Effort-Index-Based Software Reliability Growth Models and Performance Assessment

Chin-Yu Huang*, Sy-Yen Kuo*, and Michael R. Lyu**
*Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
sykuo@cc.ee.ntu.edu.tw
**Computer Science & Engineering Department
The Chinese University of Hong Kong
Shatin, Hong Kong
lyu@cse.cuhk.edu.hk

Abstract

In this paper, we first show that the logistic testing-effort function is practically acceptable/helpful for modeling software reliability growth and providing a reasonable description of resource consumption. Therefore, in addition to the exponential-shaped models, we will integrate the logistic testing-effort function into S-shaped model for further analysis. The model is designated as the Yamada Delayed S-shaped model. A realistic failure data set is used in the experiments to demonstrate the estimation procedures and results. Furthermore, the analysis of the proposed model under imperfect debugging environment is investigated. In fact, from these experimental results and discussions, it is apparent that the logistic testing-effort function is well suitable for making estimations of resource consumption during the software development/testing phase.

1. Introduction

The complexity and size of a computer system have grown dramatically for the past two decades. The growing trend of software criticality has generated more researches into the field of high-quality software development. In highly complex modern software systems, reliability is the most important factor since it quantifies software failures during the process of software development and software quality control. Software reliability is the probability that a given software will be functioning without failure in a given environment during a specified period of time [1-3]. A common approach for measuring software reliability is by using an analytical model whose parameters are generally estimated from available data on software failures [1-9]. In the field of software reliability modeling, the span of time may be considered as calendar time, clock time, and execution time. Musa et al. [2-3] and Ohba [6] showed that the effort index or the execution time is a better time domain for software reliability modeling than the calendar time because the shape of observed reliability growth curve depends strongly on the time distribution of the testing-effort. Unfortunately, most papers assumed that the consumption rate of testing resource expenditures during the testing phase is a constant or even do not consider such testing effort. If the effort index data or execution-time-based data are available in the actual observed data set, software reliability models should be developed by incorporating the testing-effort functions in real development environment [2-3, 6-7, 14-16]. In this paper, we will first review a SRGM based on the *Non-Homogeneous Poisson Process* (NHPP) which incorporates a logistic testing-effort function. We show some equations to describe the mathematical properties of this model. Besides, in order to demonstrate the applicability/superiority of the logistic testing-effort function and make fair comparisons with other conventional software reliability growth models, we thus try to integrate the logistic testing-effort function into the S-shaped model, particularly the Yamada S-shaped software reliability growth model. Experiments have been performed based on real test/debug data sets and the results show that the proposed SRGM with logistic testing-effort function is a simple and compact model and can estimate the number of initial faults better than previous models. In addition, the analysis of the proposed model under imperfect debugging environment is also investigated. From these experimental results and discussions, it is apparent that the extended model to incorporate imperfect debugging is isomorphic to the original model we proposed in this paper.

In the remaining of the paper, there are four more sections. We investigate how to incorporate logistic testing-effort function into the exponential-shaped and the S-shaped software reliability growth modeling in section

0-7695-0792-1/00 $10.00 © 2000 IEEE

2. The applications of these models to actual test/debug data set are discussed in section 3. Besides, the imperfect-debugging problem based on the proposed model is discussed in section 4. Finally, section 5 gives some concluding remarks on the results obtained.

2. Software reliability growth modeling and testing-effort function

2.1 SRGM with logistic testing-effort function

In this section, we will first review logistic testing-effort functions. During the software-testing phase, significant test-effort is required, such as the number of test cases, human power, and CPU time. As usual, the test-effort during the testing phase and the time-dependent behavior of development effort in the software development process can be described by a Weibull-type consumption curve, see Yamada et al. [10-13]. In fact, since actual testing-effort data represent various expenditure patterns, sometimes the testing-effort expenditures are difficult to be described by only an exponential or a Rayleigh curve. Although the Weibull-type curve can fit the data well under general software development environment, however, it will have an extreme peak work rate when the shape parameter $m \geq 3$. That is, when m=3, 4, and 5, we can find that these testing-effort curves have an apparent peak point (i.e. non-smoothly increasing and degrading consumption curve) during the software development process [14-16]. This fact seems not so realistic because it is not commonly used to interpret the actual software development/test process and may not be suitable for modeling the test effort consumption curve. Therefore, we will use a logistic testing-effort function instead of the Weibull-type testing-effort consumption function as the testing-effort function to describe the test effort patterns during the software development process. This function differs from the Weibull-type function described in the above subsection and was used to derive the resource consumption curve of a software project over its life cycle and predict the future costs/schedules [14-16]. It is a dynamic model since the resource consumption is estimated from a set of variables that are interdependent. Besides, DeMarco also reported that this function was fairly accurate in the Yourdon 1978-1980 project survey [17]. The cumulative testing-effort consumption of logistic testing-effort function in time $(0, t]$ is depicted in the following:

$$W(t) = \frac{N}{1 + A\exp[-\alpha t]} \quad (1)$$

where N is the total amount of testing effort to be eventually consumed, α is the consumption rate of testing-effort expenditures, and A is a constant.

Therefore, the current testing-effort expenditures at testing time t are given by

$$w(t) = \frac{NA\alpha \times \exp[-\alpha t]}{\left(1 + A\exp[-\alpha t]\right)^2} = \frac{NA\alpha}{(\exp[\frac{\alpha t}{2}] + A\exp[-\frac{\alpha t}{2}])^2} \quad (2)$$

We can see that the $w(t)$ is a smooth bell-shaped function and its left side is tailed. Besides, the $w(t)$ reaches its maximum value at time

$$t_{\max} = \frac{\ln A}{\alpha} \quad (3)$$

This basic SRGM is based on the following assumptions:

1. The fault removal process follows the Non-Homogeneous Poisson Process (NHPP).
2. The software system is subject to failures at random times caused by faults remaining in the system.
3. The mean number of faults detected in the time interval $(t, t+\Delta t]$ by the current test-effort is proportional to the mean number of remaining faults in the system.
4. The proportionality is a constant over time.
5. The consumption curve of testing effort is modeled by a logistic testing-effort function.
6. Each time a failure occurs, the fault that caused it is immediately removed, and no new faults are introduced.

Therefore, we can describe the mathematical expression of a testing-effort-based as the following:

$$\frac{dm(t)}{dt} \times \frac{1}{w(t)} = r \times [a - m(t)], \; a>0, \; 0<r<1 \quad (4)$$

that is,

$$\frac{dm(t)}{dt} = w(t) \times r \times [a - m(t)] \quad (5)$$

Solving the above differential equation under the boundary condition $m(0)$=0 (i.e., the mean value function $m(t)$ must be equal to zero at time 0), we have

$$m(t) = a(1 - e^{-r(W(t)-W(0))}) = a(1 - e^{-rW^*(t)}) \quad (6)$$

where $m(t)$ is the expected mean number of faults detected in time $(0, t]$, $w(t)$ is current testing-effort consumption at time t, a is the expected number of initial faults, and r is error detection rate per unit testing-effort at testing time t that satisfies $r>0$.

2.2 Yamada S-shaped model with logistic testing-effort function

The Delayed S-shaped SRGM was originally proposed by Yamada et at. [6-7, 12-13] and was a simple modification of the NHPP to obtain an S-shaped growth

curve for the cumulative number of failures detected. This model's software fault detection process can be viewed as a learning process that the software testers become familiar with the testing environments and tools as time progresses, these testers' skills gradually improve and then level off as the residual faults become more difficult to uncover [1, 6-7, 12-13]. Because the original S-shaped model is for the analysis of fault isolation data, i.e. the testing process contains not only a fault detection process, but also a fault isolation process. Following the similar steps described in subsection 2.1, we can get the following relationship between $m(t)$ and $w(t)$ for an extended Yamada S-shaped software reliability model:

$$\frac{df(t)}{dt} \times \frac{1}{w(t)} = \varpi \times [a - f(t)] \quad (7)$$

$$\frac{dg(t)}{dt} \times \frac{1}{w(t)} = \varepsilon \times [f(t) - g(t)] \quad (8)$$

where $f(t)$ is the cumulative number of failures detected up to t and $g(t)$ is the cumulative number of failures isolated up to t.

Solving Eq. (7) and Eq. (8) under the boundary condition $f(0) = g(0) = 0$, we have

$$f(t) = a(1 - e^{-\varpi(W(t)-W(0))}) = a(1 - e^{-\varpi W^*(t)}) \quad (9)$$

where ϖ is the failure detection rate.
And

$$g(t) = a[1 - \frac{1}{\varpi - \varepsilon}(\varpi e^{-\varepsilon W^*(t)} - \varepsilon e^{-\varpi W^*(t)})] \quad (10)$$

where ε is the failure isolation rate.

By assuming the fault detection rate parameter the fault isolation rate parameter (ε) $\cong$ the fault detection rate parameter (ϖ), the NHPP model with a Delayed S-shaped growth curve of detected software faults is:

$$m(t) = a(1 - (1 + \psi(W(t) - W(0)) \times e^{-\psi(W(t)-W(0))})$$

$$= a(1 - (1 + \psi W^*(t)) \times e^{-\psi W^*(t)}) \quad (11)$$

where ψ = fault detection rate per unit testing-effort at testing time t that satisfies $\psi > 0$.

3. Numerical examples

The data set analyzed here is from a study by Ohba [6]. The total cumulative number of detected faults after a long period of testing is 358 and this value can be used as an additional comparison criterion. Through using the methods of MLE and LSE, these estimated parameters of the logistic testing-effort function are N=54.8364, A=13.0334, and α=0.226337. Table 1 shows the estimated parameters of Eq. (6), comparisons with the estimated initial faults a, and MSF of other general SRGMs. In addition, the testing effort function reaches the maximum at time t=11.3438 weeks which corresponds to $w(t)$= 3.10288 CPU hours and $W(t)$=23.5107 CPU hours. Furthermore, the number of faults removed up to this time t_{max} is 245.421. Fig. 1(a) graphically shows the actual fitting number of software faults, the mean value function of Eq. (6), and the 90% upper and lower confidence bounds vs. test time. It shows that the variation in $m(t)$ constantly increases with time t and eventually becomes a constant. The observed failure data and the fitted curve for the extended S-shaped model with logistic testing-effort function are plotted in Fig. 1(b). From these figures, the testers can use the results to estimate the number of additional tests to run and the additional amount of resources needed to reach the given objective, and such information are more useful than those based on historical data. Besides, from the information provided by Fig. 1(a), (b) and Table 1, we can see that the model in Eq. (6) gives a better fit to the observed data than the Delayed S-shaped model with logistic testing-effort function does. Moreover, according to the study in [6], even the Delayed S-shaped model does not fit the observed data well when the testing effort spent on failure detection/isolation is not a constant. But by integrating the testing-effort function we proposed into the Delayed S-shaped software reliability model, we can find that the extended Delayed S-shaped model with logistic testing-effort function has smaller AE and MSF than with Rayleigh testing-effort functions from Table 1. From these figures and comparison results, we can conclude that the advantage of logistic testing-effort function is the applicability to various kinds of models and it can yield better predictions for other reliability metrics. It means that the logistic testing-effort function we proposed is really good enough to give a more accurate description of resource consumption during the software development phase.

Table 1: Comparison results.

Model	a	r (or ψ)	AE (%)	MSF
SRGM with Logistic TEF	394.076	0.0427223	10.06	118.29
SRGM with Rayleigh TEF	333.18	0.100415	6.93	798.49
Yamada Delayed S-Shaped Model	374.05	0.197651	4.48	368.67
Delayed S model with Logistic TEF	338.136	0.10004	5.54	242.79
Delayed S model with Rayleigh TEF	459.08	0.0273367	28.23	268.42
G-O Model	760.00	0.0322688	112.29	139.82
Inflection S-Shaped Model	389.10	0.0935493	8.69	333.53
Exponential Model	455.371	0.0267368	27.09	206.93

AE = Accuracy of Estimation
MSF = Mean of Square fitting Faults
TEF = Testing-Effort Function

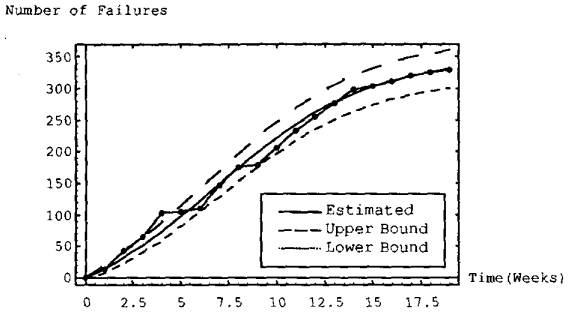

Figure 1(a): Mean value function $m(t)$ of Eq. (6) and the 90% confidence bounds vs. time.

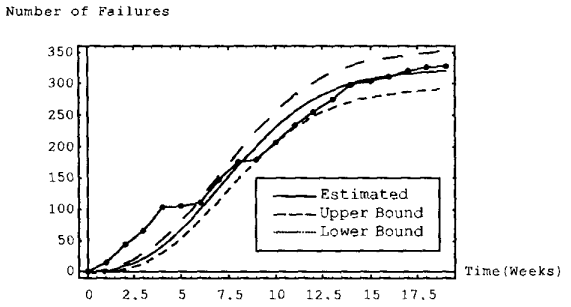

Figure 1(b): Mean value function $m(t)$ of Eq. (11) and the 90% confidence bounds vs. time.

4. Investigation of imperfect debugging

4.1 Modeling SRGM with logistic testing-effort function under imperfect debugging environment

From our studies [1-3], different SRGMs make different assumptions and therefore can be applied to different situations. Most SRGMs published in the literature assume that each time a failure occurs, the error that caused it is immediately removed and no new errors are introduced. Besides, some people also assume that the correction of an error takes only negligible time and an error detected is removed with certainty [18-20]. These assumptions help to reduce the complexity of modeling software reliability growth [21-24]. In this section, we plan to incorporate a relaxation of the above assumption in order to make the SRGMs more realistic and practical. Specially, the software reliability growth model with a logistic testing-effort function under the imperfect debugging environment can be illustrated. To show this, we modify the sixth assumption presented in Section 2:

1. When a fault is detected and removed, it is possible to generate another one;

2. When removing/fixing a detected fault, the probability of introducing another fault is a constant β.

Based on the assumptions (1)–(5) described in Section 2 and the above modified assumptions, we demonstrate the modified software reliability growth model under imperfect debugging environment in detail. Due to the limitation of space, here we only use Eq. (6) as the estimated mean value function for describing the imperfect debugging. Similarly, the Eq. (11) can also be applied based on the same procedure. Rewriting the Eq. (5) as

$$\frac{dm(t)}{dt} = w(t) \times r \times [n(t) - m(t)] \qquad (12)$$

Here we propose a fault content function:

$$n(t) = a + \beta \times m(t) \qquad (13)$$

Solving the above two differential equations, we have

$$m(t) = \frac{a}{1-\beta} \times (1 - e^{-r \times (1-\beta) \times (W(t) - W(0))}) \qquad (14)$$

$$n(t) = \frac{a}{1-\beta} \times (1 - \beta \times e^{-r(1-\beta)(W(t) - W(0))}) \qquad (15)$$

It is noted that the failure intensity function $\lambda(t)$ is given by

$$\lambda(t) = arw(t)exp[-r(1-\beta)W^*(t)] \qquad (16)$$

The expected number of remaining faults after testing time t is

$$m_{\text{remaining}}(t) = n(t) - m(t) = a(exp[-r(1-\beta)W^*(t)]) \qquad (17)$$

It is clear that the above equation $m_{\text{remaining}}(t)$ is a monotonic decreasing function in testing time t. However, the above equations can represent the case where a fault is not successfully removed and new faults are introduced during the testing/debugging phase. Besides, the expected number of detected faults after an infinite amount of test time is

$$m(\infty) = \frac{a}{1-\beta} \times (1 - exp[-r(1-\beta)\frac{NA}{1+A}]) \qquad (18)$$

4.2 Fitting imperfect debugging model to real software data set

Using the proposed imperfect-debugging model, we now show a real numerical illustration for software reliability measurement. Here, in order to validate the imperfect-debugging model, the AE and MSF are selected as the evaluation criteria. Table 2 shows the estimated parameters of Eq. (14) and the comparison among the estimated initial faults a and MSF of different models presented. The observed data, estimated growth curves of the cumulative number of detected faults, and the introduced faults versus time are plotted in Figure 2. It shows that Eq. (14) fits this data set well. According to

Table 2, we found that the proposed imperfect debugging model has smaller AE and MSF. Furthermore, the extended model of Eq. (14) has a better fit to the observed data compared with other proposed imperfect debugging models. Hence, we can conclude that the fault removal process in the software development/testing environment may not be a pure perfect debugging process (the estimated fault introduction rate β=0.0149353 is close to zero but not a zero). However, if we ignore the impact of imperfect debugging, the original model in Eq. (6) still can have a better fit than other models even in the imperfect-debugging environment. The plot of estimated remaining fault content at testing time is shown in Figure 3. We can see that it decreases as time progresses but does not approach to zero. That is, some faults are eventually undetected/removed in the testing phase. Moreover, we note that β can be pre-calculated from experiences or previous projects releases and the possible values of β are listed in Table 2. From the simulation results, we know that the larger the fault introduction rate, the larger the fault detection rate and the smaller the number of initial faults. The simulation results reflect that the introduction of new faults during the correction process tends to be a minor effect in many development efforts if we apply our proposed model. The reason is that the derivations of logistic testing-effort function in its basic concepts already incorporate the human factors and uncertainties into consideration [14]. Therefore, we can conclude that the proposed model in Eq. (6) has the built-in flexibility and has been tested on real software failure data to show its applicability even if the assumption of perfect debugging is eliminated. In fact, in [22] they found that about 14 percent of the errors detected and removed during the observation period were introduced as a result of imperfect debugging. They also supported the conjecture that the exponential SRGMs (particularly the G-O model) still can be used in practice even when the assumption of perfect debugging does not hold [22, 25]. Actually, in any case, the extended model to incorporate imperfect debugging in Eq. (14) is isomorphic to Eq. (6).

Table 2: Comparison results under the imperfect-debugging environment

Model	a	r	β	MSF	AE
Original model	394.076	0.0427223	0.00	118.29	10.06
Eq. (14)	389.66	0.0422699	0.0149353	114.08	8.66
$m_c(t)$ [23]	352.00	0.0841	0.062	153.0	11.68
K-G [23]	455.00	0.0309	0.140	207.0	27.09
Eq. (14)	387.65	0.0424883	0.02	114.08	8.28
Eq. (14)	383.703	0.0429263	0.03	114.08	7.18
Eq. (14)	379.747	0.0433735	0.04	115.0	6.07
Eq. (14)	375.7981	0.0439301	0.05	119.0	4.97
Eq. (14)	371.036	0.0442963	0.06	114.08	3.64
Eq. (14)	367.88	0.0447726	0.07	114.08	2.76
Eq. (14)	363.24	0.0452593	0.08	1140.08	1.46
Eq. (14)	359.968	0.0457567	0.09	114.08	0.55
Eq. (14)	356.013	0.0462651	0.10	114.08	0.56

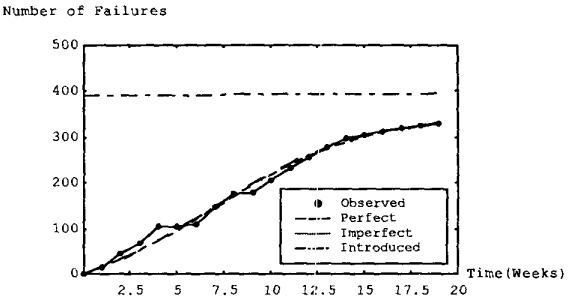

Figure 2: The cumulative number of observed/ estimated failures vs. time.

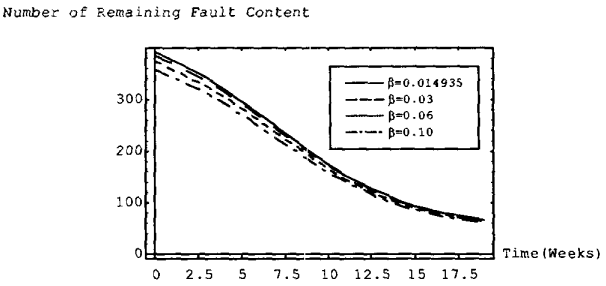

Figure 3: Expected number of remaining fault content.

5. Conclusions

In this paper, we have proposed a SRGM incorporating the logistic testing-effort function that is completely different from the traditional Weibull-type curve. We show that the software reliability growth analysis based on the effort index is more accurate than that based on the calendar time data through several experiments on real data. The reason is that most software reliability growth curves are similar and depend on the distributions of the testing-effort expenditures in the real world. In addition to incorporating the logistic testing-effort function into the exponential-shaped software reliability model, we also integrate this testing-effort function into the Yamada S-shaped software reliability model. The experimental results indicate that the proposed two models fits the real data set fairly well and gives us an exact/reasonable description of fault detection process. Ohba [6] mentioned that Delayed S-shaped model does not fit the observed data well when the testing effort spent on failure detection/ isolation is not a constant. However, by integrating the testing-effort function into the Delayed S-shaped software

reliability model, we can find that the extended Delayed S-shaped model with logistic testing-effort function still has smaller AE and MSF than with Rayleigh testing-effort function. In addition, we also discuss the extended SRGM where the assumption of perfectly removing faults is not adopted. The simulation results reflect that the introduction of new faults during the correction process tends to be a minor effect in many development efforts since the derivations of logistic testing-effort function in the original basic concepts already incorporate the human factors into consideration. Therefore, the assumption of fault introduction rate being a constant over time in this model should be valid and reasonable.

Acknowledgment

We would like to express our gratitude for the support of the National Science Council, Taiwan, R.O.C., under Grant NSC 87-TPC-E-002-017. The work described in this paper was partially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region (Project No. CUHK4432/99E), and by a grant from France/Hong Kong Joint Research Scheme 1999/2000. Besides, the authors also thank several anonymous referees for their constructive reviews and comments.

References

[1] M. R. Lyu (1996). *Handbook of Software Reliability Engineering*. McGraw Hill.

[2] J. D. Musa, A. Iannino, and K. Okumoto (1987). *Software Reliability, Measurement, Prediction and Application*. McGraw Hill.

[3] J. D. Musa (1999). *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*. McGraw-Hill.

[4] M. R. Lyu, "Reliability-Oriented Software Engineering: Design, Testing, and Evaluation Techniques," *IEE Software–Proceedings*, vol. 145, no. 6, pp. 191-197, 1998

[5] A. L. Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Trans. on Software Engineering*, Vol. SE-11, No. 12, Dec. 1985.

[6] M. Ohba, "Software Reliability Analysis Models, " *IBM J. Res. Develop.*, Vol. 28, No. 4, pp. 428-443, July 1984.

[7] P. N. Misra, "Software Reliability Analysis, " *IBM Systems Journal*, Vol. 22, No. 3, pp. 262-279, 1983.

[8] W. Everett, S. Keene, and A. Nikora, "Applying Software Reliability Engineering in the 1990s, " *IEEE Transactions on Reliability*, Vol. R-47, pp. 372-378, Sept. 1998.

[9] A. Gana and S. T. Huang, "Statistical Modeling Applied to Managing Global 5ESS-2000 Switch Software Development," *Bell Labs Technical Journal*, Vol. 2, No.1, pp. 144-153, winter 1997.

[10] S. Yamada, H. Ohtera, and H. Narihisa, "Software Reliability Growth Models with Testing Effort, " *IEEE Trans. on Reliability*, vol. R-35, No. 1, pp. 19-23, 1986.

[11] S. Yamada, J. Hishitani, and S. Osaki, "Software Reliability Growth Model with Weibull Testing Effort: A Model and Application, " *IEEE Trans. on Reliability*, Vol. R-42, pp. 100-105, 1993.

[12] S. Yamada, J. Hishitani, and S. Osaki, " A Software Reliability Growth Model for Test-Effort Management," *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, pp. 585-590, Sept. 11-13, 1991, Tokyo, Japan.

[13] J. Hishitani, S. Yamada, and S. Osaki, "Comparison of Two Estimation Methods of the Mean Time-Interval Between Software Failures, " *Proceedings of the Ninth Annual International Phoenix Conference on Computers and Communications Conference*, pp.418-424, March 21-23, 1990, Scottsdale, Arizona, USA.

[14] C. Y. Huang, S. Y. Kuo and I. Y. Chen, "Analysis of a Software Reliability Growth Model with logistic Testing-Effort Function, " *Proceedings of the 8th International Symposium on Software Reliability Engineering*, pp. 378-388, 1997, Albuquerque, New Mexico. U.S.A.

[15] C. Y. Huang, J. H. Lo and S. Y. Kuo, "A Pragmatic Study of Parametric Decomposition Models for Estimating Software Reliability Growth," *Proceedings of the 9th International Symposium on Software Reliability Engineering*, pp. 111-123, 1998, Paderborn, Germany.

[16] C. Y. Huang, J. H. Lo, S. Y. Kuo, and M. R. Lyu, "Software Reliability Modeling and Cost Estimation Incorporating Testing-Effort and Efficiency," *Proceedings of the 10th International Symposium on Software Reliability Engineering*, pp. 62-72, 1999, Boca Raton, FL, U.S.A.

[17] T. DeMarco (1982). *Controlling Software Projects:* Management, Measurement and Estimation. Prentice-Hall, Englewood Cliffs, NJ.

[18] S. Chatterjee, R. B. Misra and S. S. Alam, "Joint Effect of Test Effort and Learning Factor on Software Reliability and Optimal Release Policy," *International Journal of Systems Science*, Vol. 28, No. 4, pp. 391-396, 1997.

[19] G. Xia, P. Zeephongsekul, and S. Kumar, "Optimal Software Release Policy with a Learning Factor for Imperfect Debugging," *Microelectronics and Reliability*, Vol. 33, pp. 81-86, 1993.

[20] H. Pham, L. Nordmann, and X. Zhang, "General Imperfect-Software-Debugging Model with S-Shaped Fault-Detection Rate, " *IEEE Transactions on Reliability*, Vol. R-48, No. 2, pp. 169-175, June 1999.

[21] S. Yamada, K. Tokuno, and S. Osaki, "Imperfect Debugging Models with Fault Introduction Rate for Software Reliability Assessment," International Journal of Systems Science, Vol. 23, No. 12, pp. 2241-2252, 1992.

[22] M. Ohba and X. Chou, "Does Imperfect Debugging Affect Software Reliability Growth?," Proc. 11th International Conference on Software Engineering, pp. 237-244, 1989.

[23] P. K. Kapur and R. B. Garg, "Modeling an Imperfect Debugging Phenomenon in Software Reliability," *Microelectronics and Reliability*, Vol. 36, pp. 645-650, 1996.

[24] R. H. Hou, S. Y. Kuo and Y. P. Chang, "Hyper-Geometric Distribution Software Reliability Growth Model with Imperfect Debugging," *Proceedings of the 1995 International Symposium on Software Reliability Engineering*, pp. 195-200, Nov. 1995, Tolouse, France.

[25] M. R. Lyu and A. Nikora, " Using Software Reliability Models More Effectively," *IEEE Software*, pp. 43-52, July 1992.