A SOFTWARE RELIABILITY GROWTH MODEL FOR VITAL QUALITY METRICS

R. Subburaj[1], **G. Gopal**[2] **and P.K.Kapur**[3]

[1]Electronics Test and Development Center (ETDC) and
Center For Reliability (CFR), India
Subburaj_spr@yahoo.com

[2]Department of Statistics
University of Madras, India
govgopal@yahoo.com

[3]Department of Operational Research
University of Delhi, India
pkkapur1@gmail.com

ABSTRACT

A Non-Homogenous Poisson Process (NHPP) model whose failure intensity function has the same mathematical form as that of a generalized exponential function was proposed for application as a Software Reliability Growth Model (SRGM). However, in order to facilitate collecting quality metrics pertaining to the degree of imperfect or efficient debugging phenomena and the number of faults left in the software, in this paper the authors propose an extension to the above SRGM. This SRGM enables adequate goodness of fit statistic and predictive validity, even when the software projects witness learning phenomenon of the testing team, either imperfect or perfect or efficient software debugging phenomenon, as well as wide fluctuations in time between failures – either occurring alone or in combinations thereof.

OPSOMMING

'n Nie-homogene Poissonproses (NHPP) waarvan die mislukkingsdigtheidsfunksie soortgelyk is aan 'n algemene eksponensiële funksie word voorgehou as 'n programmatuur-betroubaarheidsgroeimodel (PBGM). Die model lewer toereikende passingsgoedheid en voorspellingsgeldigheid onder uiteenlopende leereienskappe van toetsers, swak of goeie ontfouting van programmatuur, en groot verskille tussen waardes van tyd tussen mislukkings. Die outeurs stel ook voor dat die goedheid van ontfoutingsaksies gemeet word met behulp van 'n uitbreiding van die PBGM-model.

1. INTRODUCTION

Software Reliability Growth Models (SRGMs) find wide use in the software industry to measure reliability growth achieved during system testing, as well as to determine when to stop testing based on the reliability growth achieved. Some of these models also provide information on the quality of the testing and debugging processes, and can be used to study the level of competence exhibited by the various teams in the project. The models can be used to determine the quality of development process in terms of the number of faults in the software at the beginning of a system test, as well as to quantify the number of faults remaining in the software when it is released for operational use, so as to plan for maintenance.

Most SRGMs assume, for the sake of simplicity, perfect debugging – that is, when a failure is observed, the corresponding fault is identified and corrected with certainty, and no new faults are introduced [1]. According to Ohba and Chou [2], it is not realistic to assume that faults in a program are completely removed, since new errors are sometimes introduced when an error that was originally in the software is removed in response to to a failure. They propose an extension to the Goel-Okumoto (G-O) SRGM to eliminate the assumption of perfect debugging. They propose an additional parameter β such that $0 \leq \beta < 1$, as the probability of introducing new errors when fixing a detected error.

Kapur and Garg [3] note that when a failure occurs, an attempt is made to correct the cause of the failure. However, it is not always possible to find the cause and remove it, owing to lack of sufficient knowledge about the software. They assume that on a failure, the fault count is reduced by one with a probability $p_{0,}$ and fault contents are unchanged with probability $(1\text{-}p_0)$. Owing to such imperfect debugging, one fault may cause more than one failure. Thus, the number of failures at infinite testing time will be more than the total number of faults in the software at the beginning of testing.

When a failure occurs, sometimes the debugging team may also find and correct other faults in addition to the one that caused the failure. We call this an efficient debugging phenomenon since, subsequent to one failure, more than one fault is detected and corrected. Kapur and Garg (1992) [4] proposed an SRGM for this type of fault removal phenomenon with two mean value functions: one with an exponential growth curve for the failure phenomenon, and another with an S-shaped growth curve for the fault detection phenomenon. This leads to the situation that at infinite testing time, the cumulative number of failures will be less than the total number of detected faults, since some faults will be detected without their causing a failure.

Imperfect debugging is in contrast with efficient debugging, and they address mutually exclusive fault detection phenomena. However, inconsistency in the quality of debugging means that the occurrence of both the phenomena at different times during the execution of a project cannot be ruled out. In such cases, one has to look at the nett result. It may be noted that in both cases, the perfect debugging assumption made in many models [1] does not hold. Thus, the quality of debugging

can be classified into three categories: imperfect, perfect, and efficient debugging. Only in the case of perfect debugging will the number of failures be equal to the number of faults detected.

Furthermore, well-known Non-Homogenous Poisson Process models (NHPP) such as Goel-Okumoto (G-O) [5], Musa's Logarithmic Poisson Execution Time (LPET) model and Basic Execution Time (BET) [6] model, and the Kapur-Garg model (1992) [4] discussed above, as well as imperfect debugging models proposed by Kapur and Garg (1990) [3] and Ohba and Chou [2] are based on the assumption that failure intensity function decreases monotonically from the beginning of a system test, and hence do not address initially increasing and then decreasing (I/D) patterns of failure intensity function, observed in some projects owing to the learning phenomenon of test teams. The failure intensity function of both Yamada's delayed S-shaped SRGM [7] and Ohba's inflection S-shaped SRGM always assume an I/D pattern [8]. Models such as Goel's generalized NHPP model [5], and Log Power, derived from the well-known Duane Model [9], which are proposed to address both the above patterns of failure intensity variations, are at times sensitive to fluctuations in the number of failures in the data sets, and sometimes even miss the I/D pattern [10].

The generalized exponential Poisson model – the NHPP model with Generalized Exponential (GE) function Rate of Occurrence of Failures (ROCOF) – proposed by Subburaj and Gopal (S-G) [10] is found to fit failure data adequately from projects which possess the following characteristics, either alone or in combinations:

- Learning phenomenon of test team leading to I/D pattern of failure intensity function.
- Wide fluctuations in time between failures.
- Perfect debugging, or imperfect debugging, or efficient debugging.

This is owing to the advantages of Generalized Exponential distribution, such as increased flexibility for analyzing skewed data sets facilitated by the scale and shape parameters, increasing as well as decreasing hazard rate depending on the shape parameter, and ability to fit better [11]. However, since the model is based on failures and not faults, it does not give information about the degree of imperfect or efficient debugging and the quality of the development process (total number of faults), as well as the number of faults in the software at the time of product release. In this paper, the authors propose a simple extension to the S-G model to derive all the above quality metrics.

If an organization does not intend to collect the additional metrics discussed above, it can continue to use the S-G model: it is simpler, with only three parameters – and according to Ohba and Chou [2], the conventional SRGMs give fairly accurate estimates, even in conditions when the perfect debugging assumption is not valid. The reasons cited by them are given below:

- The new faults introduced while removing the initial faults might be negligibly small.

- Estimation error might absorb the effect caused by the imperfect debugging.

This paper is organized as follows. In Section 2, we derive the proposed SRGM and discuss the methodology for estimating its parameters for a given data set. In Section 3, we discuss the goodness of fit tests carried out on the proposed model and the conclusions reached. A better predictive ability for the proposed model has been established, and this is presented in Section 4. A comparison of the proposed extended model with the original model proposed by Subburaj-Gopal is given in Section 5, and a case study is offered in Section 6. The paper's conclusions are given in Section 7.

2. THE PROPOSED SRGM

2.1 NHPP models

The NHPP models have been quite successful tools in practical software reliability engineering [12]. G–O proposed an NHPP model assuming perfect debugging of faults, with no presence of learning phenomena in the testing. The mean value function of this exponential growth model is given by [5]:

$$\mu(t) = a[1 - \exp(-bt)] \qquad a > 0,\ b > 0 \text{ and } t > 0 \tag{1}$$

where

a: number of inherent faults detectable at infinite testing time
b: rate of detection of new fault

In order to facilitate modeling the learning phenomenon, as well as strictly decreasing the pattern of failure intensity function, and making the model robust to handle fluctuations in time between failures, Subburaj-Gopal (S-G) extended the G-O model. The mean value function of the S-G model is given by [10]:

$$\mu(t) = N\left[1 - \exp\left(-\frac{t}{\theta}\right)\right]^{\beta} \qquad N > 0,\ \beta > 0,\ \theta > 0 \text{ and } t > 0 \tag{2}$$

where

N: cumulative number of failures that will be observed over an infinite testing time
θ: scale parameter
β: shape parameter

The above models are based on the assumption that the debugging process is perfect. However, in some projects it is possible that fault detection and correction are sometimes erroneous and result in additional failures. In order to model imperfect debugging, Kapur and Garg (1990) [3] and Ohba and Chou [2] separately proposed similar imperfect debugging models. The mean value function for failures of the models is of the following form:

$$\mu(t) = \frac{a}{p}\left[1 - \exp(-bpt)\right] \qquad 0 < p \leq 1,\ a > 0,\ b > 0 \text{ and } t > 0 \tag{3}$$

where

p: probability of perfect debugging

If perfect debugging is witnessed in the project, then p will be equal to unity; if not, p will be less than 1, but greater than 0. This model cannot capture the I/D pattern of failure intensity function, arising out of the learning phenomenon.

Pham *et al* [12] show that faults are not always fully repaired and that new ones can be introduced as part of the fault repair process. They further assume that the learning of the testing team and imperfect debugging occur together at the same time, and accordingly they have integrated learning and imperfect debugging in the PNZ model they propose. In practice, it is found that the learning phenomenon of the testing team and the imperfect debugging phenomenon are independent of each other, and only in some cases do they occur together.

Zhang *et al* [13] note that when a failure occurs, a debugging effort is initiated immediately with probability p. For each debugging effort, whether the fault is successfully removed or not, some new faults may be introduced into the software system with a probability $\beta < p$. They also assume that the fault detection rate b(t) is a non-decreasing function with inflection S-shaped curve to capture the learning process. Thus this model is also restrictive, since it is assumed that learning occurs always with imperfect debugging. Their two case studies bring out a fault introduction probability of 0.012 and 0.00054 respectively. These are quite small, and will not have any effect on the model results reported by Ohba and Chou [2] and Goel [5].

Zeephongsekul *et al* [14] show that when a primary failure occurs a, debugging effort occurs immediately. That effort removes the fault with probability p_0. For an imperfect debugging of a primary failure, a secondary fault is introduced. When it is detected, it is removed with probability q_0.

A careful analysis of all the above previous work brings out the following:

- An increase in fault count owing to imperfect debugging is negligible
- A model should be capable of addressing imperfect debugging and the learning phenomenon separately as well as in combination, and there is a need to evolve such a model.

According to Xie [1], in most of the existing imperfect debugging SRGMs, there is a parameter p as mentioned above, which is defined as the probability of perfect debugging. It is to be noted that these models consider imperfect debugging in isolation. They have not addressed efficient debugging, wherein after a failure the debugging team finds and corrects more than one fault, which occurs in practice. Kapur and Garg [4] proposed another model to address efficient debugging. However, it will be convenient for software professionals if both efficient debugging

and imperfect debugging can be addressed by only one model.

In the Kapur-Garg model [3], parameter *a* is the total number of faults that will be detected at infinite time, and parameter *p* is the probability of perfect debugging. *N,* the total number of failures expected to occur at infinite time, can also be written as given below:

$$\lim_{t\to\infty}\mu(t) = N = \frac{a}{p} \tag{4}$$

Since the model proposed by them considered imperfect debugging only, they assumed that $0< a \leq N$ and therefore $p\leq 1$. We propose to include efficient debugging in the model as well, and therefore we propose to allow *N* the flexibility to assume values higher than or lower than or equal to *a* so that *p* is greater than zero, but not necessarily less than unity. When the projects witness efficient debugging, *p* is greater than 1.

The authors accordingly propose that the imperfect debugging model can be extended to address efficient debugging by redefining *p* as a real number >0. We will denote it as debugging index (*c*) to distinguish it from *p,* the probability. Since it is not a probability, it can even be greater than 1. In such a case, the differential equation given by Kapur and Garg [3] can be written as given below:

$$dm_2(t)/dt = bc(a - m_2(t)) \tag{5}$$

where

c: debugging index

Assuming *c* greater than zero, but not necessarily less than or equal to 1, has the following significance:

When $c<1$, the rate of change of mean value function for faults $m_2(t)$ will be slower than when there is perfect debugging; and when $c>1$, it will be faster, which is logically and mathematically correct. Thus this modification additionally enables the modeling of efficient debugging. Therefore, the equation for mean value function for faults corrected [3] as given below is valid even when $c>1$.

$$\mu_2(t) = a[1 - \exp(-bct)] \tag{6}$$

Similarly, the mean value function for failures given by Kapur and Garg [3] is also valid when the debugging index is redefined as a real number >0. It is given by:

$$\mu(t) = \left(\frac{a}{c}\right)[1 - \exp(-bct)] \tag{7}$$

Furthermore, according to Farr [15], the mean value function of Poisson type models is given by:

$$\mu(t) = \alpha F(t) \tag{8}$$

where

α: number of faults (failures) that will be detected in the software at infinite time
F(t): cumulative distribution function (cdf) of the time at failure of an individual fault

It may be noted that *F(t)* in the equations for $\mu_2(t)$ and $\mu(t)$ of the Kapur and Garg model (1990) [3] follows exponential distribution. Hence they cannot model the learning phenomenon – but in practice many projects witness the learning phenomenon along with imperfect, perfect, or efficient debugging. Therefore it is desirable to combine the learning phenomenon with various patterns of debugging. In order to address the learning phenomenon, as well as fit failure data from software projects irrespective of wide fluctuations in time between failures, the authors propose to use generalized exponential distribution for *F(t)* as in the case of S-G SRGM [10]. The mean value function of the proposed SRGM for faults and failures respectively, with modified generalized exponential function ROCOF, is given below:

$$\mu_2(t) = a\left[1 - \exp(-bct)\right]^d \tag{9}$$

$$\mu(t) = \left(\frac{a}{c}\right)\left[1 - \exp(-bct)\right]^d \tag{10}$$

where

c: debugging index greater than zero

Note that *c* is greater than zero, but not necessarily less than 1. This definition leads to modeling efficient, perfect, and imperfect debugging. Imperfect debugging is indicated by $0<c<1$, perfect debugging by $c=1$, and efficient debugging by $c>1$. When $c=1$, it reduces the model to S-G SRGM.

If $d<1$, then the failure intensity function decreases monotonically, and if $d>1$, the failure intensity function follows the I/D pattern, indicating the presence of the learning phenomenon. The property of the generalized exponential ROCOF enables data to be fitted accurately and consistently in spite of wide fluctuations in time between failures. Thus we derived an SRGM that provides vital quality metrics, with minor extension to the S-G model proposed earlier.

2.2 NHPP model with Modified Generalized Exponential ROCOF

The mean value function for failures of the proposed model, which is an extension of the S-G model, is parameterized and defined as given below:

$$\mu(t) = \frac{a}{c}\left[1 - \exp\left(-\frac{ct}{\theta}\right)\right]^{\beta} \qquad a>0,\ c>0,\ \beta>0,\ \theta>0 \tag{11}$$

where *a* is the eventual number of faults that will be detected over an infinite amount

of testing time, θ is the scale parameter >0, β is the shape parameter >0, and c is debugging index > 0.

It is to be noted that the cumulative number of failures, $\mu(0)$ =0 and $\mu(\infty) = (a/c)$. If imperfect debugging is witnessed in the project, then c will be less than *1*. Therefore the number of failures at infinite testing time will be greater than *a.* On the other hand, if the fault detection follows an S-shaped growth curve owing to efficient debugging, then c>*1*. This will lead to the condition that the number of failures at infinite testing time will be less than *a.* Thus, depending on the value of c, the proposed model will be able to model both the phenomena. When $c=1$, it is a case of perfect debugging, and the number of both faults and failures detected at infinite time will be equal to a.

The failure intensity function $\lambda(t)$ is the rate of change of mean value function over time, or the number of failures per unit time, and so it is the derivative of mean value function with respect to time [15], and is an instantaneous value. It is given by:

$$\lambda(t) = a\left(\frac{\beta}{\theta}\right)\exp\left(-\frac{ct}{\theta}\right)\left[1-\exp\left(-\frac{ct}{\theta}\right)\right]^{\beta-1} \tag{12}$$

The hazard rate of the above increases monotonically if $\beta > 1$, it is constant if $\beta = 1$, and it decreases monotonically if $\beta < 1$. In fact, it is this phenomenon that enables us to propose a suitable NHPP model that brings out all patterns of variations of failure intensity function correctly. Thus, the model will address the learning phenomenon with $\beta > 1$.

2.3 Determination of parameters of the model

The four parameters of the model for a given failure data set can be estimated using non-linear regression software tools. When a sufficient number of failures is observed, the parameters may be estimated with data consisting of the cumulative test time (x) and the corresponding cumulative number of failures (y), resulting in best fit of the data to the equation (11) for $\mu(t)$ of the model. The method of Maximum Likelihood Estimation (MLE) is preferred for parameter estimation because of its many desirable properties, such as asymptotic normality, asymptotic efficiency, and invariance [15]; and so the same methodology may be adopted in the tool for estimating parameters. The software tool will estimate the exact value of a, β, θ and c, but the ranges have to be specified by the user. The parameters thus obtained have to be substituted in the above equation to get the y values – namely, $\mu(t)$ corresponding to the cumulative test time at which the failures 1, 2,.. actually occurred.

2.4 Some salient features of the proposed NHPP model

- The total number of faults expected to be detected and failures expected to be observed in a software product at infinite time are finite, and are denoted by a and N respectively.

- Depending on the β value, the failure intensity may initially increase and then decrease, or start decreasing monotonically. The model addresses both patterns of variations of failure intensity function equally well. The learning phenomenon in the testing phase is indicated by $\beta > 1$.
- Imperfect debugging is brought out with c less than 1, and perfect debugging with c equal to 1. If the project witnesses an S-shaped growth of the fault detection phenomenon owing to efficient debugging, c will be greater than 1.
- The learning phenomenon can occur along with imperfect debugging, or perfect debugging, or S-shaped growth of fault detection.
- The model assumes that only one of the phenomena, namely imperfect debugging, or perfect debugging, or efficient debugging, is present in a project.
- The model can be used with time t expressed in terms of execution time, or test time, or calendar time.

3. PERFORMANCE EVALUATION OF THE PROPOSED MODEL

The performance of an SRGM can be assessed by its ability satisfactorily to fit the past failure data (goodness of fit, GoF) and to predict the time of occurrence of failures in the future (predictive validity) [16]. In order to check the GoF of the model, we have chosen Musa's [17] failure interval data from ten software projects – Musa P1, P2, P5, P6, P14C, P17, P27, SS2, SS3, and SS4. We will discuss goodness of fit measures in this section, and predictive validity in the next section.

A number of metrics have been evolved over the years for finding Goodness of Fit (GoF) of a model for given data. Only when GoF is achieved can the model be used for prediction or other purposes. Some of the GoF measures are given below:

- Coefficient of determination (R^2)
- Mean of Square Fitting Faults (MSF), which is also called Mean Square Fitting Error (MSE).
- Bias, Variation, and Root Mean Square Prediction error (RMSPE)

3.1 Coefficient of determination (R^2)

Coefficient of determination is a well known and widely used measure of the correlation between the dependent and independent variables in a regression analysis, carried out to determine the parameters of the model for a given data set. This measure is also known as the multiple correlation coefficient, and is denoted as R^2. It represents the proportion of data that is closest to the line of best fit. The value of R^2 may vary from 0 to 1. If the model fits the data perfectly, then $R^2=1$. If the model does not fit the data at all, then $R^2=0$. The closer R^2 is to 1, the better the fit. Most regression analysis tools provide R^2 metric readily on completion of the estimation of parameters for a given data set.

3.2 Mean of Square Fitting Faults

Mean of Square Fitting Faults (MSF) [16] is given by:

$$MSF = \left(\frac{1}{n}\right)\sum_{i=1}^{n}\left(x_i - E_i\right)^2 \quad (13)$$

where

n: number of data points
x_i: actual number of failures observed at test time *i*
E_i: number of failures estimated to occur by the model at the same instant of test time *i*

Table 1 below contains a number of important statistics pertaining to the chosen data sets, as well as GoF of the proposed model, as detailed below:

- R^2 and MSF, which indicate the closeness of fit of the proposed model for the given data set
- c, the debugging index of the project as estimated by the model
- β, if >1, indicates the existence of a learning phenomenon in the testing project
- Remarks give a summary of the nature of the project as inferred from the data analysis

Data set	R^2	MSF	β	c	Remarks
P1	0.995	3.8	0.5	1.08	No learning, mildly efficient debugging
P2	0.995	1.7	0.5	0.84	No learning, imperfect debugging
P14C	0.991	1.2	3.45	1.15	Learning, efficient debugging
P17	0.988	1.0	2.14	1.44	Learning, highly efficient debugging
P6	0.968	12.5	0.92	1.03	No learning, mildly efficient debugging
P27	0.97	3.6	0.7	0.95	No learning, imperfect debugging
SS2	0.993	25.1	1.57	0.424	Learning, imperfect debugging
SS3	0.994	38.5	1.42	1.14	Learning, efficient debugging
SS4	0.995	9.9	1.14	0.477	Learning, imperfect debugging
P5	0.991	433	0.867	0.33	No learning, highly imperfect debugging

Table 1: Goodness of Fit and other metrics

Table 1 confirms the consistently good performance of the proposed model, as R^2 was found to be very close to 1 for all the data sets, which in turn confirms the flexibility of the model to fit adequately in all cases. A smaller MSF indicates a smaller fitting error and better performance [16] of the model for the given data set. Learning of the test team in the project is indicated by $\beta > 1$. The data sets P14C, P17, SS2, SS3, and SS4 depicted an I/D pattern of failure intensity variations. Note also that β varies from project to project, and that it is high in project P14C. The debugging index *c* takes different values ranging from 0.33 to 1.44. Project P5

witnessed highly imperfect debugging, and project P17 highly efficient debugging. Thus the model is able to reveal valuable information about how the testing and debugging went in the projects.

3.3 Additional metrics for Goodness of Fit

Additional measures are available in the literature [18] to check the goodness of fit of SRGMs. They are defined below:

i) Prediction Error (PE): The difference between the observed number of failures and the predicted number of failures at any instant of time *i* is known as PE_i.

ii) Bias: The average of the PEs is called bias.

iii) Variation: The standard deviation of the PEs is known as variation.

$$\text{var}\,iation = \sqrt{\frac{\sum_{i=1}^{n}(PE_i - Bias)^2}{n-1}} \tag{14}$$

where *n* is the number of observations.

iv) Root Mean Square Prediction Error (RMSPE): A measure of the closeness with which a model predicts the observation. It is given by:

$$RMSPE = \sqrt{\left(Bias^2 + \text{var}^2\right)} \tag{15}$$

The additional metrics calculated to check the goodness of fit of the model are given in Table 2 below for the chosen data sets.

Dataset	Bias	Variation	RMSPE
P1	0	2.04	2.04
P2	- 0.01	1.39	1.39
P14C	0.02	1.17	1.17
P17	- 0.05	1.07	1.07
P6	0.08	3.73	3.73
P27	-0.06	1.99	1.99
SS2	0.45	5.26	5.28
SS3	0.3	6.54	6.54
SS4	0.24	3.31	3.32
P5	3.96	21.53	21.89

Table 2: Additional metrics for Goodness of Fit

The lower the value of Bias, Variation and RMSPE, the better is the goodness of fit. Table 2 confirms the ability of the model to fit data sets with widely varying characteristics.

4. PREDICTIVE VALIDITY

The predictive validity metrics are used to evaluate the forecasting quality of the SRGMs. According to Kapur and Garg [4], "predictive ability is defined as the ability of the model to determine future failure behaviour from present and past failure behaviour". The Relative Prediction Error (RPE) is defined as below [16]:

$$RPE = \left[\frac{\left(m(t_k) - m_k \right)}{m_k} \right] \tag{16}$$

where $m(t_k)$ is the number of failures estimated and m_k is the actually observed number of failures at the same instant of time t_k. In this connection, Chan *et al* [19] have defined normalized time Ψ as given below:

$$\psi = \frac{t_x}{T_x} \tag{17}$$

where T_x is the total observation interval for a failure data set X – or in other words, the cumulative test time at which the last failure is observed – and t_x is the time at which the model parameters are estimated for the same data set. To study predictive validity, we estimate the model parameters for the data set at different points in time (t_i) – that is. various values of Ψ less than 1.

The long-term predictive validity gives the relative error between estimated and observed values at the end of the observation time window. The model parameters are estimated at various values of Ψ to calculate long-term RPE. The long-term predictive validity for the proposed model for Musa's data set SS4 is given in Table 3.

Ψ	RPE
0.2	-0.02
0.4	-0.02
0.6	0.02
0.8	-0.015

Table 3: Long term predictive validity

The above table confirms the excellent predictive validity of the proposed model.

5. COMPARISON WITH S-G GE NHPP MODEL

While the performance (goodness of fit and predictive validity) of the proposed model (modified GE NHPP) is just as good as that of the GE NHPP model, the

proposed model provides additional quality metrics, namely the quality of debugging and the number of faults in the software product at the beginning of system testing. Table 4 below gives the additional information provided by the proposed model, and a comparison of *N*, the estimated number of failures at infinite time, yielded by both models. The S-G model brings out *N,* and the same can also be deduced from the proposed model, as given in Table 4.

Data Set	Modified GE NHPP			GE NHPP
	a	c	N = a/c	N
P1	200	1.08	186	185
P2	70	0.84	83.7	83.85
P5	980	0.33	2970	2950
P6	121	1.03	118	117
P14C	41.9	1.15	36.5	36.5
P17	47	1.44	32.9	34
P27	45	0.95	47.4	47.4
SS2	207.2	0.424	488	487.8
SS3	375.3	1.14	328	327.7
SS4	210	0.477	440	440.46

a: Predicted total number of faults
N: Predicted total number of failures
c: debugging index

Table 4: Comparison of N values predicted by the original model and the extended model

The N values predicted by both models match. Note that in Musa's project P5, owing to highly imperfect debugging, 2,950 failures have to be observed to correct 980 faults. This is one data set which most existing SRGMs fail to fit adequately. Efficient debugging was witnessed in five out of ten projects as chosen above. Thus the model provides the vital few metrics needed by any software development organization. Table 4 also validates the revised assumption for the debugging index, since the model estimates correctly the cumulative number of failures at infinite time when the project witnesses imperfect as well as efficient debugging.

Figure 1 below gives the comparison of mean value function – the cumulative number of failures estimated by both the models and the number of failures actually observed in Musa's project P2. Both the models estimate a nearly equal number of failures at various times of testing, thus confirming that the introduction of additional parameter *c* for capturing a debugging index has not affected the characteristics of the original GE NHPP model.

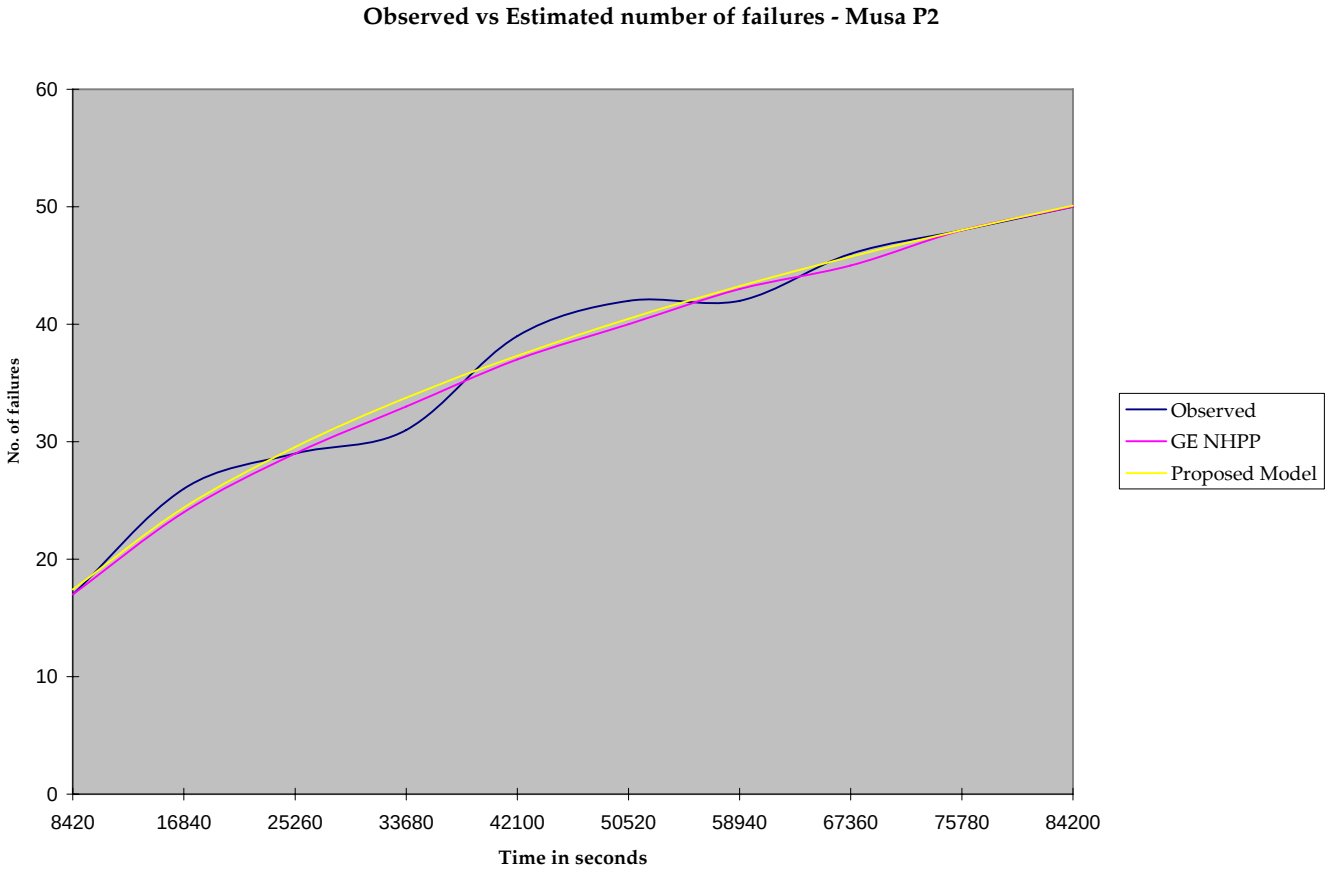

Figure 1

6. CASE STUDY: ONE APPLICATION OF THE MODEL

This case study confirms the predictive validity of the modified GE NHPP Model, and provides an example of determining when to stop testing. Musa's Project P1 was selected to establish the ability of the proposed model to replicate the past and predict the future. The parameters were estimated from the first twenty failures. The 20^{th} failure occurred at 1,986 seconds. The corresponding failure intensity was found to be 0.006 failures per second. The target failure intensity was fixed at 0.0025 failures per second. Then the total testing time needed, as well as the number of failures to be observed, for achieving the target was found, using a spreadsheet, as given below:

Testing time needed: 10,680 seconds; number of failures to be observed: 56

Actually, as per the Musa data set P1, the 54^{th} failure occurred at 10,625 seconds and the 55^{th} failure occurred at 11,175 seconds. Thus the target failure intensity has been achieved at the 55^{th} failure, and testing can be stopped after observing the 55^{th} failure, when the failure intensity has gone below 0.0025 failures per second. The above example establishes that the proposed model is able both to replicate past failures and to predict failures in the future accurately.

It is interesting to note that, in this project, it was predicted that a total of 200 faults would be detected, and that 186 failures would be observed at infinite time. Since we would stop testing on the occurrence of the 55th failure, only (55 x 1.08=) 60 faults would have been detected. This means that the product would be shipped with (200-60) 140 faults.

7. CONCLUSION

Although a host of SRGMs have been evolved in the last three decades, it is widely believed that no single model is suitable in all situations. Subburaj and Gopal proposed an NHPP model with the generalized exponential function ROCOF, which adequately fits software failure data from widely varying situations such as the learning phenomenon of the testing team; the imperfect, perfect, efficient software debugging phenomenon; and wide fluctuations in time between failures, either occurring alone or in combinations. It is found that the model possesses better predictive validity, and can be used in all projects. However, a software organization may need to collect a few vital quality metrics, such as degree of imperfect or efficient debugging, and the number of faults in the software product at the beginning as well as at the end of system testing. In order to facilitate the collection of such additional quality metrics, the authors propose a minor extension to the above model in this paper. The proposed extended model retains the excellent goodness of fit and predictive validity characteristics of the original model. This model also enables the prediction of reliability growth with testing, determining the testing time needed to achieve the required failure intensity objective, the number of failures to be observed before the target failure intensity will be achieved, and the number of faults remaining in the software product at the time of its release for use. Thus, it is a flexible model that addresses various patterns of debugging and testing; but at the same time it is not vulnerable to reasonable fluctuations in data, and above all it brings out a number of quality metrics in quantitative terms. The model parameters can be estimated through non-linear regression software tools. This model may be used no matter how the testing time is computed: execution time, testing time, or calendar time.

8. REFERENCES

[1] **Xie M.**, *A study of the effect of imperfect debugging on software development cost,* IEEE Transactions on Software Engineering, Vol.29, No.5, pp. 471-473, May 2003.

[2] **Ohba M. and Chou X.M.**, D*oes imperfect debugging affect software reliability growth?*, Proc. 11th Int'l Conf. Software Eng., pp. 237-244, 1989.

[3] **Kapur P.K. and Garg R.B.**, *Optimal software release policies for software reliability growth models under imperfect debugging*, Operations Research, Vol. 24, n 3, pp. 295-305, 1990.

[4] **Kapur P.K. and Garg R.B.**, *A software reliability growth model for an error-removal phenomenon*, Software Engineering Journal, Vol. 7(4), pp. 291-294, 1992.

[5] **Goel A.L.**, *Software reliability models: Assumptions, limitations and applicability*, IEEE Transactions on Software Engineering, SE-11 (12), pp.

1411-1423, 1985.

[6] **Musa J.D. and Ackerman A.F.**, *Quantifying software validation: When to stop testing?*, IEEE Software, pp.19-27, May 1989.

[7] **Yamada S., Ohba M. and Osaki S.,** *S-shaped reliability growth modeling for software error detection*, IEEE Transactions on Reliability, R-32(5), pp. 475-478, Dec. 1983.

[8] **Yamada S. and Osak**i **S.**, S*oftware reliability growth modeling: Models and applications*, IEEE Transactions on Software Engineering, Vol. SE-11(12), pp. 1431-1437, Dec. 1985.

[9] **Zhao M.and Xie M.**, *On the Log-Power NHPP Software Reliability model*, Proceedings of Third International Symposium on Software Reliability Engineering, pp. 14-22, Oct.1992.

[10] **Subburaj R. and Gopa G.** l, *Generalized Exponential Poisson Model for software reliability growth*, International Journal of Performability Engineering, Vol.2, No.3, pp. 291-301, July, 2006.

[11] **Gupta R.D. and Kundu D.**, *Generalized exponential distributions*, Austra. & New Zealand J.Statist., pp. 173-188, 41(2), 1999.

[12] **Pham H., Nordmann L. and Zhang X.**, *A general imperfect-software-debugging model with S-shaped fault-detection rate*, IEEE Transactions on Reliability, Vol.48, No.2, pp. 169-175, 1999.

[13] **Zhang X., Teng X. and Pham H.**, *Considering fault removal efficiency in software reliability assessment*, IEEE Transactions on Systems, Man and Cybernetics – part A: Systems and Humans, vol. 33 (1), pp.114-120, January 2003.

[14] **Zeephongsekul P., Xia G. and Kuma**r **S.**, *Software-reliability growth model: Primary-failures generate secondary-faults under imperfect debugging*, IEEE Transactions on Reliability, Vol. 43(3), pp. 408-413, 1994.

[15] **Farr W.**, *Chapter 3, Software Reliability Modeling Survey*, Handbook of Software Reliability Engineering, M.R. Lyu, editor, IEEE Computer Society Press and McGraw-Hill, 1996.

[16] **Kapur P.K., Garg R.B.and Kumar S.**, *Contributions to hardware and software reliability*, World Scientific Ltd., Singapore, 1999.

[17] **Mus**a, *DACS software reliability dataset*, Data & Analysis Center for Software, 1 **J.D.** 980. www.dacs.dtic.mil/databases/sled/swrel.shtml, downloaded on Sept. 19, 2005.

[18] **Huang C.Y. and Kuo S.Y.**, *Analysis of incorporating logistic testing – effort function into software reliability modeling*, IEEE Transactions on Reliability, Vol. 51, No.3, pp. 261-270, September 2002.

[19] **Chan F.C.L., Dasiewicz P.P., and Seviora R.E.,** *Metrics for evaluation of software reliability growth models*, International Symposium on Software Reliability Engineering, pp. 163-167, May, 1991.