Software Reliability Modeling and Cost Estimation Incorporating Testing-Effort and Efficiency

Chin-Yu Huang[1], Jung-Hua Lo[1], Sy-Yen Kuo[1], and Michael R. Lyu[2].

[1]Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
sykuo@cc.ee.ntu.edu.tw

[2]Computer Science & Engineering Department
The Chinese University of Hong Kong
Shatin, Hong Kong
lyu@cse.cuhk.edu.hk

Abstract

Many studies have been performed on the subject of software reliability, but few explicitly consider the impact of software testing on the reliability process. This paper presents two important issues on software reliability modeling and software reliability economics: testing effort and efficiency. First, we will discuss on how to extend the logistic testing-effort function into a general form. The generalized logistic testing-effort function has the advantage of relating the work profile more directly to the natural flow of software development. Therefore, it can be used to describe the actual consumption of resources during software development process and get a conspicuous improvement in modeling testing-effort expenditures. Furthermore, we will incorporate the generalized logistic testing-effort function into software reliability modeling and its fault-prediction capability is evaluated through four numerical experiments on real data. Then, we will address the effects of automated techniques or tools on increasing the efficiency of software testing. New testing techniques will usually increase test coverage. We propose a modified software reliability cost model to reflect these effects. From the simulation results, we obtain a powerful software economic policy which clearly indicates the benefits of applying new automated testing techniques and tools during software development process.

1. Introduction

When computer applications permeate our daily life, reliability becomes a very important characteristic of a computer system. In modern society, computer-controlled and computer-embedded systems heavily depend on the correct performance of software. Software reliability is one of the most important features for a critical system which can affect human's life. Therefore, it is necessary to measure and control the reliability of a software system. A number of *Software Reliability Growth Models* (SRGMs) have been proposed [12, 26]. Among these models, Goel and Okumoto considered an NHPP as the stochastic process to describe the fault process [11]. Yamada et al. [1-3] modified the G-O model and incorporated the concept of testing-effort in an NHPP model to get a better description of the software fault phenomenon. Later, we [7-8] also proposed a new software reliability growth model with the logistic testing-effort function. In this paper, we extend the logistic testing-effort function to a generalized form. The generalized logistic testing-effort function has the advantage of relating a work profile more directly to the natural structure of the software development. Therefore, it can be used to pertinently describe the resource consumption during the software development process and get a conspicuous improvement in modeling the distribution of testing-effort expenditures.

In general, we will have more confidence in the measured software reliability with more software tests. Unfortunately, testing with ineffective or redundant test cases may lead to excessive cost. To avoid such phenomenon, we need to know when to stop testing. One alternative is to restrict the test data such that testing will stop when the odds of detecting additional faults (estimated by SRGMs) are very low. But this may not be realistic since testers typically want to test for all possible valuable failure data, even the cost of testing is significant. Okumoto and Goel [11] first discussed the software optimal release policy from the cost-benefit viewpoint and proposed a software reliability cost model. It was shown that the optimal software release time can be obtained based on a cost criterion when minimizing the total expected cost. Recently, many papers discussed such optimal software release time problem based on the cost-reliability relationship [4-6, 8-11, 113, 18-19, 21, 24]. In

Sy-Yen Kuo is with AT&T Labs-Research, New Jersey. He is on leave from National Taiwan University, Taipei, Taiwan.

fact, to detect additional faults during the test phase of a software development process, the testers or debuggers may use some new automated tools or methods that are just discovered and become available. These tools, techniques or methods can greatly help the developers and testers to create tests and eliminate some redundant test cases. As time progresses, they can detect additional faults during testing, which saves the greater expense of correcting faults during the operational phase. These approaches have improved software testing and productivity recently, allowing project managers to maximize software reliability. Hence the extra cost trade-off based on new techniques and tools can be considered in software reliability cost model and viewed as the investment required to improve long-term competitiveness and to speed up the software product release in the commercial market. In this paper, we propose a new reliability cost model that provides a means of assessing whether the software cost is under control and the software quality is improving with time. The methods we propose allow the software testers and software quality assurance (SQA) engineers to decide when the software is likely to be of adequate quality for release.

2. Relationship between SRGM and testing-effort function

In this section we propose a set of generalized software reliability growth models incorporating testing-effort functions. The mathematical relationship between reliability models and testing effort expenditures is explicitly described in detail. Numerical results are given to illustrate the advantage of this new approach.

2.1 Software reliability modeling descriptions

2.1.1 Review of SRGM with Logistic testing-effort function

A typical software reliability model is based on the following assumptions [12]:

1. The fault removal process is modeled by a *Non Homogeneous Poisson Process* (NHPP).
2. The software system is subject to failures at random times caused by manifestation of remaining faults in the system.
3. The mean number of faults detected in the time interval $(t, t+\Delta t]$ to the current testing-effort is proportional to the mean number of remaining faults in the system at time t.
4. The proportionality is a constant over time.
5. Testing effort expenditures are described 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.

Based on the third assumption, we obtain the following differential equation:

$$\frac{dm(t)}{dt}\times\frac{1}{w(t)}=r\times[a-m(t)] \tag{1}$$

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

$$m(t)=a(1-\exp[-r(W(t)-W(0))])=a(1-\exp[-rW^{*}(t)]) \tag{2}$$

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

Eq. (2) is an NHPP model with mean value function considering the testing-effort consumption. From the above description, we know that $w(t)$ represents the current testing-effort consumption (such as volume of test cases, human power, CPU time, and so on) at time t during the software testing/debugging phase. The consumed testing-effort can indicate how effective the faults are detected in the software. Therefore, this function plays an important role in modeling software reliability and it can be described by different distributions. From the studies in [1-5, 14], several testing-effort pattern expressions, such as *Exponential, Rayleigh,* and *Weibull-type* curves, can be applied. Moreover, we [7-8] proposed a *Logistic* testing-effort function to describe the possible test effort patterns, in which the current testing-effort consumption is

$$w(t)=\frac{NA\alpha\times\exp[-\alpha t]}{\left(1+A\exp[-\alpha t]\right)^{2}}=\frac{NA\alpha}{(\exp[\frac{\alpha t}{2}]+A\exp[-\frac{\alpha t}{2}])^{2}} \tag{3}$$

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.

The cumulative testing effort consumption of *Logistic* testing-effort function in time $(0, t]$ is

$$W(t)=\frac{N}{1+A\exp[-\alpha t]} \tag{4}$$

and

$$W(t)=\int_0^t w(\delta)d\delta \tag{5}$$

Besides, the testing effort $w(t)$ reaches its maximum value at time

$$t_{\max}=\frac{\ln A}{\alpha} \tag{6}$$

2.1.2 A generalized Logistic testing-effort function

From the previous studies in [7-8], we know that the

Logistic testing-effort function (i.e. the Parr model [14]) is based on a description of the actual software development process and can be used to describe the work profile of software development. In addition, this function can be used to consider and evaluate the effects of possible improvements on software development methodology, such as top-down design or stepwise refinement. Therefore, if we relax some assumptions when deriving the original Parr model and take into account the structured development effort, we obtain a generalized *Logistic* testing-effort function as:

$$W_{\kappa}(t) = N \times \left(\frac{(\kappa+1)/\beta}{1 + Ae^{-\alpha\kappa t}} \right)^{1/\kappa} \quad (7)$$

where κ is a structuring index with a large value for modeling well-structured software development efforts, and β is a constant.

If κ=1, the above equation becomes

$$W_{\kappa}(t) = \frac{N}{1 + Ae^{-\alpha t}} \times \frac{2}{\beta} \quad (8)$$

If β is viewed as a normalized constant and β =2, the above equation is reduced to Eq. (4).

Similarly, if κ=2, we have

$$W_{\kappa}(t) = \frac{N}{\sqrt{1 + Ae^{-2\alpha t}}} \sqrt{\frac{3}{\beta}} \quad (9)$$

Similarly, if we set β=κ+1, we get a more generalized and plain solution for describing the cumulative testing effort consumption in time (0, *t*]:

$$W_{\kappa}(t) = \frac{N}{\sqrt[\kappa]{1 + Ae^{-\alpha\kappa t}}} \quad (10)$$

In this case, the testing effort *w*(*t*) reaches its maximum value at time

$$t^{\kappa}_{\max} = \ln(\frac{A}{\kappa}) / \alpha\kappa \quad (11)$$

2.2 Numerical examples

2.2.1 Numerical example 1

The first data set is from Ohba [17] where the testing time is measured in CPU hours. The *Maximum Likelihood Estimation* and *Least Squares Estimation* are used to estimate the parameters of Eq. (2), Eq. (4), and Eq. (10), and we substitute the calculated normalizing value for β. The estimated values of parameters for the generalized logistic testing-effort function are listed in Table 1. From Table 1, κ=2.63326 is the real estimated value for the first data set and the other possible values of κ are pre-calculated. Figure 1 depicts the fitting of the estimated current testing effort by using generalized logistic testing-effort function, in which we find that the peak work rate occurs when about half of the work on the project has been done. This phenomenon can be interpreted as that in a well-structured software development environment, the slope of the testing-effort consumption curve may grow slowly initially, but a compensating reduction will happen later. Table 2 shows the estimated values of parameters by using different SRGMs and two comparison criteria, *Accuracy of Estimation* (AE) and *Mean of Square Fitting Faults* (MSF) [7-8]. The smaller MSF and AE indicate fewer number of fitting faults and better performance. From Table 2, we know that when the value of κ varies from 1 to 3, both *MSF* and *AE* will be less than other existing SRGMs; therefore, it is conceivable that the proposed model has a better goodness-of-fit.

Table 1: Parameters of generalized logistic testing-effort function for the first data set.

N	*A*	α	κ
54.8364	13.0334	0.226337	1
52.0072	40.6042	0.188809	1.5
50.2178	115.228	0.170001	2
49.00	126.00	0.158763	2.5
48.7768	429.673	0.158042	2.63326
48.1368	833.105	0.151344	3
48.1693	2188.22	0.144234	3.5
47.8507	5709.29	0.139933	4
47.6561	14839.3	0.136507	4.5

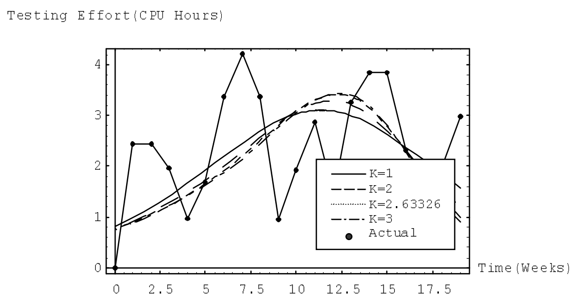

Figure 1: Observed/estimated testing-effort vs. time for the first data set.

Table 2: Comparison results for the first data set.

Model	*a*	*r*	AE(%)	MSF
Proposed Model (κ=1)	394.076	0.042722	10.06	118.29
(κ=1.5)	384.707	0.045037	7.46	114.32
(κ=2)	377.157	0.047815	5.35	112.41
(κ=2.6332)	369.029	0.050955	3.08	110.73

(κ=3)	367.829	0.051905	2.75	105.91
(κ=3.5)	412.871	0.039938	15.32	820.76
(κ=4)	414.426	0.039861	15.76	889.21
(κ=4.5)	416.114	0.039732	16.23	952.38
G-O Model	760.00	0.032268	112.29	139.82
G-O with Weibull fun	565.35	0.019659	57.91	122.09
G-O with Rayl. Fun.	459.08	0.027336	28.23	268.42
G-O with Exp. fun.	828.252	0.011783	131.35	140.66
Inflection S Model	389.1	0.093549	8.69	133.53
Delayed S Model	374.05	0.197651	4.48	168.67
Exp. Model	455.371	0.026736	27.09	206.93
Delayed S Model with Ray. fun.	333.18	0.100415	6.93	798.49
S-Shaped Model with logistic fun.	338.136	0.10004	5.54	242.79
HGDM	387.71	NA	8.3	138.12
HGDM with linear factor	387.709	NA	8.30	138.11
HGDM with Exp. factor	385.132	NA	7.56	111.24
Musa Log. Poisson	NA	*	*	171.23

2.2.2 Numerical example 2

The second data set is cited from Musa et al. [4-5]. The software were tested for 21 weeks (25.3 CPU Hours were used) and 136 faults were detected. The *Maximum Likelihood Estimation* and *Least Squares Estimation* are used to estimate the parameters of the Eq. (2), Eq. (4), and Eq. (10) and we substitute the calculated normalizing value for β. The estimated values for the parameters are listed in Table 3. In fact, from Table 3, κ =1.27171 is the real estimated value for the second data set and other possible values of κ are pre-calculated. Figure 2 depicts the fitting of the estimated current testing effort by using generalized logistic testing-effort function. Table 4 shows the estimated values of parameters and the comparison results between the observed and the estimated values obtained by the other SRGMs. Similarly, smaller AE and MSF indicate less fitting errors and better performance. We find that when the value of κ varies from 1.5 to 4.5, both *MSF* and *AE* will be less than other existing SRGMs. Hence, we still can conclude that the proposed model is good enough to give a more accurate description of resource consumption during the software development phase and gives a better fit in this experiment.

Table 3: Parameters of generalized logistic testing-effort function for the second data set.

N	*A*	α	κ
29.1095	4624.89	0.493515	1
28.153	20903.9	0.44470	1.27171
28.1513	45843.8	0.39737	1.5
28.1458	260550	0.33307	2
28.0464	3784150	0.257234	3
27.5626	5329270	0.221165	3.5
27.0202	7428092	0.195207	4
26.0532	22955900	0.186095	4.5

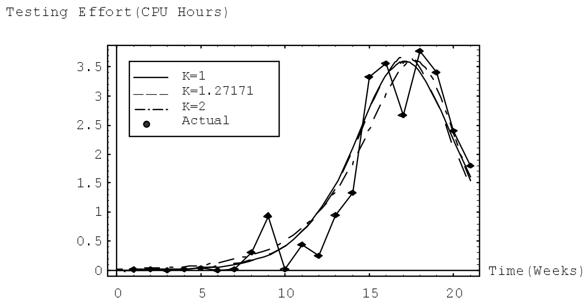

Figure 2: Observed/estimated testing-effort vs. time for the second data set.

Table 4: Comparison results for the second data set.

Model	***a***	***r***	**AE (%)**	**MSF**
Proposed Model (κ=1)	138.026	0.145098	26.58	62.41
(κ =1.27171)	140.013	0.137916	25.52	53.79
(κ =1.5)	139.191	0.141159	25.96	35.04
(κ =2)	142.505	0.12406	24.19	17.62
(κ =3)	147.808	0.103272	21.37	9.17
(κ =3.5)	154.144	0.089175	18.01	18.58
(κ =4)	162.235	0.077407	13.70	32.38
(κ =4.5)	168.265	0.072116	10.49	36.58
G-O Model	142.32	0.1246	24.29	2438.3
G-O with Rayleigh fun.	866.94	0.009624	361.13	89.24
Exp. Model	137.2	0.156	27.12	3019.66
Delay S-shaped Model	237.196	0.096344	26.16	245.25
Delayed S with Exp. fun.	688.593	0.019762	266.27	235.19
Delayed S with Logistic function.	137.49	0.330611	26.86	207.37

2.2.3 Numerical example 3

The third set of real data is the pattern of discovery of faults in the software that supported Space Shuttle flights STS2, STS3, STS4 at the Johnson Space Center [22]. The system is also a real-time command and control application. A weekly summary of software test hours and the faults of various severity discovered is given in [22]. The cumulative number of discovered faults up to thirty-eight weeks is 227. Similarly, the *Maximum Likelihood Estimation* and *Least Squares Estimation* are used to estimate the parameters of the Eq. (2), Eq. (4), and Eq. (10), and we substitute correct normalizing value for β. The estimated values of parameters for the generalized logistic testing-effort function are listed in Table 5. In fact, from Table 5, κ=1.25262 is real estimated value for this data set and the other possible values of κ are pre-calculated. Figure 3 depicts the fitting of the estimated current testing effort by using generalized logistic testing-effort function. Table 6 shows the estimated values of parameters by using different SRGMs and the comparison criteria. Therefore, the estimation results of individual models show that the proposed model gives the better AE.

Table 5: Parameters of generalized logistic testing-effort function for the third data set.

N	*A*	α	κ
2828.88	10.5057	0.0988842	1
2626.32	18.3734	0.093100622	1.25262
2664.54	30.3765	0.082482	1.5
2570.80	80.3661	0.074041	2
2507.67	203.404	0.0689604	2.5
2463.20	503.125	0.0655753	3
2460.83	1229.1	0.063166	3.5

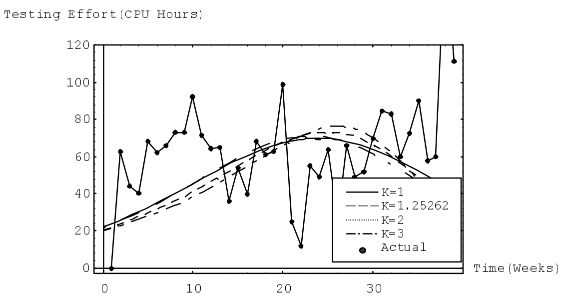

Figure 3: Observed/estimated testing-effort vs. time for the third data set.

Table 6: Comparison results for the third data set.

Model	***a***	***r***	**AE (%)**
Proposed Model (κ=1)	241.325	0.000907	75.5
(κ=1.25262)	240.842	0.000908	63.4065
(κ=1.5)	234.686	0.000983	71.9053
(κ=2)	229.605	0.001078	71.9443
(κ=2.5)	227.027	0.001108	73.6626
G-O Model	597.887	0.000209	78.87
G-O with Rayleigh Function	245.017	0.0007158	183.366

2.2.4 Numerical example 4

The fourth set of real data is the pattern of discovery of faults by Thoma in [23]. The debugging time and the number of detected faults per day are reported. The cumulative number of discovered faults up to twenty-two days is 86 and the total consumed debugging time is 93 CPU hours. All debugging data are used in this experiment. Similarly, we can estimate each parameter by the *Maximum Likelihood Estimation* and *Least Squares Estimation* in the proposed SRGM and they are shown in Table 7. In fact, from Table 7, κ=1.76033 is real estimated value for this data set and the other possible values of κ are pre-calculated. Figure 4 depicts the fitting of the estimated current testing effort by using generalized logistic testing-effort function. Table 8 shows the estimated values of parameters by using different SRGMs and the comparison criteria. Therefore, in this data set, we conclude that our proposed model gets a reasonable prediction in estimating the number of software faults and fits this data set better than others.

Table 7: Parameters of generalized logistic testing-effort function for the fourth data set.

N	*A*	α	κ
99.9028	28.0091	0.257426	1
95.6453	109.068	0.2148526	1.5
95.00	231.00	0.2055126	1.76033
94.90	389.026	0.1936605	2
93.20	1336.10	0.1811212	2.5

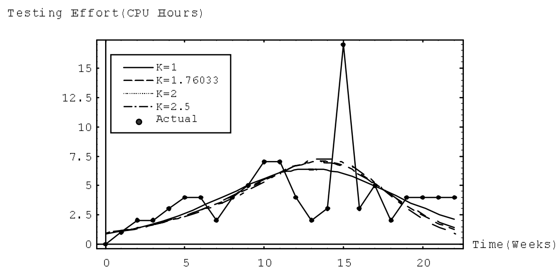

Figure 4: Observed/estimated testing-effort vs. time for the fourth data set.

Table 8: Comparison results for the fourth data set.

Model	*a*	*r*	AE (%)
Proposed Model (κ=1)	88.8931	0.0390591	55.015
(κ=1.5)	88.699	0.0385438	22.2717
(κ=1.76033)	90.354	0.0371217	29.4215
(κ=2)	90.4078	0.0373532	28.4941
(κ=2.5)	90.6226	0.0373478	28.2564
G-O Model	137.072	0.0515445	25.33
HGDM	88.3	*	33.6812

3. Optimal release time incorporating test efficiency

In the last section we describe a generalized approach to incorporate testing effort into software reliability models. In this section we will identify the efficiency of testing and study its impact on software reliability. In particular, we discuss how to incorporate testing efficiency into reliability models and how to determine the optimal software release time.

3.1 Impact of new tools/techniques on software testing efficiency

As soon as software coding is completed, the necessary but expensive testing phase starts. During the testing phase, the developers will need to make a software reliability evaluation and determine when to stop testing. If the results meet the requirement specifications and the reliability criteria are also satisfied, then the software product is ready for release. Therefore, adjusting specific parameters in an SRGM and adopting the corresponding actions appropriately can help to achieve the goal of determining the software release time. Several approaches can be applied. For example, we have discussed the applications of testing-effort control and management problem in our previous study [7]. Using the proposed methods, we can easily control the modified consumption rate of testing-effort expenditures and detect more faults in a specified time interval. This means that the developers and testers can devote their time and resource to complete their testing tasks based on well-controlled expenditures.

In addition to controlling the testing-effort expenditures, we can achieve a given operational quality at a specified time by introducing new automated testing tools and techniques. That is, through the adoption of new testing techniques and tools, we can detect and remove more additional faults (i.e. those faults that are not easily exposed during the testing phase). These new methods, however, will impose extra software development cost. For example, professional experts can help developers to assess the original software development process, to meet their quality goals, and to reduce risks. In general, these external personnel can offer efficient and effective approaches to test planning, module-level unit testing, or testing strategies. Moreover, many automated testing tools and techniques are available to increase test coverage and replace traditional manual testing. The benefits of applying new techniques and tools include increased software quality, reduced testing costs, improved release time to market, repeatable test steps, and improved testing productivity [5, 12, 15, 20]. Consequently, it is desirable that these experts and automated testing tools/techniques can greatly help the developers in detecting additional faults that are difficult to find during regular testing and usage, in identifying and correcting faults effectively, and in improving their software development processes.

An important step toward these new approaches, then, is to offer enough information about these approaches to software developers and reliability engineers. Before adopting the automated techniques and tools, we should get quantitative information from the industrial data relative to these methods' past performance (i.e. the previous testing experience), or get qualitative information from the evaluation on the methods' attributes. Basically, these methods' past performance should be evaluated in determining whether they will be successful in managing reliability growth [20]. In addition, they can be evaluated by performing various simulations based on actual data sets. Finally, the test team's capacity in applying these techniques and tools and the related operational profiles also play an important role. We discuss how the software reliability modeling process can include these testing methods, and how a new optimal software release time problem can be formulated and solved.

3.2 Optimal software release time problem

Okumoto and Goel [11] first discussed the software optimal release policy from the cost-benefit viewpoint. The total cost of testing-effort expenditures at time T, $C1(T)$, can be expressed as [1-3, 7, 9-11, 13, 18-19, 24]:

$$C1(T) = C_1 m(T) + C_2 [m(T_{LC}) - m(T)] + C_3 \times \int_0^T w(x)dx \tag{12}$$

where T_{LC}=software life-cycle length
C_1=cost of correcting an error during testing
C_2=cost of correcting an error during operation
C_3=cost of testing per unit testing-effort expenditures.

From the work by B. Boehm [16], we know $C_2 > C_1$ as C_2 is usually an order of magnitude greater than C_1. In order to detect additional faults during testing, the testers and debuggers may use new automated tools or techniques. The cost trade-off of these new tools and techniques, therefore, should be considered in the software cost model, including their expenditures and benefits. Consequently, we modify the overall software cost model as follows [24]:

$$C2(T) = C_0(T) + C_1 \times (1+P) \times m(T) + C_2[m(T_{LC}) - (1+P) \times m(T)] + C_3 \times \int_0^T w(x)dx \quad (13)$$

where $C_0(T)$ is the cost function for developing and acquiring the automated tools and techniques that detect an additional fraction P of faults during testing.

We note that the cost for developing and acquiring new tools or techniques, $C_0(T)$, does not have to be a constant during the testing. Moreover, the testing cost for $C_0(T)$ can be parameterized and estimated based on actual data. From our experience, we found that $C_0(T)$ may have different forms as time progresses, which depends on the characteristics of a tool's performance, testing effort expenditures, effectiveness, and so on. We can formulate this cost function as simple linear functions or simple non-linear functions. In general, the longer the software is tested, the more the testing cost $C_0(T)$. Under the cost-benefit considerations, the automated tools or techniques will pay for themselves if

$$C1(T) - C2(T) \geq 0 \quad (14)$$

That is, $C_1 m(T) + C_2[m(T_{LC}) - m(T)] + C_3 \times \int_0^T w(x)dx - C_0(T) - C_1(1+P)m(T) - C_2[m(T_{LC}) - (1+P)m(T)] - C_3 \times \int_0^T w(x)dx \geq 0$

Rearranging the above equation, we obtain

$$C_0(T) \leq P \times m(T) \times (C_2 - C_1) \quad (15)$$

Eq. (15) is used to decide whether the new automated tools or techniques are effective or not. If $C_0(T)$ is low enough or if the new methods are effective in detecting additional faults, this investment is worthwhile. Usually appropriate automated tools or techniques are best selected depending on how thoroughly failure data are collected and faults are categorized [15]. Sometimes incorporating new automated tools and techniques into a software development process may introduce excessive, that is, $C1(T) - C2(T) < 0$. This phenomenon usually occurs infrequently, but if it can really shorten the testing period under the same software reliability requirements, we may still consider applying the new techniques. By differentiating Eq. (13) with respect to the time T we have:

$$\frac{d}{dT}C2(T) = \frac{d}{dT}C_0(T) + C_1\frac{d}{dT}((1+P)m(T)) - C_2 \times \frac{d}{dT}((1+P)m(T)) + C_3 \times w(T) \quad (16)$$

If we let Eq. (16) be equal to zero and use the mean value function in Eq. (2), we can get a finite and unique solution T_0 for the determination of an optimal software release time problem based on the new cost criterion.

From Eq. (16), if we let $C_1(1+P) = C_1^*$ and $C_2(1+P) = C_2^*$, then we have

$$\frac{d}{dT}C2(T) = \frac{d}{dT}C_0(T) + C_1^*\frac{d}{dT}m(T) - C_2^*\frac{d}{dT}m(T) + C_3 \times w(T) \quad (17)$$

If the mean value function is given in Eq. (2), we obtain

$$\frac{d}{dT}C2(T) = \frac{d}{dT}C_0(T) + C_1^* arw(T)\exp[-rW^*(T)] - C_2^* arw(T) \times \exp[-rW^*(T)] + C_3 \times w(T) \quad (18)$$

Without loss of generally, we consider several possibilities for $C_0(T)$ in order to interpret the cost consumption:

(1) $C_0(T)$ is a constant.
(2) $C_0(T)$ is proportional to the testing-effort expenditures.
(3) $C_0(T)$ is exponentially related to the testing-effort expenditures.

A. $C_0(T) = C_0$, $T \geq T_s$; $C_0(T) = 0$, $T < T_s$

$$\frac{d}{dT}C2(T) = w(T) \times [-(C_2^* - C_1^*)ar\exp[-r((W(T) - W(0))] + C_3] \quad (19)$$

Since $w(t) > 0$ for $0 < T < \infty$, $\frac{d}{dT}C2(T) = 0$ if

$$(C_2^* - C_1^*)ar\exp[-r(W(T) - W(0))] = C_3 \quad (20)$$

The left-hand side in Eq. (20) is a monotonically decreasing function of T. Here we let T_s be the starting time of adopting new techniques/tools. If $(C_2^* - C_1^*)ar\exp[-r(W(T_s) - W(0))] \leq C_3$, then $(C_2^* - C_1^*)ar\exp[-r(W(T_{LC}) - W(0))] < C_3$ for $T_s < T < T_{LC}$. Therefore, the optimal software release time $T^* = T_s$ since $\frac{d}{dT}C2(T) > 0$ for $T_s < T < T_{LC}$. Similarly, if $(C_2^* - C_1^*)ar\exp[-r(W(T_s) - W(0))] > C_3$ and $(C_2^* - C_1^*)ar\exp[-r(W(T_{LC}) - W(0))] < C_3$, there exists a finite and unique solution T_0 satisfying Eq. (20). That is,

$$T_0 = \frac{1}{\alpha} \times \ln\left(\frac{A\Theta^\kappa}{N^\kappa - \Theta^\kappa}\right) \text{ minimizes } C2(T) \quad (21)$$

where $\Theta = \frac{1}{r}\left(\ln\left(ar\frac{C_2^* - C_1^*}{C_3}\right)\right) + \frac{N}{\sqrt[\kappa]{1+A}}$

since $\frac{d}{dT}C2(T) < 0$ for $T_s < T < T_0$ and $\frac{d}{dT}C2(T) > 0$ for

$T_0 < T < T_{LC}$.

If $(C_2^* - C_1^*) ar \exp[-r(W(T_{LC}) - W(0))] \geq C_3$, then

$(C_2^* - C_1^*) ar \exp[-r(W(T) - W(0))] > C_3$ for $T_s < T < T_{LC}$. Therefore, the optimal software release time $T^* = T_{LC}$ since $\frac{d}{dT} C2(T) < 0$ for $T_s < T < T_{LC}$.

Theorem 1:
Assume $C_0(T) = C_0$ (constant), $C_0>0$, $C_1>0$, $C_2>0$, $C_3>0$, $C_2>C_1$, we have

CASE $(C_2^* - C_1^*) ar \exp[-r(W(T_s) - W(0))] > C_3$ and $(C_2^* - C_1^*) ar \exp[-r(W(T_{LC}) - W(0))] < C_3$: there exists a finite and unique solution T_0 satisfying Eq. (20) and the optimal software release time is $T^* = T_0$.

CASE $(C_2^* - C_1^*) ar \exp[-r(W(T_s) - W(0))] < C_3$: $T^* = T_s$.

CASE $(C_2^* - C_1^*) ar \exp[-r(W(T_{LC}) - W(0))] > C_3$: $T^* = T_{LC}$.

B. $C_0(T) = C_{01} + C_0 \int_{Ts}^{T} w(t)dt$, $T \geq Ts$; $C_0(T) = 0$, $T < T_s$

where C_{01} is an nonnegative real number that indicates the basic cost of adopting new techniques/tools, and T_s is the start time of adopting new techniques/methods.

$$\frac{d}{dT} C2(T) = C_0 w(T) + C_1^* arw(T) \exp[-rW^*(T)] - C_2^* arw(T) \exp[-rW^*(T)] + C_3 \times w(T)$$

$$= w(T) \times [(C_1^* - C_2^*) ar \exp[-r((W(T) - W(0))] + C_3 + C_0] \quad (22)$$

Since $w(t)>0$ for $0 \leq T < \infty$, $\frac{d}{dT} C2(T) = 0$ if

$$(C_2^* - C_1^*) ar \exp[-r(W(T) - W(0))] = C_3 + C_0 \quad (23)$$

As the left-hand side in Eq. (23) is a monotonically decreasing function of T, therefore, if

$(C_2^* - C_1^*) ar \exp[-r(W(T_s) - W(0))] > C_3 + C_0$ and

$(C_2^* - C_1^*) ar \exp[-r(W(T_{LC}) - W(0))] < C_3 + C_0$, there exists a finite and unique solution T_0 satisfying Eq. (23).

$$T_0 = \frac{1}{\alpha} \times \ln\left(\frac{A\Theta^\kappa}{N^\kappa - \Theta^\kappa} \right) \text{ minimizes } C2(\mathrm{T}) \quad (24)$$

where $\Theta = \frac{1}{r}\left(\ln\left(ar \frac{C_2^* - C_1^*}{C_3 + C_0} \right) \right) + \frac{N}{\sqrt[\kappa]{1 + A}}$

Theorem 2:
Assume $C_0(T) = C_{01} + C_0 \int_{Ts}^{T} w(t)dt$, C_{01}, $C_0>0$, $C_1>0$, $C_2>0$, $C_3>0$, $C_2>C_1$, we have

CASE $(C_2^* - C_1^*) ar \exp[-r(W(T_s) - W(0))] > C_3 + C_0$ and $(C_2^* - C_1^*) ar \exp[-r(W(T_{LC}) - W(0))] < C_3 + C_0$: there exists a finite and unique solution T_0 satisfying Eq. (23) and the optimal software release time is $T^* = T_0$.

CASE $(C_2^* - C_1^*) ar \exp[-r(W(T_s) - W(0))] < C_3 + C_0$: $T^* = T_s$.

CASE $(C_2^* - C_1^*) ar \exp[-r(W(T_{LC}) - W(0))] > C_3 + C_0$: $T^* = T_{LC}$.

C. $C_0(T) = C_{01} + (C_0 \times \int_{Ts}^{T} w(t)dt)^m$, $T \geq Ts$; $C_0(T) = 0$, $T < T_s$

$$\frac{d}{dT} C2(T) = C_0 mw(T) \times (C_0 \int_{Ts}^{T} w(t)dt)^{m-1} + C_1^* \times arw(T) \exp[-rW^*(T)] - C_2^* arw(T) \times \exp[-rW^*(T)] + C_3 \times w(T)$$

$$= w(t) \times [(C_1^* - C_2^*) ar \exp[-rW^*(t)] + C_3 + C_0 m \times (C_0 \int_{Ts}^{T} w(t)dt)^{m-1}]$$

Because $w(t)>0$ for $0 \leq T < \infty$, $\frac{d}{dT} C2(T) = 0$ if

$$P(T) \equiv [(C_2^* - C_1^*) ar \exp[-rW^*(t)] - C_0 m \times (C_0 \times \int_{Ts}^{T} w(t)dt)^{m-1}] = C_3 \quad (25)$$

The left-hand side in Eq. (25) is a monotonically decreasing function of T. Therefore, if

$(C_2^* - C_1^*) ar \exp[-r(W(T_s) - W(0))] > C_3$ and $P(T_{LC}) < C_3$, it means that there exists a finite and unique solution T_0 satisfying Eq. (25), which can be solved by numerical methods. It is noted that $\frac{d}{dT} C2(T) < 0$ for $0 \leq T_s \leq T < T_0$ and $\frac{d}{dT} C2(T) > 0$ for $T > T_0$. Thus, $T = T_0$ minimizes $C2(\mathrm{T})$ for $T_0 < T_{LC}$. Similarly, we can get the following theorem.

Theorem 3:

Assume $C_0(T) = C_{01} + (C_0 \int_{Ts}^{T} w(t)dt)^m$, C_{01}, C_0>0, C_1>0, C_2>0, C_3>0, C_2>C_1, we have

CASE $(C_2^* - C_1^*)ar\exp[-r(W(Ts) - W(0))] > C_3$ and $P(T_{LC})<C_3$: there exists a finite and unique solution T_0 satisfying Eq. (25) and the optimal software release time is $T^* = T_0$.

CASE $(C_2^* - C_1^*)ar\exp[-r(W(Ts) - W(0))] < C_3$: $T^* = T_s$.

CASE $P(T_{LC})>C_3$: $T^* = T_{LC}$.

D. $C_0(T) = C_{01} + C_0 \times (\exp[m\int_{Ts}^{T} w(t)dt] - 1)$, $T \geq Ts$; $C_0(T)$= 0, $T<T_s$.

$$\frac{d}{dT}C2(T) = C_0 mw(T)\exp[m\int_{Ts}^{T} w(t)dt] + C_1^* arw(T) \times$$

$$\exp[-rW^*(T)] - C_2^* arw(T)\exp[-rW^*(T)] +$$

$$C_3 \times w(T)$$

$$= w(t) \times \{(C_1^* - C_2^*)ar\exp[-rW^*(T)] +$$

$$C_3 + C_0 \times m\exp[m\int_{Ts}^{T} w(t)dt]\}$$

Since $w(t)$>0 for $0 \leq T < \infty$, $\frac{d}{dT}C2(T) = 0$ if

$$Q(T) \equiv (C_2^* - C_1^*) \times ar\exp[-rW^*(t)] - C_0 m \times$$

$$\exp[m\int_{Ts}^{T} w(t)dt] = C_3 \qquad (26)$$

The left-hand side in Eq. (26) is a monotonically decreasing function of T. Therefore, if $(C_2^* - C_1^*)ar\exp[-r(W(T_s) - W(0))] - C_0 m > C_3$ and $Q(T_{LC})$ $<C_3$, it means that there exists a finite and unique solution T_0 satisfying Eq. (26), which can be solved by numerical methods [26]. It is noted that $\frac{d}{dT}C2(T) < 0$ for $0 \leq T_S \leq T < T_0$ and $\frac{d}{dT}C2(T)$ >0 for T>T_0. Thus, T=T_0 minimizes C2(T) for T_0 <T_{LC}. Similarly, we can get the following theorem.

Theorem 4:

Assume $C_0(T) = C_{01} + C_0 \times (\exp[m\int_{Ts}^{T} w(t)dt] - 1)$, C_{01}>0, C_0>0, C_1>0, C_2>0, C_3>0, C_2>C_1, we have

CASE $(C_2^* - C_1^*)ar\exp[-r(W(Ts) - W(0))] - C_0 m > C_3$ and $Q(T_{LC})<C_3$: there exists a finite and unique solution T_0 satisfying Eq. (26) and the optimal software release time is $T^* = T_0$.

CASE $(C_2^* - C_1^*)ar\exp[-r(W(Ts) - W(0))] - C_0 m < C_3$: T^*=T_s .

CASE $Q(T_{LC})$>C_3, : $T^* = T_{LC}$.

3.3 Numerical example

We have considered several different cases of minimizing the software cost in which the new automated tools and techniques are introduced during testing. Due to the limitation of space, we choose Eq. (10) as the testing-effort function for a software development project. Other logistic testing-effort functions with different κ values can be similarly applied based on the same procedure. From the previously estimated parameters for the first data set in Table 2, we get N=48.7768, A=429.673, α=0.158042, κ=2.63326, a=369.029, r=0.0509553. We further set C_{01}=\$1000, C_1=\$10 per error, C_2=\$50 per error, C_3=\$100 per unit testing-effort expenditures, and T_{LC}=100 weeks. We will consider the following two types of cost function $C_0(T)$:

1. $C_0(T) = C_{01} + (C_0 \int_{Ts}^{T} w(t)dt)^m$
2. $C_0(T) = C_{01} + C_0 \times (\exp[m\int_{Ts}^{T} w(t)dt] - 1)$

Here we assume C_0=\$10, T_s=19, T_{LC}=100, and m=1, that is, $C_0(T) = 1000 + 10\int_{19}^{100} w(t)dt$. From Theorem 3, the relationship of the cost optimal release time with different P is given in Table 9. From Table 9, we find that if the P value is larger, the optimal release time is larger and the total expected software cost is smaller. This reflects that when we have better testing performance, we can detect more latent faults through additional techniques and tools. Therefore, we can shorten testing time and release software soon. Compared with the estimated values of traditional software cost model (i.e. Eq. (12)) where T*=24.2828, $C(T*)$=4719.66, we can see that in Table 9, same optimal release time is achieved when P=0.10 (i.e., T*=24.2839), then $C(T*)$= 4130.91. It means that the $C2(T)$ is smaller than $C1(T)$ with equal optimal release time; that is, the assumption $C1(T) - C2(T) \geq 0$ is satisfied. Besides, the *Operational Quality Index* (OQI) is increased from 89.15% to 98.062% [7]. Similarly, the relationships of the optimal release time with various P values based on different cost functions are shown in Table 10-14. From these tables we conclude the following facts:

1) As P increases, the optimal release time T* increases but the total expected software cost $C(T*)$ decreases. This is because we can detect more faults and reduce the cost of correcting faults during operational phase.
2) Under the same P value and with different cost

functions (such as $C_0(T) = C_{01} + (C_0 \int_{Ts}^{T} w(t)dt)^m$ or $C_0(T) = C_{01} + C_0 \times (\exp[m\int_{Ts}^{T} w(t)dt] - 1)$), the larger the cost function is, the smaller the optimal release time is. However, the difference in estimating the total expected software cost is insignificant.

Table 9: Relationship between the cost optimal release time *T, *C*(*T**), and *P* based on the cost function**

$$C_0(T) = 1000 + 10 \times \int_{19}^{100} w(t)dt$$

P	*T**	*C*(*T**)	*P*	*T**	*C*(*T**)
0.01	19.7381	5574.05	0.07	21.8541	4613.69
0.02	20.0016	5414.5	0.08	22.4464	4452.94
0.03	20.2887	5254.74	0.09	23.2027	4292.02
0.04	20.6072	5094.77	0.10	24.2839	4130.91
0.05	20.965	4934.6	0.11	26.1106	3969.62
0.06	21.9747	4774.24			

Table 10: Relationship between the cost optimal release time *T, *C*(*T**), and *P* based on the cost function**

$$C_0(T) = 1000 + (10 \times \int_{19}^{100} w(t)dt)^{1.2}$$

P	*T**	*C*(*T**)	*P*	*T**	*C*(*T**)
0.01	19.1465	5574.88	0.07	19.6383	4620.26
0.02	19.2013	5415.96	0.08	19.7589	4460.84
0.03	19.2669	5256.98	0.09	19.8915	4301.32
0.04	19.3433	5097.93	0.10	20.0358	4141.69
0.05	19.4307	4938.8	0.11	20.1936	3981.95
0.06	19.5289	4779.57			

Table 11: Relationship between the cost optimal release time *T, *C*(*T**), and *P* based on the cost function**

$$C_0(T) = 1000 + (\exp[\int_{19}^{100} w(t)dt] - 1)$$

P	*T**	*C*(*T**)	*P*	*T**	*C*(*T**)
0.01	21.3447	5565.5	0.07	23.0113	4601.14
0.02	21.5892	5405.0	0.08	23.3608	4440.12
0.03	21.8434	5244.41	0.09	23.747	4279.03
0.04	22.1096	5083.72	0.10	24.1866	4117.88
0.05	22.3909	4922.94	0.11	24.682	3956.67
0.06	22.6902	4762.08			

Table 12: Relationship between the cost optimal release time *T, *C*(*T**), and *P* based on the cost function**

$$C_0(T) = 1000 + (\exp[1.2 \times \int_{19}^{100} w(t)dt] - 1)$$

P	*T**	*C*(*T**)	*P*	*T**	*C*(*T**)
0.01	20.8548	5565.97	0.07	21.8278	4603.35
0.02	21.0159	5405.72	0.08	21.9943	4442.68
0.03	21.1771	5245.39	0.09	22.163	4281.96
0.04	21.3384	5084.98	0.10	22.3349	4121.18
0.05	21.5003	4924.5	0.11	22.5104	3960.35
0.06	21.6634	4763.96			

Table 13: Relationship between the cost optimal release time *T, *C*(*T**), and *P* based on the cost function**

$$C_0(T) = 1000 + 5 \times (\exp[\int_{19}^{100} w(t)dt] - 1)$$

P	*T**	*C*(*T**)	*P*	*T**	*C*(*T**)
0.01	19.775	5572.13	0.07	20.3058	4613.99
0.02	19.8669	5412.61	0.08	20.3903	4454.09
0.03	19.9572	5253.02	0.09	20.4739	4294.13
0.04	20.0461	5093.36	0.10	20.5568	4134.12
0.05	20.1338	4933.63	0.11	20.639	3974.06
0.06	20.2203	4773.84			

Table 14: Relationship between the cost optimal release time *T, *C*(*T**), and *P* based on the cost function**

$$C_0(T) = 1000 + 5 \times (\exp[1.2 \times \int_{19}^{100} w(t)dt] - 1)$$

P	*T**	*C*(*T**)	*P*	*T**	*C*(*T**)
0.01	19.545	5573.23	0.07	19.9397	4616.49
0.02	19.6152	5413.91	0.08	20.0002	4456.85
0.03	19.6834	5254.54	0.09	20.0594	4297.17
0.04	19.7499	5095.11	0.10	20.1175	4137.45
0.05	19.8147	4935.62	0.11	20.1745	3977.68
0.06	19.8779	4776.08			

4. Summary and conclusions

In this paper we study the impact of software testing effort and efficiency on the modeling of software reliability, including the reliability measure and the cost for optimal release time. We propose a generalized logistic testing-effort function which relates work profile directly to the natural flow of software development. This function is used to describe the actual consumption of resources during software testing which provides more accurate information for reliability modeling purpose. We also describe the effects of applying new tools and techniques for increased efficiency of software testing and studied the related optimal software release time problem from the cost-benefit viewpoint. New reliability problems are formulated to incorporate software testing effort and efficiency. Finally, numerical examples are provided to demonstrate these new approaches.

5. Acknowledgments

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 also partially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region (Project No. CUHK4432/99E). The authors are pleased to thank Professor Y. K. Malaiya for his many constructive and insightful suggestions for improving the details of this manuscript. Besides, we also thank several anonymous referees for their critical review and comments.

References

[1] 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.

[2] 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, April 1986.

[3] S. Yamada and S. Osaki, " Cost-Reliability Optimal Release Policies for Software Systems", *IEEE Trans. on Reliability*, Vol. 34, No. 5, pp. 422-424, 1985.

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

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

[6] M. E. Helander, M. Zhao, and N. Ohisson, "Planning Models for Software Reliability and Cost," *IEEE Trans. on Software Engineering*, Vol. 24, No. 6, pp. 420-434, June 1998.

[7] 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* (ISSRE'98), pp. 111-123, Nov. 4-7. 1998, Paderborn, Germany.

[8] 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* (ISSRE'97), pp. 378-388, Nov. 1997, Albuquerque, New Mexico. U.S.A.

[9] R. H. Huo, S. Y. Kuo, and Y. P. Chang, "Optimal Release Times for Software Systems with Scheduled Delivery Time Based on HGDM," *IEEE Trans. on Computers*, Vol. 46, No. 2, pp. 216-221, Feb. 1997.

[10] R. H. Huo, S. Y. Kuo, and Y. P. Chang, "Optimal Release Policy for Hyper-Geometric Distribution Software Reliability Growth Model," *IEEE Trans. on Reliability*, Vol. 45, No. 4, pp. 646-651, Dec. 1996.

[11] K. Okumoto and A. L. Goel, "Optimum Release Time for Software Systems Based on Reliability and Cost Criteria", *Journal of Systems and Software*, Vol. 1, pp. 315-318, 1980.

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

[13] S. R. Dalal and C. L Mallows, "When Should One Stop Testing Software, " *Journal of the American Statistical Association*, Vol. 83, No. 403, pp. 872-879, September 1988.

[14] F. N. Parr, "An Alternative to the Rayleigh Curve for Software Development Effort," *IEEE Trans. on Software Engineering*, SE-6, pp. 291-296, 1980.

[15] M. Lipow, "Prediction of Software Failures, " Journal *of Systems and Software*, Vol. 1, pp. 71-75, 1979.

[16] B. Boehm (1981). *Software Engineering* Economics. Prentice-Hall, Englewood Cliffs, NJ.

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

[18] Y. W. Leung, "Optimum Software Release Time with a Given Budget", *Journal of Systems and Software*, Vol. 17, pp. 233-242, 1992.

[19] S. Yamada, H. Narihisa, and S. Osaki, " Optimum Release Policies for a Software System with a Scheduled Delivery Time," *Int. J. of Systems Science*, Vol. 15, pp. 905-914, 1984.

[20] J. Farquhar and A. Mosleh, "An Approach to Quantifying Reliability-Growth Effectiveness," *Proceedings Annual Reliability and Maintainability Symposium*, pp. 166-173, 1995.

[21] S. S. Gokhale, M. R. Lyu, and K. S. Trivedi, "Software Reliability Analysis Incorporating Fault Detection and Debugging Activities," *Proceedings of the 9th International Symposium on Software Reliability Engineering* (ISSRE'98), pp. 202-211, November 4-7 1998, Paderborn, Germany.

[22] P. N. Misra, "software reliability analysis," *IBM Systems Journal*, Vol. 22, No. 3, pp. 262-279, 1983.

[23] Y. Tohma, R. Jacoby, Y. Murata, and M. Yamamoto, "Hyper-Geometric Distribution Model to Estimate the Number of Residual Software Faults," *Proc. COMPSAC-89*, Orlando, pp. 610-617, 1989.

[24] C. Y. Huang, S. Y. Kuo, and M. R. Lyu, " Optimal Software Release Policy Based on Cost, Reliability and Testing Efficiency," *The Twenty-Third Annual International Computer Software and Applications Conference* (COMPSAC'99), October 27-29, 1999, Phoenix, Arizona, U.S.A. (accepted for publication)

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

[26] Xie, M., *Software Reliability Modeling*, World Scientific Publishing Company, 1991.