no distinction is made between users and purchasers here as Bass model has been successfully applied to describe the growth in number of both of them). In each period there will be both innovators and imitators using the software product. However as the process continues the relative number of innovators will diminish monotonically with time. Imitators are however influenced by the number of previous buyers and increase relative to the number of innovators as the process continues.

The likelihood of adoption at time t given that one has yet not occurred is [15]

$$\frac{f(t)}{(1-F(t))}=(r+sF(t)) \qquad \ldots(4.3)$$

Where f(t) is the density function of F(t). The term $[r+sF(t)]$ represent the combined rate of first purchasing of innovators and imitators per remaining adoption and increases through time because F(*t*) increases through time. Whereas the fraction of non-adopters (1-F(t)) will decrease with time. The shape of the resulting sales curve will depend upon relative rate of these two tendencies. If software product is successful, the coefficient of imitation is likely to exceed the coefficient of innovation i.e., r<s. On the other hand, if r>s, the sales curve will fall continuously.
The solution of (4.3) for F(*t* = 0) = 0 is

$$F(t)=\frac{1-\exp^{-(r+s)t}}{1+(s/r)\exp^{-(r+s)t}} \qquad \ldots(4.4)$$

So if $\bar{N}$ denote upper limit on the number of license buyers of the software and γ is the average number of users within the user environment then total number of users of the software by time t is given as

$$S(t)=\bar{N}\gamma F(t)=mF(t) \quad Where \quad m=\bar{N}\gamma \qquad \ldots(4.5)$$

Givon et al. [4] have used the modified version of the above model to estimate the number of licensed users as well as users of pirated copies of the software. Though it can be reasonably assumed that only the licensed-copy holders would report the failures, and hence equation (4.5) can be used to find the expected number of users at any time during the life cycle of the software. If the new software is expected to go through the same history as some previous software (very likely for versions of the same software) the parameters of earlier growth curve may be used as an approximation.
Estimating the expected number of licensed user of software the rate at which instructions in the software are executed can be estimated. Since the usage function depends on the number of executions of the software Therefore we assume the usage function for product type software is a function of the total number of users of the software. For simplicity we that ν is the average execution rate at which the software is used within a user environment i.e.

$$W(t)=f(S(t))=\nu S(t) \qquad \ldots(4.6)$$

Some other functional relationship can be assumed used depending upon the user environment and number of people assessing the software within a particular licensed user environment.

The various testing effort functions and the usage functions discussed above can be used in the SRGM given by equation (2.6) to model the reliability growth of software in testing and operational phase respectively depending upon the testing efforts used or the type of software.

5. MODEL VALIDATION AND PARAMETER ESTIMATION

The success of software reliability growth model depends heavily upon quality of failure data collected. The parameters of the SRGMs are estimated based upon these data. Method of least squares or maximum likelihood has been suggested and widely used for estimation of parameters of an SRGM. The models discussed in this paper are non-linear model and it is difficult to find solution for nonlinear models using Least Square method and require numerical algorithms to solve it. Statistical software packages such as SPSS help to overcome this problem.

5.1 COMPARISON CRITERIA

1. The Mean Square Fitting Error (MSE):
The model under comparison is used to simulate the fault data, the difference between the expected values, $\hat{m}(t_i)$ and the observed data y*ᵢ* is measured by MSE as follows.

$$MSE=\sum_{i=1}^{k}\frac{(\hat{m}(t_i)-y_i)^2}{k}$$

Where k is the number of observations. The lower MSE indicates less fitting error, thus better goodness of fit.

2. Coefficient of Multiple Determination (R^2):
We define this coefficient as the ratio of the sum of squares resulting from the trend model to that from constant model subtracted from 1.

5.2 MODEL VALIDATION

To validate the models three real software failure data sets have been chosen. First is collected during the testing phase, Second data set is based on the failure reports of software in operation phase for project software while Third data sets are based on failure reports of software in operational use for product software. First using the observed data we have estimated the testing effort function. The testing effort function which best describe the data is chosen and using those estimated values parameters of the SRGM for testing phase are estimated. To estimate the parameters of the SRGM for operation phase first we substitute the usage functions in the model expression and then estimate the parameters for both

SRGM and usage function for both type of softwares since the data related to the usage of software is not available.

Data set-1: This failure data set is for a command, control and communication system cited in Brooks and Motley [3]. The software was tested for 12 months and 2657 faults were identified during this period. The estimated values of testing effort functions discussed in this paper are given in table 1. From the table 1 it can be seen that exponential testing effort functions fits best to this data set. Using the estimated values of exponential effort function parameter of the mean value function for the SRGM for testing phase given by equation (2.6) are estimated and are tabulated in table 2. The Fitting of the models is illustrated graphically in figure 1 and 2.

Table-1 Estimation results for Testing Effort Functions

Testing Effort Function	Estimated Parameters			Comparison Criteria	
	W	c	μ	MSE	R^2
Exponential	**35237**	**0.0297**	**-**	**12869.59**	**0.99854**
Rayleigh	10153	0.0491	-	414823.9	0.95302
Logistic	11714	0.3488	8.773	25977.78	0.99706
Weibull	33524	0.0313	1.002	13070.04	0.99852

Table-2 Estimation Results of SRGM for Testing Phase with Exponential Effort Function

Estimated Parameters					Comparison Criteria	
a	α	β	b	p	MSE	R^2
3756	0.00998	0.00001	0.000058	0.203	923.3002	0.99812

Figure 1:

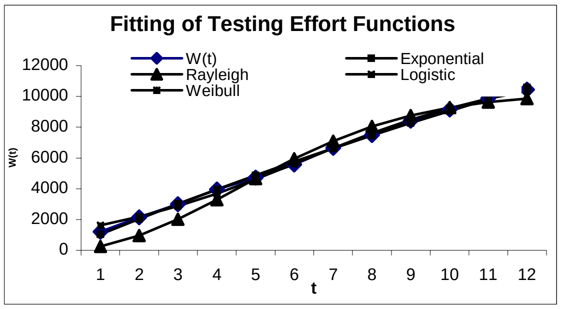
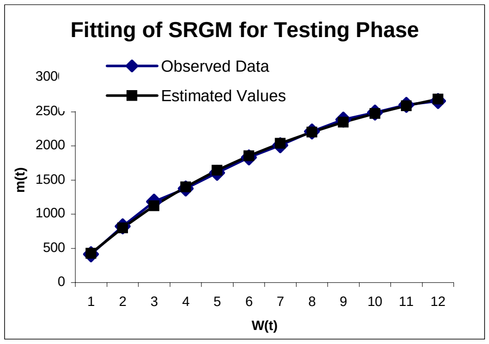


Figure 2:

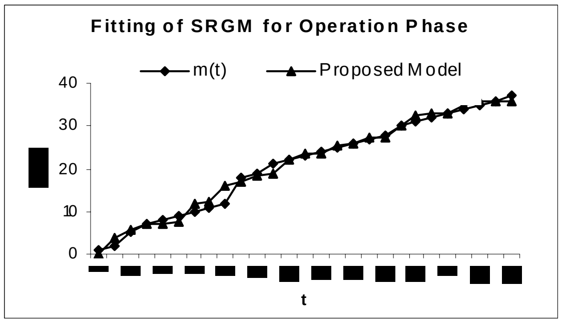


Data Set 2: This failure data is for Real time system collected for operation phase cited in Musa [13]. The data is available for 192 days during which 37 faults were identified. Substituting the usage functions given by equation (4.1) in (2.6) the parameters of the mean value function for the SRGM in operation phase for project type software are estimated and are tabulated in table 3. The Fitting of the model is illustrated graphically in figure 3.

Table-5 Estimation Results of SRGM for Operation Phase for Project Type Software

Estimated Parameters								Comparison Criteria	
a	α	β	b	p	56.7748	s	c	MSE	R^2
31	0.2413	56.8	0.6028	0.9680	0.1591	12.61	0.01253	1.4821	0.9880

Figure 5:

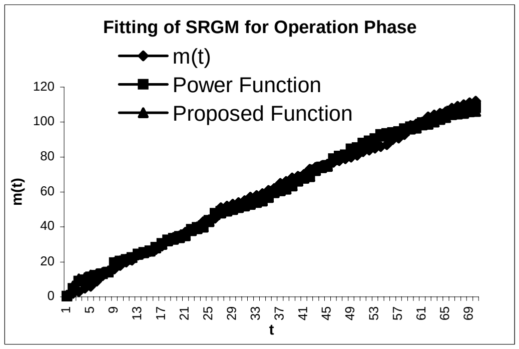


Data set-3: This failure data is for an operating system collected for operation phase cited in Musa [13]. The data is available for 148 day during which 112 faults were identified. Substituting the usage functions given by equation (4.2) and (4.6) in (2.6) the parameters of the mean value function for the SRGM in operation phase for product type software are estimated and are tabulated in table 4. The Fitting of the models is illustrated graphically in figure 4.

Table-6 Estimation Results of SRGM During Operation Phase for Product Type Software

Usage Function	Estimated Parameters						Comparison Criteria	
Power Function	a	α	β	b	p	k	MSE	R^2
	107	0.12587	0.78402	0.00628	0.85	0.3242	10.59705	0.98969
Proposed Function	a	α	β	b	p		16.45913	0.99071
	166	0.18999	20.2267	0.0028	0.7185			
	m	r	s	ν				
	86	0.01423	0.00150	22.138				

Figure 6:



6. CONCLUSION

In this paper we have proposed a general framework for modeling reliability growth of software during testing and operational phase under an imperfect debugging environment. SRGM for the testing phase is modeled with respect to testing effort function, which can be extended to the operation phase simply by replacing testing effort functions by the usage functions. An attempt has been made to model the reliability growth of both type of software during the operation phase linking to an appropriate usage function. Proposed models are validated on real life data sets and the results are encouraging.

FUTURE SCOPE

In this paper we have considered a software in isolation. However large software systems are not designed in isolation rather a software developer develops and releases multiple versions of a software product successively. The latest version of software retains some code of the previous version of the software, some proportion of which can also be retained in the versions to be released in future and some new code is added to the software to further enhance the functioning of the latest version. Therefore there is some interdependence in the failure phenomenon of these releases. In future we will focus on developing a SRGM considering the interdependence in the failure phenomenon of multiple releases.

REFERENCES

[1] Bass FM. "A new product growth model for consumer durables" *Management Science* 15(5): 215-224, 1969.

[2] Bardhan AK. "Modelling in software reliability and its interdisciplinary nature" *Ph.D. thesis, University of Delhi, Delhi* 2002.

[3] Brooks WD, Motley RW. "Analysis of discrete software reliability models - Technical Report (RADC-TR-80-84)*" New York: Rome Air Development Center,* 1980.

[4] Givon M, Mahajan V, Muller E. "Software piracy: Estimation of lost sales and the impact on software diffusion*" Journal of Marketing* 59: 29-37,1980.

[5] Goel AL, Okumoto K. "*Time dependent error detection rate model for software reliability and other performance measures" IEEE Transactions on Reliability* R-28(3): 206-211,1979.

[6] Jha. P.C. "Optimization and Modeling in Software Reliability and Marketing", *Ph.D. thesis, University of Delhi, Delhi* 2003..

[7] Huang C-Y, Kuo S-Y, Chen JY. "Analysis of a software reliability growth model with logistic testing effort function", *Proc. 8th International Symposium on Software Reliability Engineering*, 378-388, November 1997.

[8] Kapur PK, Garg RB. "*A software reliability growth model for an error removal* phenomenon" *Software Engineering Journal* 7: 291-294, 1992.

[9] Kapur, P.K. and Younes, S., "Modeling an Imperfect Debugging Phenomenon in Software Reliability", *Microelectronics and Reliability*, 36(5), pp 645-650,1996.

*[10]*Kapur PK, Garg RB, Kumar S. "Contributions to hardware and software reliability", *World Scientific, Singapore,* 1999.

[11] Kapur P.K., Jha P. C., Goswami D.N., Shatnawi O and Bardhan A.K., "General Framework for Modeling Software Reliability Growth During Testing and Operational Use", *Recent Developments in Quality, Reliability and Information Technology, P. K. Kapur (Eds), IMH-Publisher, New Delhi, India,* 2003.

[12] Kenny GQ. "Estimating defects in a commercial software during operational use" *IEEE Trans. on Reliability* 42(1); 107-115, 1993.

[13] Musa JD. "Software reliability data" *Data and Analysis Center for software, USA and www.dacs.dtic.mil/,*1980.

[14] Musa JD, Iannino A, Okumoto K. "Software reliability: Measurement, Prediction, Applications" *New York: Mc Graw Hill,* 1987.

[15] Norton, J. A. and Bass, F. M., "A diffusion theory model of adoption and substitution for successive generations of high-technology products", *Management Science*, 33 (9), 1069-1086, 1987.

[16] Ohba M. and Chou, X.M., "Does Imperfect Debugging Effect Software Reliability Growth", *proceedings of 11th International Conference of Software Engineering*, pp 237-244, 1989.

[17] Ohba M. "Software reliability analysis models ",*IBM Journal of Research and Development* 28: 428-443, 1984.

[18] Pham, H., Nordmann, L. and Zang, X., "A General Imperfect Software Debugging Model with S-Shaped Fault Detection Rate", *IEEE Trans. Rel*., Vol. 48, pp 169-175, 1999.

[19] Trachtenberg M. "A general theory of software reliability modeling" *IEEE Trans. on Reliability"*, 39(1), 92-96, 1984.

[20] Wood A. "Predicting software reliability" *IEEE Computers*, 11: 69–77, 1996.

[21] Yamada S, Ohtera H, Narihisa H. "Software reliability growth models with testing-effort", *IEEE Trans. on Reliability* R-35, 19-23, 1986.

[22] Yamada S, Hishitani J, Osaki S "Software reliability growth model with a weibull test effort: A model and application" *IEEE Trans. On Reliability,* R-42, 100-106, 1993.

[23] Zang, X., Teng, X. and Pham, H., "Considering Fault Removal Efficiency inSoftware Reliability Assessment", *IEEE Trans. on Systems, Man and Cybernetics-Part A: Systems and Humans, Vol. 33, (1), 114-120*, 2003.