A Study on the Failure Intensity of Different Software Faults

Kazuyuki Shima Shingo Takada Ken'ichi Matsumoto Koji Torii
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, Japan 630-01
+81-7437-2-5312
shima@is.aist-nara.ac.jp

ABSTRACT

We describe an experiment investigating the distribution of failure intensity in software reliability growth models. We found that the assumption of conventional models that the failure intensity follows a gamma distribution is not always true. Our new software reliability model does not make this assumption; rather, the failure intensity is calculated from failure data. We show that our new model predicts more accurately the number of detected faults for our study project than the conventional models.

Keywords

Failure intensity, testing, software reliability growth model, hyperexponential SRGM, Littlewood model, gamma distribution

INTRODUCTION

Software reliability growth model (SRGM) is a model which can be used for evaluating and/or estimating software reliability. It is plotted on a graph with the elapsed time on the X-axis and the cumulative number of detected software faults on the Y-axis. The elapsed time may be either calendar time or computer execution time, but normally calendar time is used. Thus, SRGM is a formalization of the characteristics of this growth curve and can be used to estimate the number of remaining faults from data on detected faults and the time needed to detect these remaining faults.

There are many types of SRGMs with different assumptions and preconditions. Some examples are: exponential SRGM [2], delayed S-shaped SRGM [7], hyperexponential SRGM [4], and connective exponential SRGM [5]. When these models are applied to actual software development projects, the corresponding assumptions and preconditions must apply to the target project.

ICSE 97 Boston MA USA

In exponential SRGM, there is an assumption that the failure intensity of all faults in a software, i.e. the probability of a fault actually causing failure, are the same. This, however, has been found to be questionable from an experiment at NASA [6].

Littlewood has proposed a model which assumes that if faults in a software are chosen at random, the failure intensity of the faults would have a probability distribution such as the gamma distribution [3]. Although gamma distribution has the characteristics of being easy to handle from a mathematical standpoint, no evidence has been shown that failure intensity actually takes the shape of gamma distribution.

This paper shows a method for measuring the failure intensity of software faults and applies it to a small program. Furthermore, based on the results, we propose a model where the failure intensity of faults have discrete distribution, and evaluate the accuracy of the estimation of cumulative number of faults. Hyperexponential SRGM is an example of using discrete distribution as the distribution of failure intensity. However, in hyperexponential SRGM, the failure intensity of the same module is assumed to be the same. In our proposed model, it is possible for the failure intensity of the same module to take different values.

FAILURE INTENSITY OF FAULTS

Software failure is the phenomena where software does not function properly in terms of its requirements. Faults are the defects found within a program that causes one or more failures. The occurrence of a failure depends on conditions such as input data. Just because there is a fault, it doesn't mean that it will cause a failure. So, we can think of the frequency of a fault causing a failure. We define this frequency of fault F causing a failure (average number of times per time unit) as the failure intensity b_F of fault F.

If we define the probability of fault F causing a failure in a certain time unit as ϕ_F, the average time interval of fault F causing a failure can be shown as follows[1]:

$$\frac{1}{b_F} = \int_0^\infty (1-\phi_F)^t dt \qquad (1)$$

$$= \frac{-1}{\log(1-\phi_F)} \quad (2)$$

So, the following relation holds:

$$b_F = -\log(1-\phi_F) \quad (3)$$

Generally, different faults appear in different places within the software, and the type of input data that causes failure differs between faults. In other words, the failure intensity of each fault is different. Furthermore, it is not known what sort of fault exists in the software.

Therefore, when randomly choosing faults within software, we can consider the failure intensity of the chosen fault to be a random variable. The probability distribution is called the failure intensity distribution of the fault. In the rest of this paper, we will be referring to this when we say failure intensity distribution.

FAILURE INTENSITY IN CONVENTIONAL SRGM

This section will describe the following five conventional SRGMs with respect to failure intensity:

Exponential SRGM A model which assumes that the failure intensity of all faults are the same.

S-shaped SRGM A model where the failure intensity changes with time.

Hyperexponential SRGM A model which assumes that the failure intensity of faults within the same module are the same.

Connective Exponential SRGM A model which assumes that the failure intensity of all faults are the same.

Littlewood model A model where the failure intensity takes a certain distribution.

Exponential SRGM

Exponential SRGM assumes that the number of faults found at time t is proportionate to the number of faults left in the software. This means that the probability of the failures for faults actually occurring, i.e. being found, is constant. This can be shown as the following differential equation:

$$\frac{dH(t)}{dt} = b(a - H(t)) \quad (4)$$

where

a: the expected total number of faults that exist in a software before testing

b: the failure intensity of a fault

$H(t)$: the expected number of faults at time t

This can be solved as follows:

$$H(t) = a(1 - e^{-bt}) \quad (5)$$

b is also called the detection rate, the rate of incident or the appearance rate, but this paper will refer to this as failure intensity.

S-shaped SRGM

In the delayed S-shaped SRGM, the reliability growth curve is an "S" curve. The detection rate of faults become greatest at a certain amount of time after testing begins, after which it decreases exponentially. So, since the detection rate of faults as well as the remaining number of faults changes with time, we can say that this SRGM assumes that the failure intensity differs among faults.

Hyperexponential SRGM

Hyperexponential SRGM extends exponential SRGM by dividing software into modules using 0 such as new parts and reused parts. The failure intensity of faults within different modules are assumed to be different, while the failure intensity of faults within the same module are assumed to be the same. The expected number of faults detected for each module are exponential. So, the expected number of faults detected for the software as a whole can be expressed as follows:

$$H(t) = a\sum_{i=1}^{n} p_i(1 - e^{-b_i t}) \quad (6)$$

where

$$a > 0 \quad , \quad 0 < p_i < 1, \quad (7)$$

$$\sum_{i=1}^{n} p_i = 1 \quad (8)$$

and

a: the expected total number of faults existing in software before testing

b_i: the failure intensity of one fault within the ith module

p_i: the density of faults for the ith module

Connective Exponential SRGM

Nakagawa analyzed the characteristics of the growth curve using the number of faults detected per time unit.

From his analysis, he postulated that the basic shape of the growth curve is exponential, and that an "S"-curve forms due to the test. Based on this, he proposed the Connective Exponential SRGM. In this model, a group of modules called "main route module" are first tested, and then the rest of the modules are tested. Even if the failure intensity of the faults in the main route module and faults in the other modules are the same, since the search for their detection starts at different points in time, the growth curve becomes an "S"-curve. The expected number of faults detected for the software as a whole can be expressed as follows:

$$\begin{aligned} H(t) &= a_1[1 - exp(-bt)] \\ &+ a_2[1 - exp(-by)] \end{aligned} \tag{9}$$

where

$$a_2 \gg a_1 > 0, b > 0, \tag{10}$$

$$y = \begin{cases} 0 & t > t_0 \\ t - t_0 & t \geq t_0 \end{cases} \tag{11}$$

and

a_1: The number of faults that are expected to be detected in the main route module

a_2: The number of faults that are expected to be detected in modules other than the main route module

b: The failure intensity

t_0: The starting time for testing modules other than the main route module

Littlewood Model

Littlewood proposed a model which handles the time interval between the occurrences of failures as random variable, and focused on hazard rates. The hazard rate is a probability density where software does not fail for a certain time span and in the next moment fails. The probability density function $pdf(T_i|\lambda_i)$ of random variable T_i which shows the time interval between detection of the $(i-1)$ th and i th failure is assumed to be:

$$pdf(T_i|\lambda_i) = \lambda_i e^{-\lambda_i T_i}$$

where λ_i is the software's hazard rate between the $(i-1)$ th and i th failure, and is assumed to be the sum of the failure intensity of the faults remaining in the software (equation(15)). The model assumes that generally the failure intensity of faults ν_j differs between each fault, and that it is a random variable which conforms to the following probability density function:

$$pdf(\nu) = \frac{e^{-\nu \sum_1^{i-1} t_j} f(\nu)}{\int_0^\infty e^{-\nu \sum_1^{i-1} t_j} f(\nu) d\nu}$$

$f(\nu)$ is the distribution function of hazard rate ν, and for example takes the gamma distribution. Gamma distribution takes two parameters α, β, and can be represented as follows:

$$f(\nu_j) = \frac{\beta^\alpha}{\Gamma(\alpha)} \nu_j^{\alpha-1} e^{-\beta \nu_j} \tag{12}$$

The model is represented as follows:

$$F_i(x) = 1 - \left(\frac{\beta + \sum_{j=1}^{i-1} t_j}{\beta + \sum_{j=1}^{i-1} t_j + x} \right)^{(N-i+1)\alpha} \tag{13}$$

$$f_i(x) = \frac{(N-i+1)\alpha \left(\beta + \sum_{j=1}^{i-1} t_j \right)^{(N-i+1)\alpha}}{\left(\beta + \sum_{j=1}^{i-1} t_j + x \right)^{(N-i+1)\alpha+1}} \tag{14}$$

$$\lambda_i = \sum_{j=1}^{N-i+1} \nu_j \tag{15}$$

where,

$F_i(x)$: The probability of a software failing within time x after the $(i-1)$th failure

$f_i(x)$: The probability density function of $F_i(x)$

N: The total number of faults that exist within the software before testing

t_j: The measured time interval between the occurrence of the $(j-1)$th failure and jth failure

λ_i: The hazard rate when the $(i-1)$th failure occurred

ν_j: The failure intensity of the jth fault remaining in the software

α, β: The parameters for the failure intensity distribution (Gamma distribution)

FAILURE INTENSITY EXPERIMENT

The Gamma distribution which is used as the failure intensity distribution in Littlewood's model is often used as a continuous distribution in Bayes statistics, and is also often used as the Bayesian model for software reliability models. However, no experiments have been done to confirm or refute if in fact actual failure intensity conforms to the Gamma distribution. This section reports on such an experiment and gives its results.

Experiment Goal and Hypothesis

An experiment has been carried out to find out if the failure intensity in actual software conforms to the failure intensity assumed in conventional SRGMs. The hypothesis of the experiment is as follows:

H1 The failure intensity of faults in the software are the same.

H2 The failure intensity of faults in the same module are the same.

H3 The failure intensity conforms to the Gamma distribution.

Experiment Strategy

The experiment is carried out in the following three steps:

1. Test and debug the target program.

   We first test and debug the target program using test data T. The target program does not include any errors which result in compilation errors. During this step, the fault and modifications to the program are recorded. This will result in a completed program P with little or no faults, and reports for faults $F_1, F_2, \cdots F_n$.

2. Embed faults.

   In order to check the failure intensity of each fault, we use the debugged program and fault reports that were obtained in step 1. Each fault is embedded once into one version of the debugged program, resulting in n versions of the program each having one fault.

3. Measure the failure intensity.

   Testing is done on each program, that was obtained in step 2, containing one fault. The number of times failure occurs within a time unit is checked, and we calculate the failure intensity of each fault $b_{F_1}, b_{F_2}, \cdots, b_{F_n}$.

   Failure intensity is also affected by the frequency of input (or more precisely, the number of inputs per time unit). However, when this is not available, we make a simple assumption where the frequency is 1. The probability of fault F causing a failure in a time unit can then be shown as follows:

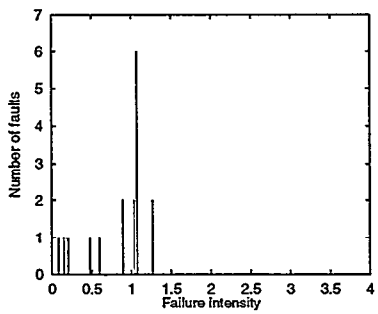

Figure 1: Failure Intensity of Faults (Program 1)

$$\phi_F = \frac{N_F}{N} \tag{16}$$

where N_F is the number of possible inputs that results in F causing a failure and N is the total number of possible inputs.

In reality, the total possible number of inputs is too large, making it very difficult to test all possible inputs. In such cases, testing is done on as much input as possible, and the equation is approximated as follows:

$$\phi_F = \frac{N'_F}{N'} \tag{17}$$

where N'_F is the number of test cases that results in F causing a failure and N' is the total number of test cases.

In this way, the probability of each fault causing a failure within a time interval $\phi_{F_1}, \phi_{F_2}, \cdots \phi_{F_n}$ can be calculated, and through equation (3), the failure intensity $b_{F_1}, b_{F_2}, \cdots b_{F_n}$ can be obtained.

Experimental Design and Results

Four programs were used for this experiment. Three of them had the same requirements (A: program for stock management), and the other one (B) was a subprogram for numerical analysis. The size of the programs were each about 300 lines.

To avoid failure happening on purpose, test data was randomly generated. 100 such sets for each program were used in step 1 (test and debug). This resulted in 21, 13, 13, and 25 faults for each of the programs[1]. Then, steps 2 and 3 were carried out to calculate the failure intensity distribution of faults for each program.

[1] Note that some of the data sets may not cause a failure, so the number of test data is greater than the number of faults found.

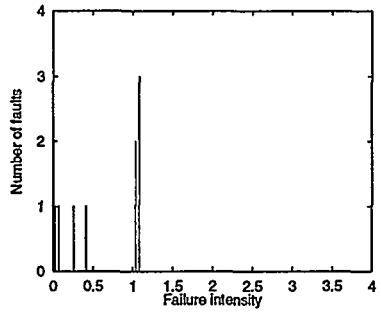

Figure 2: Failure Intensity of Faults (Program 2)

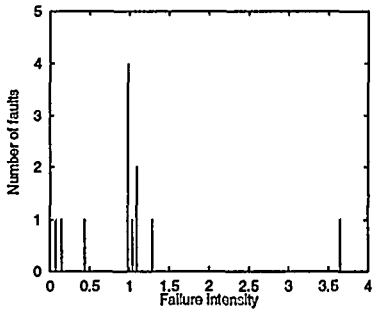

Figure 3: Failure Intensity of Faults (Program 3)

The resulting distribution for each program are given in Figures 1 to 4.

We now turn to threats to external validity. Such threats limit our ability to generalize the results of our experiment to other real programs. We identified three such threats:

- The subjects carrying out step 1 in our experiment, i.e. testing and debugging, may not be representative of professional software programmers. The subjects were undergraduate level students, and none had any industrial experience.
- The target program may not be representative of industrial software. The program we used was small, each about 300 lines long.
- The testing process in our experiment may not be representative of processes used in industry. Our test data was generated randomly, which would normally not be used in industry.

Although such threats exist, we feel that our experiment is a necessary step towards finding out about the hypothesis we have stated. Of course, replication of our experiment would greatly reduce the risk of these threats, and hence would confirm our analysis results.

Analysis of Hypothesis H1
First we check the first hypothesis, i.e. "the failure intensity of faults in the software are the same."

The failure intensity was clearly different between faults. This means that there is a problem if the failure intensity of faults only has one parameter. On the other hand, there are plenty of examples where the failure intensity of different faults are the same, and the failure intensity distribution can be seen as a discrete distribution. The fact that the target program was small may have enhanced this trend, but we think that this can be said for most software since this is caused by the control structure of computer software.

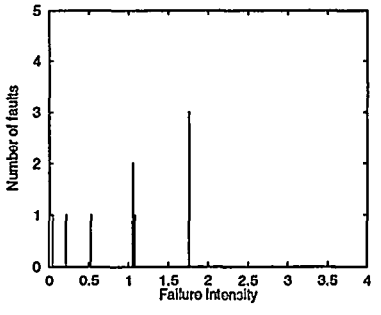

Figure 4: Failure Intensity of Faults (Program 4)

The control structure that is used most often in current software development is procedural. Procedural control structure is composed of sequential construct, conditional branching, and looping. In conditional branching, one of two possible branches are chosen depending on the condition. Suppose a fault resides in both branches and that these faults have the same conditional probability for causing a failure. If the probability of executing either of the branches is the same, then the failure intensity of the faults in each branch would also be the same. But in most cases, the probability of the branch being executed differs. If this is so, the failure intensity of the faults in each branch would also differ. So, we can say that the failure intensity for different faults will not always be the same.

Next, we take an example where there are two faults in a sequential construct that does not have a conditional branch. We assume that failure always occurs when executing this sequential construct. So, the failure intensity for these two faults are the same. Thus, there exists cases where the failure intensity of two faults do not differ. Since no two cases are the same in an ideal continuous distribution, this means that the failure intensity distribution is *approximately* continuous, i.e. in reality it is discrete.

Analysis of Hypothesis H2

Next, we check the second hypothesis, i.e. "the failure intensity of faults in the same module are the same."

Hyperexponential SRGM assumes that the failure intensity for each module differs, but that the failure intensity for different faults within the same module are the same. Although our experiment results are only for a small program consisting of one module, we could not conclude the failure intensity of different faults to be the same. If the program was larger, since there would be more faults, it would become very difficult to assume that the failure intensity are the same.

Analysis of Hypothesis H3

We now check the third hypothesis, i.e. "the failure intensity of faults conform to the Gamma distribution." The parameters for Gamma distribution are first estimated using maximum likelihood estimation. Assuming the measured failure intensity data are $x_1, x_2, \cdots, x_n$, then parameter α can be obtained using the following equation:

$$\frac{\Gamma(\alpha)'}{\Gamma(\alpha)} - \log\alpha = \frac{1}{n}\log\prod_{k=1}^{n} x_k - \log\frac{\sum_{k=1}^{n} x_k}{n} \quad (18)$$

Then, parameter β can be obtained by:

$$\beta = \frac{n\alpha}{x_1 + x_2 + \cdots + x_n} \quad (19)$$

There are cases where a fault may cause failure for any input which would result in the failure intensity to be infinity. This can be problematic. Such cases can be handled in one of two ways:

Method 1 Instead of the failure intensity being infinity, give it a very large number (such as 10000). This will allow us to check the effects of faults whose failure intensity is infinity.

Method 2 Ignore faults whose failure intensity is infinity. Faults whose failure intensity is large can be detected easily, making the possibility of them remaining to be low. So, such faults can be ignored even if they are detected, and the cumulative number of faults is not increased.

We now test if the Gamma distribution using the above estimated parameters matches actual data. We use Kolmogorov-Smirnov test of goodness of fit.

We calculated the parameters for the Gamma distribution and the Kolmogorov distance for the four programs, and then tested at the significance level of 2.5 %. Programs 1 and 3 both did not have any failure intensity of infinity, so there was no need to apply the above methods 1 or 2. Table 1 shows the results for programs 1 and 3.

Table 1: Goodness of Fit of Gamma Distribution to Failure Intensity (Programs 1 and 3)

Program	prog1	prog3
Parameter α	0.6499	2.0711
Parameter β	0.1072	0.4324
Kolmogorov distance	0.2449	0.3886
result of test	accept	reject

Table 2: Goodness of Fit of Gamma Distribution to Failure Intensity (Programs 2 and 4)

Program	prog2	prog4
Parameter α	0.6499	0.1036
Parameter β	0.1072	4.351×10^{-5}
Kolmogorov distance	0.2449	0.5152
result of test	accept	reject

Programs 2 and 4 both contained faults whose failure intensity was infinity, so methods 1 and 2 were applied for parameter estimation and test of goodness of fit. Table 2 shows the results using method 1, and table 3 shows the results using method 2.

We can summarize our findings as follows:

- The failure intensity distribution can be considered as discrete distribution, because there are many cases where different faults have the same failure intensity.
- It was not possible to conclude whether or not the distribution was Gamma distribution from the shape of the distribution resulting from the measurement. There are both accepted cases and rejected cases on test of goodness of fit of Gamma distribution.
- Even for small programs (modules), it can be said that cases where all faults have the same failure intensity are rare.

PROPOSED MODEL

Table 3: Goodness of Fit of Gamma Distribution to Failure Intensity Using Method 2 (Programs 2 and 4)

Program	prog2	prog4
Parameter α	0.2866	0.3888
Parameter β	0.6250	1.3123
Kolmogorov distance	0.1637	0.1470
result of test	accept	accept

The Model

Although hyperexponential SRGM only assumes that the failure intensity differs between modules, it still has the assumption that different faults will have different failure intensity. Also, it does not consider the failure intensity distribution to be continuous distribution. Rather it considers it to be discrete distribution which matches our experiment results. So, we extend the hyperexponential SRGM so that each fault will have different failure intensity.

$$H(t) = \sum_{i=1}^{n} a_i(1 - e^{-b_i y_i}) \quad (20)$$

$$y_i = \begin{cases} 0, & t < t_i \\ t - t_i, & t \geq t_i \end{cases} \quad (21)$$

where,

$H(t)$ The expected number of faults to be detected by time t

n The number of times that testing was done to debug all faults

b_i The failure intensity of a fault

t_i The time when faults were introduced

a_i The number of faults which were introduced at time t_i with failure intensity b_i

This is similar to hyperexponential SRGM, except definitions in hyperexponential SRGM for b_i, t_i, and initial number of faults are based on modules. Our proposed model assumes that even the failure intensity of faults in the same module are different. So, we cannot use the method for determining parameters by identifying faults for each module as was the case in hyperexponential SRGM. Also, it is not possible to apply our experiment method of embedding faults and testing in an actual development environment.

Estimating Parameters

We now propose a method which will allow an estimation of the necessary parameters in our model from data concerning detection time for faults and the number of detected faults. The parameters are determined so that the estimated data fits the observed data, so they do not show the actual time when faults were introduced and the actual failure intensity of faults. This, however, can also be said for parameters in conventional SRGMs, and is essential to make actual use of the model.

Since our model has many parameters, numerical analysis is needed for estimating them. But, we do not use maximum likelihood estimation since the calculation cost will become very large. We will instead use an estimation method based on an algorithm that is simple and needs little computation.

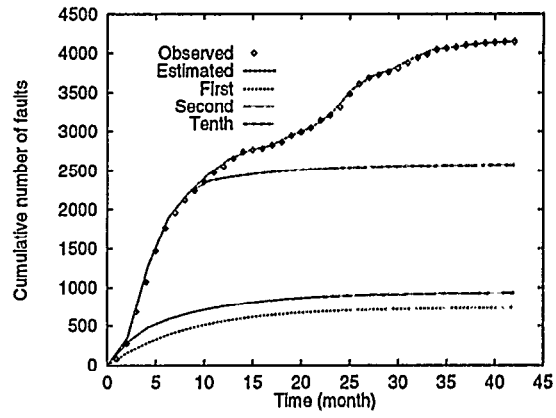

Figure 5: Estimation of Reliability Growth Curve Based on the Proposed Model

The failure intensity and initial number of faults are determined by taking faults in the order in which they were introduced. To determine the parameters, the following two procedures are repeated as one step for every time point that was deemed to have been the time when a fault was introduced:

- The time when faults were introduced is determined, and only such faults are assumed to be detected up to that time.
- Under the constraint that the cumulative number of further faults do not go over the total cumulative number of faults, the failure intensity is determined.

We now show an example of how our model is applied. Figure 5 shows the result of applying our proposed method in determining the time when faults were introduced and failure intensity of faults. The X-axis shows the time (with months as the unit), and the Y-axis shows the number of detected faults. The "Observed" points show the actual measured values, and the "Estimated" points show the values estimated from the proposed model. "First" shows the results of the first step in the parameter estimation method. By "step" we mean the step noted previously to determine the parameters, or i in equation 20 and 21. Since there is a constraint where the curve must not go over the actual value, the estimated value is far below the measured value. "Second" shows the results of the second step. The number of faults from the second step is actually added on to the number of faults from the first step. By the tenth step ("Tenth"), the estimated value gets quite

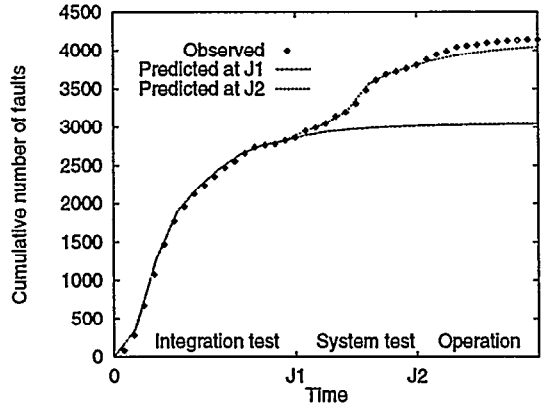

Figure 6: Estimation of the Cumulative Number of Faults Based on the Proposed Model

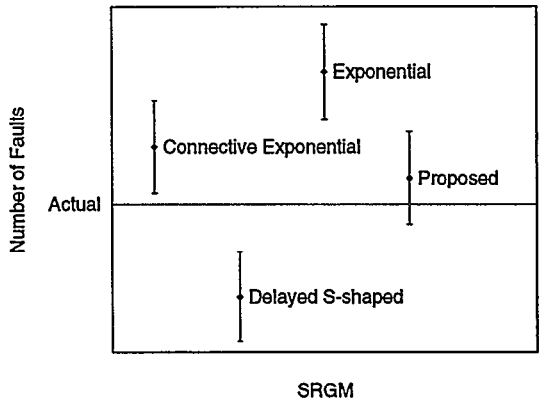

Figure 7: Comparison of the Estimation Precision of the Number of Cumulative Faults (Actual value = 4142)

near to the actual value. This is repeated until the 20th step which is "Estimated" in the figure.

Application of Proposed Model

Figure 6 shows the results of estimating actual fault data [5] based on our proposed model. The X-axis is time, and the Y-axis is the number of detected faults. J1 denotes the end of integration testing, and J2 denotes the end of system testing. The integration testing occurs between 0 and J1, system testing between J1 and J2, and operation is from J2. "Observed" points are the actual values. "Predicted at J1" and "Predicted at J2" show the estimated results at the end of integration testing and at the end of system testing, respectively. As can be seen, the prediction at J2 is very near to the final actual number of faults. But, this is not so for the prediction at J1. Our model does not take into account differences in phases. So, for the prediction at J1, it did not take into account that system testing may result in an increased rate in detecting the number of faults. This is one shortcoming of our model, and we need to address this in the future.

Figure 7 also uses the data in [5], and shows a comparison between the results for our proposed model and other conventional SRGMs (exponential, delayed S-shaped, connective exponential) at the end of system testing. The 90% confidence bounds for each estimation is shown by the vertical line protruding from each estimate. For example, the confidence bound for our proposed model shows that it would be between 4097 and 4311. Since our model was the only one whose confidence bound included the actual value, we can conclude that our model was more accurate than the other models for the data that was used.

Figure 7 does not show the results for hyperexponential SRGM and Littlewood Model. In order to estimate hyperexponential SRGM, we would need data concerning the classification of the modules which contains faults, and their number. As for Littlewood model, it is an interfailure time model and we would need data on the time that the failure 0. Since we did not have such data, we did not include hyperexponential SRGM and Littlewood Model.

CONCLUSION

This paper examined the failure intensity of different faults, and proposed an SRGM which assumes that the failure intensity distribution is discrete. We also showed a method for estimating the parameters in the method. Conventional SRGMs were based on assumptions such as the failure intensity of faults within the same software/module are the same, or that the failure intensity distribution takes the form of Gamma distribution. Our proposed model allows for different failure intensities for different faults, and assumes that its distribution takes the form of discrete distribution.

We used a small program consisting of one module, and through the examination of the failure intensity for each fault, we determined that the failure intensity is not the same, and that its distribution does not necessarily take the form of Gamma distribution. Our proposed model is based on these results. We also compared the accuracy of estimating fault data using our model and conventional SRGMs (exponential, delayed S-shaped, connective exponential). In the examples used in our comparison, our model was found to be more accurate in estimating the the number of faults than the more conventional models.

For future work, we plan on incorporating the merits of other SRGMs so that we can achieve an even higher accuracy in estimating fault data. Furthermore, we need to take into account how changes in phases, such as

going from integration testing to system testing, may affect the estimation.

Acknowledgment
We are grateful to Larry Votta and the reviewers of the paper for their valuable comments which have improved the quality of the paper and our future works.

REFERENCES

[1] Richard E. Barlow and Frank Proschan, "Mathematical Theory of Reliability," John Wiley & Sons publishers, pp. 65, 1965.

[2] A. L. Goel and K. Okumoto. Time-dependent error-detection rate model for software reliability and other performance measures. IEEE Trans. Rel., R-28, 3, pp. 206–211, Aug. 1979.

[3] Littlewood, B. : "How good are they and how can they be improved ?", *IEEE Trans. Software Eng.*, SE-6, 5, pp.489-500, (Sept. 1980).

[4] K. Matsumoto, K. Inoue, T. Kikuno, and K. Torii. Experimental evaluation of software reliability growth models. Proc. of 18th FTCS, pp. 148-153, 1988.

[5] Y. Nakagawa. A connective exponential software reliability growth model based on analysis of software reliability growth curves. IEICE Trans., vol. J77-D-I, no. 6, pp. 433–442, June 1994.

[6] J. R. Dunham. Software errors in experimental systems having ultra-reliable requirements. Digest of FTCS-16, pp. 158–163, June 1986.

[7] S. Yamada, M. Ohba, and S. Osaki. S-shaped reliability growth modeling for software error detection. IEEE Trans. rel., R-32, 5, pp. 475–478, Dec. 1983.