Optimal Release Time for Software Systems Considering Cost, Testing-Effort, and Test Efficiency

Chin-Yu Huang, *Member, IEEE,* and Michael R. Lyu, *Fellow, IEEE*

***Abstract*—In this paper, we study the impact of software testing effort & efficiency on the modeling of software reliability, including the cost for optimal release time. This paper presents two important issues in software reliability modeling & software reliability economics: testing effort, and efficiency. First, we propose a generalized *logistic* testing-effort function that enjoys the advantage of relating work profile more directly to the natural flow of software development, and can be used to describe the possible testing-effort patterns. Furthermore, we incorporate the generalized *logistic* testing-effort function into software reliability modeling, and evaluate its fault-prediction capability through several numerical experiments based on real data. Secondly, we address the effects of new testing techniques or tools for increasing the efficiency of software testing. Based on the proposed software reliability model, we present a software cost model to reflect the effectiveness of introducing new technologies. Numerical examples & related data analyzes are presented in detail. From the experimental results, we obtain a software economic policy which provides a comprehensive analysis of software based on cost & test efficiency. Moreover, the policy can also help project managers determine when to stop testing for market release at the right time.**

***Index Terms*—Non-homogenous Poisson process, optimal release time, software cost model, software reliability, testing-effort.**

***Acronyms*[1]**

NHPP	nonhomogeneous Poisson process
MLE	maximum likelihood estimation
LSE	least squares estimation
SRGM	software reliability growth model

Notation

$m(t)$	mean value function, i.e., the expected number of software failures by time t
m_i	cumulative number of detected faults in a given time interval $(0, t_k]$
$w(t)$	current testing-effort estimated by a *logistic* testing-effort function
$W(t)$	cumulative testing-effort estimated by a *logistic* testing-effort function
$w_K(t)$	current testing-effort estimated by a generalized *logistic* testing-effort function
$W_K(t)$	cumulative testing-effort estimated by a generalized *logistic* testing-effort function
W_k	cumulative testing-effort actually consumed in time $(0, t_k]$
a	expected number of initial faults
M_a	actual cumulative number of detected faults after the test
r	fault detection rate per unit testing-effort
N	total amount of testing-effort eventually consumed
α	consumption rate of testing-effort expenditures in the *logistic* testing-effort function
A	constant parameter in the *logistic* testing-effort function
κ	structuring index whose value is larger for better structured software development efforts
P	additional fraction of faults detected during testing
T_{LC}	software life-cycle length
T_S	start time of adopting new techniques/methods.
C_1	cost of correcting an error during testing
C_2	cost of correcting an error during operation, $C_2 > C_1$
C_3	cost of testing per unit testing-effort expenditures
$C_0(t)$	cost function for acquiring or developing the automated test tools or new techniques
C_{01}	nonnegative real number that indicates the basic cost of adopting new tools or techniques
C_0	unit new-added test cost

Manuscript received June 4, 2003. This research was supported by the National Science Council, Taiwan, under Grant NSC 93-2213-E-007-088, and by a grant from the Ministry of Economic Affairs (MOEA) of Taiwan (Project no. 94-EC-17-A-01-S1-038). This research was also substantially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project no. CUHK4205/04E). Associate Editor: J. D. Healy.

C.-Y. Huang is with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan (e-mail: cyhuang@cs.nthu.edu.tw).

M. R. Lyu is with the Computer Science and Engineering Department, The Chinese University of Hong Kong, Shatin, Hong Kong (e-mail: lyu@cse.cuhk.edu.hk).

Digital Object Identifier 10.1109/TR.2005.859230

[1]The singular and plural of an acronym are always spelled the same.

I. Introduction

As COMPUTER applications permeate our daily life, reliability becomes a very important characteristic of the computer systems. Software reliability is consequently one of the most important features for a critical software system. According to the ANSI definition [1], software reliability is defined as the probability of failure-free software operation for a specified period of time under a specified environment. In practice, it is very difficult for the project managers to measure software reliability & quality. Since the early 1970s, a number of *Software Reliability Growth Models* (SRGM) have been proposed for the estimation of reliability growth of products during software development (SD) processes [1]–[3]. Among these models, Goel and Okumoto [1] used a NHPP as the stochastic process to describe the fault process. Yamada *et al.* [4]–[6] modified the Goel-Okumoto model, and incorporated the concept of testing-effort (TE) into an NHPP model to get a better description of the software fault phenomenon. Recently, we [7]–[11] proposed a

new SRGM with a *logistic* testing-effort function (TEF). A *logistic* TEF is a software development & test effort curve with a good predictive capability.

In this paper, we generalize the *logistic* TEF. The generalized *logistic* TEF has the advantage of relating a work profile more directly to the natural structure of the SD. Thus, it can be used to pertinently describe the resource consumption during the SD process, and provides a significant improvement in modeling the distribution of TE expenditures. We will show that the proposed model is easy to use, implement, and interpret; and it works quite well in the testing stage.

The more software tests we run, the more confidence we obtain in the measured software reliability. Unfortunately, testing with ineffective or redundant test cases may lead to excessive cost. To avoid this phenomenon, we need to know when to stop testing [12]. One alternative is to restrict the test data such that testing will stop when the odds of detecting additional faults (estimated by SRGM) are very low. But this may not be realistic because testers typically want to test for all possible failures. It was shown that we could get the optimal software release time based on a cost criterion when minimizing the total expected cost [1], [6]. Recently, many papers have discussed the optimal software release time problem considering cost & reliability [10], [11], [13]–[16].

To detect additional faults during the testing phase, new automated test tools or methods to create tests & eliminate redundant test cases may be employed. As time progresses, they can detect additional faults, which reduces the expenses of correcting faults during the operational phase [17], [18]. These approaches have improved software testing & productivity recently, allowing project managers to improve software reliability. We will further discuss the optimal release problem considering cost, TE, and efficiency. That is, based on the proposed SRGM with *logistic* TEF, we present a software cost model to incorporate the effectiveness of introducing new technologies. The methods we propose can help the software test engineers & software quality assurance engineers in deciding when the software is likely to be of adequate quality for release.

The paper is organized as follows. Section II gives an SRGM with generalized *logistic* TEF. Some numerical examples are illustrated in Section III. Finally, Section IV introduces the concept of testing efficiency factor obtained by new test tools or techniques during testing, as well as the associated optimal software release problem.

II. Software Reliability Modeling, and Testing-Effort Function

In this section, we propose a set of generalized software reliability growth models incorporating TEF. The mathematical relationship between reliability models & TE expenditures is also described. In addition, numerical results are given to illustrate the advantages of this new approach.

A. Review of SRGM With Logistic TEF

A basic SRGM is based on the assumptions:

1) The fault removal process follows a NHPP.
2) The software system is subject to failures at random times caused by the manifestation of remaining faults in the system.
3) The mean number of faults detected in the time interval $(t, t+\Delta t]$ by the current TE expenditures is proportional to the mean number of remaining faults in the system.
4) The proportionality constant is homogenous with respect to time (i.e., it does not change with time).
5) The time-dependent behavior of TE can be modeled by a *logistic* distribution.
6) Each time a failure occurs, the fault that caused it is immediately & correctly removed. Besides, no new faults are introduced.
7) Correction of errors takes only negligible time, and a detected error is removed with certainty.

Therefore, we first describe the SRGM based on TEF as follow [8]–[11]:

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

Note that (1) includes two components which influence the number of faults detected: the TE function $w(t)$, and the fault detection rate r. Solving the above differential equation with the boundary condition $m(t) = 0$, we have

$$\begin{aligned} m(t) &= a\Big(1 - \exp\Big[-r\big(W(t) - W(0)\big)\Big]\Big) \\ &= a\Big(1 - \exp\big[-rW^*(t)\big]\Big). \end{aligned} \tag{2}$$

Equation (2) is an NHPP model with mean value function (MVF) considering the TE consumption. It is noted that $W(t)$ represents the cumulative TE consumption (such as CPU time, volume of test cases, human power, and so on) at time t during the testing phase. The consumed TE can indicate how effectively 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 past studies in [4]–[6], [19], several TE expressions, such as Exponential, Rayleigh, and Weibull-type curves, etc, can be applied.

Recently, we [7]–[9] proposed a *logistic* TEF to describe the possible TE patterns, in which the current TE consumption is expressed as

$$w(t) = \frac{NA\alpha \exp[-\alpha t]}{\big(1 + A\exp[-\alpha t]\big)^2} = \frac{NA\alpha}{\Big(\exp\big[\frac{\alpha t}{2}\big] + A\exp\big[-\frac{\alpha t}{2}\big]\Big)^2}. \tag{3}$$

Because

$$W(t) = \int_0^t w(\tau) d\tau \tag{4}$$

we obtain

$$W(t) = \frac{N}{1 + A\exp[-\alpha t]}. \tag{5}$$

The current TE $w(t)$ reaches its maximum value at time [7], [8]

$$t_{\max} = \frac{1}{\alpha} \ln A. \tag{6}$$

B. A New Generalized Logistic TEF

From previous studies [7]–[9], we know that the *logistic* TEF (i.e. the Parr model [20]) can be used to describe the work profile of SD. In addition, this function can be used to consider & evaluate the effects of possible improvements on SD methodology, such as top-down design, or stepwise refinement. DeMarco [21] also reported that this function was found to be fairly accurate by the Yourdon 1978–1980 project survey. If we relax some assumptions in the original Parr model, and take into account the structured SD effort, we obtain a generalized *logistic* TEF as

$$W_\kappa(t) = N \times \left(\frac{\frac{(\kappa+1)}{\beta}}{1 + A\exp[-\alpha\kappa t]} \right)^{\frac{1}{\kappa}}. \tag{7}$$

When $\kappa = 1$, the above equation becomes

$$W_1(t) = \frac{N}{1 + A\exp[-\alpha t]} \times \frac{2}{\beta}. \tag{8}$$

If β is viewed as a normalized constant, and $\beta = 2$, the above equation is reduced to (5). Similarly, if $\kappa = 2$, we have

$$W_2(t) = \frac{N}{\sqrt{1 + A\exp[-2\alpha t]}} \sqrt{\frac{3}{\beta}}. \tag{9}$$

Note if we set $\beta = \kappa + 1$, we get:

$$W_\kappa(t) = \frac{N}{\sqrt[\kappa]{1 + A\exp[-\alpha\kappa t]}} \tag{10}$$

and

$$\begin{aligned} w_\kappa(t) &\equiv \frac{dW_\kappa(t)}{dt} \\ &= \alpha N A \left(\exp\left[\frac{\alpha\kappa^2 t}{\kappa+1}\right] + A\exp\left[\frac{-\alpha\kappa t}{\kappa+1}\right] \right)^{\frac{-\kappa-1}{\kappa}}. \end{aligned} \tag{11}$$

In this case, the current TE $w_k(t)$ reaches its maximum value at time

$$t^\kappa_{\max} = \frac{\ln \frac{A}{\kappa}}{\alpha\kappa}. \tag{12}$$

Furthermore, an SRGM with a generalized *logistic* TEF can be described as

$$\begin{aligned} m(t) &= a\Big(1 - \exp\big[-r\big(W_\kappa(t) - W_\kappa(0)\big)\big]\Big) \\ &= a\Big(1 - \exp\big[-rW^*_\kappa(t)\big]\Big). \end{aligned} \tag{13}$$

III. Experimental Studies, and Results

A. Descriptions of Real Data Sets

In this section, we evaluate the performance of SRGM with generalized *logistic* TEF by using two sets of software failure data. The first set of real data is from a study by [22]. The system is a PL/I database application software consisting of approximately 1 317 000 lines of code. During nineteen weeks, 47.65 CPU hours were consumed, and 328 software faults were removed. The total cumulative number of detected faults after a long time of testing is 358 [23]. The second data set in this paper is the System T1 data of the Rome Air Development Center (RADC) projects in [2], [3]. The system T1 is used for a real-time command & control application. In this case, the size of the software is approximately 21 700 object instructions. It took twenty-one weeks, and nine programmers to complete the test. During the test phase, about 25.3 CPU hours were consumed, and 136 software faults were removed. The number of faults removed after 3.5 years of test is 188 [23].

B. Comparison Criteria

Two criteria are used for model comparisons:

1) The *Accuracy of Estimation* (AE) is defined [4]–[6] as

$$\left| \frac{M_a - a}{M_a} \right|. \tag{14}$$

For practical purposes, M_a is obtained from software fault tracking after software testing.

2) Long-term predictions: the *Mean Square Error* (MSE) is defined as [8], [9], [24]

$$\frac{\sum_{i=1}^{n} \big[m(t_i) - m_i\big]^2}{n}. \tag{15}$$

In our quantitative comparisons, we use MSE because we believe that it is easier to understand. A smaller MSE indicates a smaller fitting error, and better overall performance.

Besides, to check the performance of the generalized *logistic* TEF, and to make a fair comparison with other TEF we also select some comparison criteria [25]–[28]:

- $Prediction\ \ Error\ \ (PE_i) = Actual(Observed)_i - Predicted(Estimated)_i$
- $Variation = \sqrt{(\sum_{i=1}^{n}(PE_i - Bias)^2)/(n-1)}$.
- $Bias = \sum_{i=1}^{n}(PE_i/n)$.
- Magnitude of Relative Error (MRE)= $|(M_{estimated} - M_{actual})/M_{actual}|$.

C. Estimation of Model Parameters by Least Square Method

To validate the proposed SRGM with a generalized *logistic* TEF in (13), experiments on two real test data sets have been performed. Two popular estimation techniques are MLE, and LSE [2]. The maximum likelihood technique estimates parameters by solving a set of simultaneous equations. However, the equation sets may be very complex, and usually must be solved numerically. The method of least squares minimizes the sum of squares of the deviations between what we actually observe, and what we expect. Therefore, we decided to fit the *logistic* TEF, and the proposed model directly with the above two data sets, using the least sum of squares criterion to give a "*best fit*". Here we can easily use MSE to judge the performance of the models.

The parameters N, A, α, and κ of the generalized *logistic* TEF in (10); and the parameters a, and r given in (13) can be estimated by the method of least squares. For the method of least square sum, the evaluation formula $S1(N, A, \alpha, \kappa)$, and $S2(a, r)$ are as:

$$Minimize\ S1(N, A, \alpha, \kappa) = \sum_{k=1}^{n} \left[W_k - W_\kappa(t_k)\right]^2 \quad (16)$$

$$Minimize\ S2(a, r) = \sum_{k=1}^{n} \left[m_k - m(t_k)\right]^2. \quad (17)$$

Differentiating $S1(S2)$ with respect to N, A, α, and $\kappa(a, r)$ respectively, setting the partial derivatives to zero, and rearranging these terms, we can solve this type of nonlinear least square problems. For a simple illustration, we only consider the generalized *logistic* TEF with $\kappa = 1$, i.e., (5).

Take the partial derivatives of $S1$ with respect to N, A, and α; we get

$$\frac{\partial S1}{\partial N} = \sum_{k=1}^{n} 2\left(W_k - \frac{N}{1 + A\exp[-\alpha t]}\right)\frac{1}{1 + A\exp[-\alpha t]} = 0. \quad (18)$$

Thus, the least squares estimator N is given by solving the above equation, yielding

$$N = \frac{\sum_{k=1}^{n} 2\left(\frac{W_k}{1+A\exp[-\alpha t]}\right)}{\sum_{k=1}^{n} 2\left(\frac{1}{1+A\exp[-\alpha t]}\right)^2}. \quad (19)$$

Next, we have

$$\frac{\partial S1}{\partial A} = \sum_{k=1}^{n} 2\left(W_k - \frac{N}{1 + A\exp[-\alpha t]}\right)\frac{N\exp[-\alpha t]}{\left(1 + A\exp[-\alpha t]\right)^2} = 0 \quad (20)$$

and

$$\frac{\partial S1}{\partial \alpha} = \sum_{k=1}^{n} 2\left(W_k - \frac{N}{1+A\exp[-\alpha t]}\right)\frac{NAt\exp[-\alpha t]}{(1+A\exp[-\alpha t])^2} = 0. \quad (21)$$

The other parameters A & α can also be obtained by substituting the least squares estimator N into the above two equations.

D. Model Comparison With Real Applications

In this section, we test the predictive power of the proposed model & other existing SRGM using two sets of software failure data.

1) DS 1: For the first data set, LSE is used to estimate the parameters of (10) & (13). The estimated parameter values of the generalized *logistic* TEF are listed in Table I(a) for various values of κ. The proposed model estimates $\kappa = 2.633\,26$ for this data set. The computed $bias$, $Variation$, MRE, and $PE_{\text{end\_of\_testing}}$ for the generalized *logistic* TEF; and the *Weibull*-type TEF are listed in Table I(b). Fig. 1(a) graphically illustrates the fitting of the estimated testing effort by using (5), *Weibull*, *Rayleigh*, and *exponential* functions. Besides, Fig. 1(b) depicts the fit of the estimated current TE by using (10) with different values of κ. This shows that the peak work rate occurs when about half of the work on the project has been done. This phenomenon suggests that, in a well-structured SD environment, the slope of the TE consumption curve may grow slowly initially, but a compensating reduction will happen later. Fig. 1(c) graphically illustrates the comparisons between the observed data, and the data estimated by the proposed model. Table I(c) shows the estimated values of parameters a, and r which minimize MSE for different SRGM. The comparison criterion AE is also listed. From Tables I(b), (c), and Fig. 1(c), we can see that the proposed model achieves a better goodness-of-fit than other SRGM.

TABLE I
(a) PARAMETERS OF GENERALIZED LOGISTIC TEF FOR THE FIRST DATA SET (b) COMPARISON RESULTS FOR DIFFERENT TEF BASED ON THE FIRST DATA SET (c) COMPARISON RESULTS FOR THE FIRST DATA SET

κ	N	A	α
1	54.8364	13.0334	0.2263
1.5	52.0072	40.6042	0.1888
2	50.2178	115.228	0.1700
2.63326	48.7768	429.673	0.1580
3.5	48.1693	2188.22	0.1442
4	47.8507	5709.29	0.1399

(a)

TEFS	BIAS	VARIATION	MRE	$PE_{\text{END OF TESTING}}$
Proposed TEF (κ=1)	-0.1	1.30724	0.022161	1.05597
Proposed TEF (κ=1.5)	-0.12143	1.43605	0.026402	1.25806
Proposed TEF (κ=2)	-0.14163	1.54713	0.029944	1.42681
Proposed TEF (κ=2.63326)	-0.11617	1.67112	0.031977	1.52368
Proposed TEF (κ=3.5)	-0.2211	1.79064	0.028625	1.36399
Proposed TEF (κ=4)	-0.24341	1.851	0.027663	1.31815
Rayleigh Distribution	0.8302	2.169	0.0525	2.5038
Exponential Distribution	-0.3938	1.3685	0.0266	1.2688
Weibull Distribution	0.0342	0.9559	-0.0079	-0.3775

(b)

MODEL	A	R	AE (%)	MSE
Proposed Model (κ=1)	394.08	0.0427223	10.06	118.29
Proposed Model (κ=1.5)	384.71	0.0450374	7.46	114.32
Proposed Model (κ=2)	377.16	0.0478156	5.35	112.41
Proposed Model (κ=2.63326)	369.03	0.0509553	3.08	107.73
Proposed Model (κ=3.5)	412.87	0.0399382	15.32	120.76
Proposed Model (κ=4)	414.43	0.0398619	15.76	189.21
Goel & Okumoto Model	760.00	0.0322688	112.29	139.82
Delayed S-Shaped Model	374.05	0.1976510	14.48	168.67
Inflection S-Shaped Model	389.10	0.0935493	8.69	133.53
G-O Model with *Weibull* TEF	565.35	0.0196597	57.91	122.09
G-O Model with *Rayleigh* TEF	459.08	0.0273367	28.23	268.42
G-O Model with *exponential* TEF	828.25	0.0117836	131.35	140.66

(c)

2) DS 2: Similarly, the LSE is also applied to estimate the parameters of (10) & (13). The estimated parameter values of the generalized logistic TEF are listed in Table II(a). The proposed model estimates $\kappa = 1.271\,71$ for this data set. Additionally, other possible values of κ are listed in Table II(a). The computed $bias$, $Variation$, MRE, and $PE_{\text{end\_of\_testing}}$ for the generalized *logistic*, *Rayleigh*, and *exponential* TEF are illustrated in Table II(b). Fig. 2(a) graphically illustrates the fitting of the estimated TE by using (5), *Rayleigh*, and *exponential* functions. Fig. 2(b) depicts the fit of the estimated current TE by using different TEF, and Fig. 2(c) graphically shows the

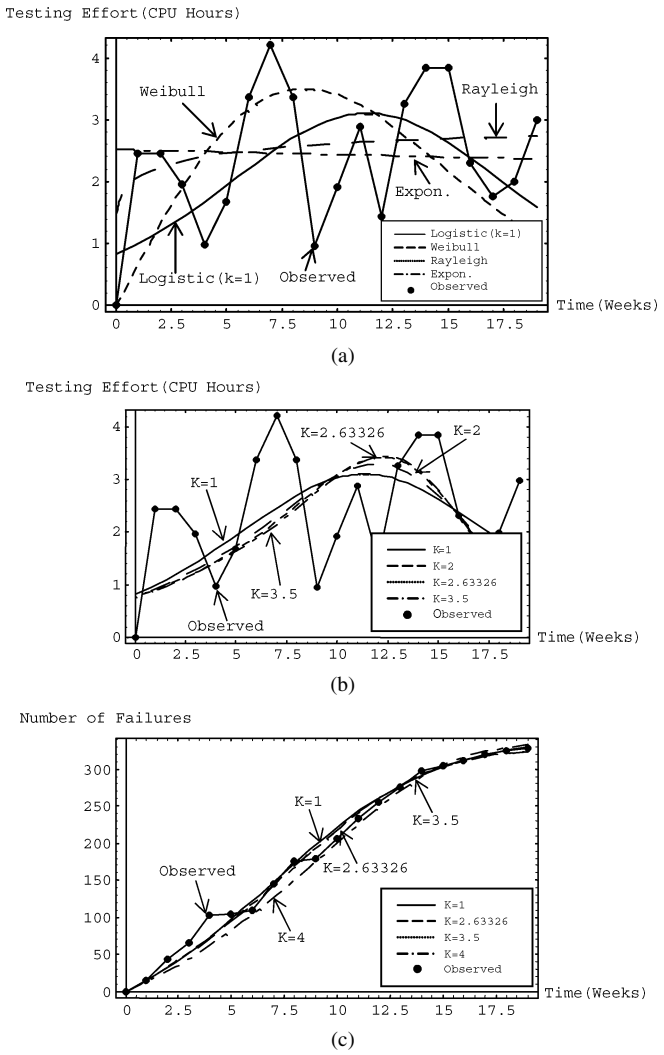

Fig. 1(a) Observed/estimated different TEF vs. time for the first data set. (b) Observed/estimated TE (by using equation (10) with different values of κ) vs. time for the first data set. (c) Estimated MVF vs. time for the first data set.

comparisons between our models with respect to the observed data. Besides, Table II(c) shows the estimated values of parameters by using different SRGM, and the comparison criteria AE & MSE. As seen from Table II(c), our proposed models have much lower MSE than their competitors. Finally, because a good SRGM should be able to predict the behavior of future failures well, Fig. 2(d) shows a u-plot analysis of predictions from the selected models on DS2 [1]. It is obvious that the u-plot of the proposed model is close to the line of unit slope. Altogether, the proposed model provides a more accurate description of resource consumption during the SD phase, and gives a better data fit for this data set.

IV. Optimal Software Release Policies

Software reliability growth models can capture the quantitative aspects of the software testing process, and be used to provide a reasonable software release time. During the software testing phase, the developers can use the SRGM to determine when to stop testing. If the reliability goal is achieved, the software product is ready for release. Several approaches can be applied. For example, we can control the modified consumption

TABLE II
(a) Parameters of Generalized Logistic TEF for the Second Data Set
(b) Comparison Results for Different TEF Based on the Second Data Set(c) Comparison Results for the Second Data Set

κ	N	A	α
1	29.1095	4624.89	0.4935
1.27171	28.153	20903.9	0.4447
1.5	28.1513	45843.8	0.3974
2	28.1458	260550	0.3330
3	28.0464	3784150	0.2572

(a)

TEF	BIAS	VARIATION	MRE	$PE_{\text{END OF TESTING}}$
Proposed Model (κ=1)	0.055515	0.35073	0.004002	-0.10125
Proposed Model (κ=1.27171)	0.026032	0.339497	0.00011	-0.00278
Proposed Model (κ=1.5)	-0.06972	0.359071	0.003385	-0.08565
Proposed Model (κ=2)	-0.19428	0.470271	0.007307	-0.18488
Proposed Model (κ=3)	-0.55257	0.824229	0.003663	-0.09267
Rayleigh Distribution	-1.1	3.4579	0.2384	6.0332
Exponential Distribution	-17	6.3495	-0.5254	-13.294

(b)

MODEL	A	R	AE (%)	MSE
Proposed Model (κ=1)	138.026	0.145098	26.58	62.41
Proposed Model (κ=1.27171)	140.013	0.137916	25.52	23.79
Proposed Model (κ=1.5)	139.191	0.141159	25.96	25.04
Proposed Model (κ=2)	142.505	0.12406	24.19	27.62
Proposed Model (κ=3)	147.808	0.103272	21.37	29.17
Goel & Okumoto Model	142.32	0.1246	24.29	2438.3
Exponential Model	137.20	0.156	27.12	3019.66
Delayed S-Shaped Model	237.196	0.0963446	26.16	245.25
G-O Model with *Rayleigh* TEF	866.94	0.00962474	361.13	89.24

(c)

rate of TE expenditures, and detect more faults in a prescribed time interval [5], [8]. In addition to controlling the TE expenditures, we can achieve a given operational quality at a specified time by introducing new automated testing tools & techniques to find those faults not easily exposed during the earlier manual testing phase [17], [18]. In the following, based on the proposed SRGM, constructive rules are developed for determining optimal software release time.

A. Optimal Software Release Time Problem

To shift the software from the testing phase to the operational phase, theoretical determination of the release time of software is a very important problem in terms of economics. In recent years, the problem of optimal software release time has been analyzed & discussed by many papers [6], [9]–[16], [29]–[35]. The release time problem is associated with the cost of a software system. Okumoto and Goel [29] first discussed the optimal software release policy from the cost-benefit viewpoint. The total cost of TE expenditures at time T, $C1(T)$, can be expressed as [36]

$$C1(T) = C_1 m(T) + C_2[m(T_{LC}) - m(T)] + C_3 \int_0^T w_\kappa(t)dt \tag{22}$$

where C_1 is the cost of correcting an error during testing, C_2 is the cost of correcting an error during operation, and C_3 is the cost of testing per unit TE expenditures. From the work by Boehm [37], [38], we know C_2 is usually an order of magnitude greater than C_1.

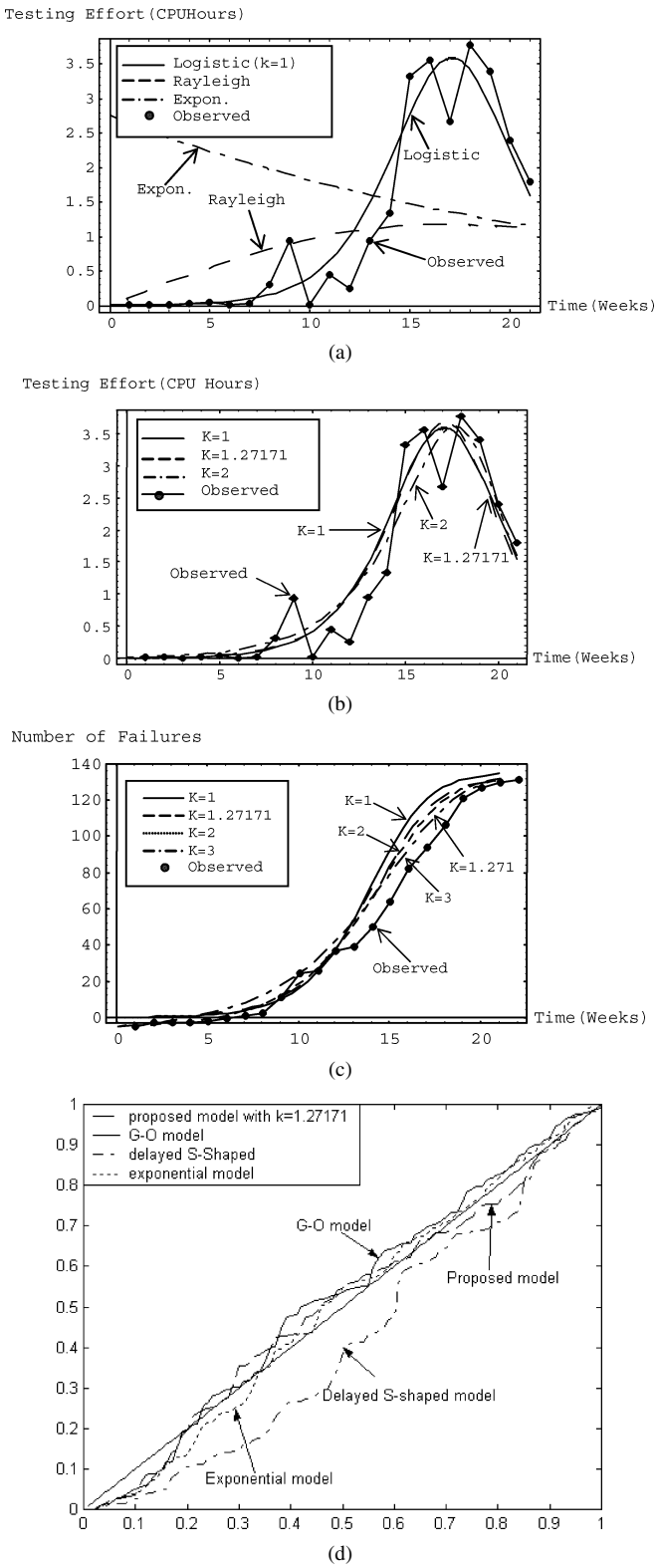

Fig. 2 (a) Observed/estimated different TE vs. time for the second data set. (b) Observed/estimated TE (by using equation (10) with different values of κ) vs. time for the second data set. (c) Estimated MVF vs. time for the second data set. (d) u-plot analysis.

Practically, to detect additional faults during testing, the test engineers may use new automated tools or techniques. The cost trade-off of these new tools & techniques, therefore, should be considered in the software cost model, including their expenditures & benefits. To study the effect of improving test efficiency on the software cost, a software cost model should take account of TE & test efficiency. Consequently, the overall software cost model can be re-formulated as

$$C2(T) = C_0(T) + C_1(1+P)m(T) + C_2\big[m(T_{LC}) - (1+P)m(T)\big] + C_3\int_0^T w_\kappa(t)dt \tag{23}$$

where $C_0(T)$ is the cost function for developing & acquiring the automated test tools or new techniques that can detect an additional fraction P of faults during testing. That is, if P is to be increased, it is expected that the extra costs are needed to engage more experienced testers, or introduce more advanced testing tools.

We note that the cost for developing & acquiring new tools or techniques, $C_0(T)$, does not have to be a constant during the testing. The testing cost for $C_0(T)$ can be parameterized & estimated from actual data. Furthermore, $C_0(T)$ may have different forms as time progresses, which depends on the characteristics of a tool's performance, TE expenditures, effectiveness, and so on. Consequently, we can try to formulate this cost function as a linear function, or a nonlinear function. 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 be preferred if

$$C1(T) - C2(T) \geq 0. \tag{24}$$

From (13), we know

$$\begin{aligned} C_1 m(T) + C_2\big[m(T_{LC}) - m(T)\big] &+ C_3\int_0^T w_\kappa(t)dt - C_0(T) - C_1(1+P)m(T) \\ &- C_2\big[m(T_{LC}) - (1+P)m(T)\big] \\ &- C_3\int_0^T w_\kappa(t)dt \geq 0. \end{aligned} \tag{25}$$

Rearranging the above equation, we obtain

$$C_0(T) \leq P \times m(T) \times (C_2 - C_1). \tag{26}$$

Equation (26) is used to decide whether the new automated tools or methods 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 selected depending on how thoroughly failure data are collected, and faults are categorized [30], [31]. Sometimes incorporating new automated tools & techniques into a SD process may introduce excessive cost, i.e., $C1(T) - C2(T) < 0$. However, if these tools & techniques can really

shorten the testing period under the same software reliability requirements, they are valuable.

By differentiating (23) with respect to the time T, we have

$$\frac{d}{dT}C2(T) = \frac{d}{dT}C_0(T) + C_1\frac{d}{dT}\big((1+P)m(T)\big) - C_2\frac{d}{dT}\big((1+P)m(T)\big) + C_3 w_\kappa(T) \quad (27)$$

If we let (27) be equal to zero, we can get a finite & unique solution T_0 for the optimal software release time problem based on the new cost criterion. From (27), if we let $C1(1+P) = C_1^*$, and $C_2(1+P) = C_2^*$, then we have

$$\frac{d}{dT}C2(T) = \frac{d}{dT}C_0(T) + C_1^*\frac{d}{dT}m(t) - C_2^*\frac{d}{dT}m(T) + C_3 w_\kappa(T). \quad (28)$$

If the MVF is given in (13), we obtain

$$\frac{d}{dT}C2(T) = \frac{d}{dT}C_0(T) + C_1^* arw_\kappa(T)\exp\big[-rW_\kappa^*(T))\big] - C_2^* arw(T)\exp\big[-rW_\kappa^*(T))\big] + C_3 w_\kappa(T). \quad (29)$$

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

1) $C_0(T)$ *is a constant*. For some small projects, it is just enough for the project managers to purchase only common types of automated tools, if they want to detect more faults. Therefore, the extra cost is almost fixed. On the other hand, some companies may need to create their own test tools due to certain project requirements. Sometimes creating these tools is not expensive because they need to hire some engineers, or consider outsourcing [39].
2) $C_0(T)$ *is linearly related to the TE expenditures*. Just as mentioned above, it is enough for the project managers to simply purchase general automated tools if they want to detect more faults, and the scale of the projects is not large. Sometimes test tool vendors offer various types of support, such as software upgrades, patches, or maintenance supports, etc. with different fees [39]. Therefore, the cost of introducing these test tools may not be constant, and it can be linearly increasing in part of the total software cost.
3) $C_0(T)$ *is exponentially related to the TE expenditures*. Due to the complexity of the software, it may not be easy for test engineers to detect additional faults if they only purchase general tools or equipment. Special software applications, such as distributed systems, enormous relational databases, and embedded software, are more difficult to comprehend for test engineers without related experience. Hence, introducing more advanced or new tools into companies may lead to the exponential growth in software development cost. These necessary automated tools or new techniques are expensive ways to detect more faults. For larger projects or on-going long-term projects, they are critical, and sometimes inevitable. Besides, in some cases, exponential increases in the volume of test cases are needed in order to satisfy some criterion during testing. Thus $C_0(T)$ is exponentially related to the TE expenditures.

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

case 1) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_S))] > C_3$, and $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] < C_3$, then there exists a finite, unique solution $T_0 = (1/\alpha)\times\ln(A\Theta^\kappa/N^\kappa - \Theta^\kappa)$, satisfying $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T))] = C_3$, and the optimal software release time is $T^* = T_0$.

case 2) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_S))] < C_3$, then $T^* = T_s$.

case 3) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] > C_3$, then $T^* = T_{LC}$.

Proof: If $C_0(T)$ is a constant; that is, $C_0(T) = C_0$, $T \geq T_s$; $C_0(T) = 0$, $T < T_s$. Taking the first derivative of (23) with respect to the time T, we obtain

$$\frac{d}{dT}C2(T) = w_\kappa(T)\times[-(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T))] + C_3]. \quad (30)$$

Because $w_k(T) > 0$ for $0 < T < \infty$, $H(T) = 0$ if

$$(C_2^* - C_1^*)\,ar\exp\big[-rW_\kappa^*(T))\big] = C_3 \quad (31)$$

The left-hand side of (31) is a monotonically decreasing function of T. There are three cases:

(1) If $(C_2^* - C_1^*)\times ar\exp[-rW_\kappa^*(T_S))] > C_3$, and $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] < C_3$, there exists a finite, unique solution T_0 satisfying (31).

$$T_0 = \frac{1}{\alpha}\times\ln\left(\frac{A\Theta^\kappa}{N^\kappa - \Theta^\kappa}\right) \text{ minimizes } C2(T), \quad (32)$$

where $\Theta = (1/r)(\ln(ar(C_2^* - C_1^*)/C3)))+(N/\sqrt[\kappa]{1+A}$ because $H(T) < 0$ for $T_s < T < T_0$, and $H(T) > 0$ for $T_0 < T < T_{LC}$.

(2) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_S))] \leq C_3$, then $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] < C_3$ for $T_s < T < T_{LC}$. Therefore, the optimal software release time $T^* = T_s$ because $H(T) > 0$ for $T_s < T < T_{LC}$.

(3) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] \geq C_3$, then $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T))] > C_3$ for $T_s < T < T_{LC}$. Therefore, the optimal software release time $T^* = T_{LC}$ because $H(T) < 0$ for $T_s < T < T_{LC}$.

The following theorems can be obtained & proved in a similar manner.

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

case 1) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_S))] > C_3 + C_0$, and $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] < C_3 + C_0$, then there exists a finite, unique solution $T_0 = (1/\alpha)\times\ln(A\Theta^\kappa/N^\kappa - \Theta^\kappa)$, with $\Theta = (1/r)(\ln(ar(C2^* - C1^*)/(C3+C0)))+(N/\sqrt[\kappa]{1+A})$, satisfying $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T))] = C_3 + C_0$, and the optimal software release time is $T^* = T_0$.

case 2) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_S))] < C_3 + C_0$, then $T^* = T_s$.

case 3) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] > C_3 + C_0$, then $T^* = T_{LC}$.

Theorem 3: Assume $\mathrm{C}_0(T) = C_{01} + (C_0 \int_{Ts}^{T} w_\kappa(t)dt)^m$, $C_{01} > 0, C_0 > 0, C_1 > 0, C_2 > 0, C_3 > 0, C_2 > C_1$; then we have

case 1) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_S))] > C_3$, and $[(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] - C_0 m \times (C_0 \int_{Ts}^{T_{LC}} w_\kappa(t)dt)^{m-1}] < C_3$, then there exists a finite, unique solution T_0 satisfying $[(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T))] - C_0 m (C_0 \int_{Ts}^{T} w_\kappa(t)dt)^{m-1}] = C_3$, and the optimal software release time is $T^* = T_0$.

case 2) If $(C_2^* - C_1^*)ar\exp[-r(W_\kappa(T_S) - W_\kappa(0))] < C_3$, then $T^* = T_s$.

case 3) If $[(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] - C_0 m (C_0 \int_{Ts}^{T_{LC}} w_\kappa(t)dt)^{m-1}] > C_3$, then $T^* = T_{LC}$.

Theorem 4: Assume $\mathrm{C}_0(T) = C_{01} + C_0 \times (\int_{Ts}^{T} w_\kappa(t)dt)^m$, $C_{01} > 0, C_0 > 0, C_1 > 0, C_2 > 0, C_3 > 0, C_2 > C_1$; then we have

case 1) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_S))] > C_3$, and $[ar(C_2^* - C_1^*)\exp[-rW_\kappa^*(T_{LC}))] - C_0 m \times (\int_{Ts}^{T_{LC}} w_\kappa(t)dt)^{m-1}] < C_3$, then there is a finite, unique solution T_0 satisfying $[ar(C_2^* - C_1^*)\exp[-rW_\kappa^*(T))] - C_0 m (\int_{Ts}^{T} w_\kappa(t)dt)^{m-1}] = C_3$, and the optimal software release time is $T^* = T_0$.

case 2) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_S))] < C_3$, then $T^* = T_s$.

case 3) If $[ar(C_2^* - C_1^*)\exp[-rW_\kappa^*(T_{LC}))] - C_0 m (\int_{Ts}^{T_{LC}} w_\kappa(t)dt)^{m-1}] > C_3$, then $T^* = T_{LC}$.

Theorem 5: Assume $\mathrm{C}_0(T) = C_{01} + C_0 \times (\exp[m \int_{Ts}^{T} w_\kappa(t)dt] - 1)$, $C_{01} > 0, C_0 > 0, C_1 > 0, C_2 > 0, C_3 > 0, C_2 > C_1$; then we have

case 1) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_S))] - C_0 m > C_3$, and $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] - C_0 m \exp[m \int_{Ts}^{T_{LC}} w_\kappa(t)dt] < C_3$, then there exists a finite, unique solution T_0 satisfying $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T))] - C_0 m \exp[m \int_{Ts}^{T} w_\kappa(t)dt] = C_3$, and the optimal software release time is $T^* = T_0$.

case 2) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_S))] - C_0 m < C_3$, then $T^* = T_s$.

case 3) If $(C_2^* - C_1^*)ar\exp[-rW_\kappa^*(T_{LC}))] - C_0 m \exp[m \int_{Ts}^{T_{LC}} w_\kappa(t)dt] > C_3$, then $T^* = T_{LC}$.

B. Numerical Examples

We have considered several different cases of minimizing the software cost in which the new automated tools & techniques are introduced during testing. As an illustration, we choose (10) (i.e., the generalized *logistic* TEF) as the TE function. In addition to the estimated parameters for the first data set, we further assume $C_{01} = \$1000$, $C_1 = \$10$ per error, $C_2 = \$50$ per error, $C_3 = \$100$ per unit TE expenditures, and $T_{LC} = 100$ weeks. Due to the limitation of space, here we only consider the following two types of cost function $C_0(T)$. Similar conclusions are also obtained from the other cost functions. The selected cases are:

(i) $\mathrm{C}_0(T) = C_{01} + (C_0 \times \int_{Ts}^{T} w_\kappa(t)dt)^m$, and

(ii) $\mathrm{C}_0(T) = C_{01} + C_0 \times (\exp[m \int_{Ts}^{T} w_\kappa(t)dt] - 1)$.

First, suppose $C_0 = \$10$, $T_s = 19$, and $m = 1$; thus $\mathrm{C}_0(T) = 1000 + 10 \times \int_{19}^{T} w_\kappa(t)dt$. Applying Theorem 2, the relationship of the optimal release time & P is given in Table III. From Table III, we see that if the value of P is larger, the optimal release time is larger, and the total expected software cost is smaller. This indicates that when we have better testing performance, we can detect more latent faults through additional techniques & methods. Compared with (22) where $T^* = 24.2828$, $C1(T^*) = 4719.66$, we can see that in Table III, almost the same optimal release time is achieved when $P = 0.10$ (i.e., $T^* = 24.2839$); then $C2(T^*) = 4130.91$. This means that the $C2(T)$ is smaller than $C1(T)$ with the equal optimal release time; that is, the assumption $C1(T) - C2(T) \geq 0$ is satisfied. Besides, the reliability is increased from 0.89 to 0.98 (here the reliability is defined as $R(t) = m(t)/a$). Similarly, the relationship of the optimal release time with various P values based on another cost function $\mathrm{C}_0(T) = 1000 + 5 \times (\exp[1.2 \times \int_{19}^{T} w_\kappa(t)dt] - 1)$ is also shown in Table IV.

TABLE III
RELATIONSHIP BETWEEN THE COST OPTIMAL RELEASE TIME T^*, $C(T^*)$, AND P, BASED ON THE COST FUNCTION $\mathrm{C}_0(T) = 1000 + 10 \times \int_{19}^{T} w_\kappa(t)dt$

P	Optimal Release Time T*	Total Expected Cost C(T*)	P	Optimal Release Time T*	Total Expected Cost C(T*)
0.01	19.7381	5574.05	0.07	21.8541	4613.69
0.02	20.0016	5414.50	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	20.9650	4934.60	0.11	26.1106	3969.62
0.06	21.9747	4774.24			

TABLE IV
RELATIONSHIP BETWEEN THE COST OPTIMAL RELEASE TIME T^*, $C(T^*)$, AND P, BASED ON THE COST FUNCTION $\mathrm{C}_0(T) = 1000 + 5 \times (\exp[1.2 \times \int_{19}^{T} w_\kappa(t)dt] - 1)$

P	Optimal Release Time T*	Total Expected Cost C(T*)	P	Optimal Release Time T*	Total Expected Cost C(T*)
0.01	19.5450	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			

From the examples illustrated in Tables III Tables IV, we can conclude the following facts:

1) When P is relatively small (such as 0.01, 0.02, 0.03, ..., to 0.06), the total expected cost is larger than the expected value of (22): $C1(T^*) = 4719.66$. This is due to the basic cost of adopting new automated techniques or tools.
2) As P increases, the optimal release time T^* increases, but the total expected software cost decreases. This is because we can detect more faults, and reduce the cost of correcting faults during the operational phase.
3) Under the same P value, and with different cost functions, the larger the cost function, the smaller the optimal release time.

The above analyses will greatly aid software personnel in choosing the best software economic policy based on cost, testing-effort, and test efficiency.

REFERENCES

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

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

[3] J. D. Musa, *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*: McGraw-Hill, 1987.

[4] S. Yamada, J. Hishitani, and S. Osaki, "Software reliability growth model with Weibull testing effort: a model and application," *IEEE Trans. Rel.*, vol. R-42, pp. 100–105, 1993.

[5] S. Yamada, H. Ohtera, and H. Narihisa, "Software reliability growth models with testing effort," *IEEE Trans. Rel.*, vol. R-35, no. 1, pp. 19–23, Apr. 1986.

[6] S. Yamada and S. Osaki, "Cost-reliability optimal release policies for software systems," *IEEE Trans. Rel.*, vol. 34, no. 5, pp. 422–424, 1985.

[7] C. Y. Huang, M. R. Lyu, and S. Y. Kuo, "A unified scheme of some nonhomogenous Poisson process models for software reliability estimation," *IEEE Trans. on Software Engineering*, vol. 29, no. 3, pp. 261–269, Mar. 2003.

[8] C. Y. Huang and S. Y. Kuo, "Analysis and assessment of incorporating logistic testing effort function into software reliability modeling," *IEEE Trans. Rel.*, vol. 51, no. 3, pp. 261–270, Sep. 2002.

[9] S. Y. Kuo, C. Y. Huang, and M. R. Lyu, "Framework for modeling software reliability, using various testing-efforts and fault-detection rates," *IEEE Trans. Rel.*, vol. 50, no. 3, pp. 310–320, Sep. 2001.

[10] C. Y. Huang, S. Y. Kuo, and M. R. Lyu, "Optimal software release policy based on cost, reliability and testing efficiency," in *Proceedings of the 23rd Annual International Computer Software and Applications Conference (COMPSAC'99)*, Phoenix, Arizona, U.S.A., Oct. 27–29, 1999, pp. 468–473.

[11] C. Y. Huang, J. H. Lo, S. Y. Kuo, and M. R. Lyu, "Software reliability modeling and cost estimation incorporating testing-effort and efficiency," in *Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE'99)*, Boca Raton, FL, U.S.A, Nov. 1–4, 1999, pp. 62–72.

[12] 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, Sep. 1988.

[13] 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.

[14] ——, "Optimal release policy for hyper-geometric distribution software reliability growth model," *IEEE Trans. Rel.*, vol. 45, no. 4, pp. 646–651, Dec. 1996.

[15] Y. W. Leung, "Optimum software release time with a given budget," *J. Syst. Softw.*, vol. 17, pp. 233–242, 1992.

[16] S. Yamada, H. Narihisa, and S. Osaki, "Optimum release policies for a software system with a scheduled delivery time," *Int. J. of Syst. Sci.*, vol. 15, pp. 905–914, 1984.

[17] H. Pham and X. Zhang, "NHPP software reliability and cost models with testing coverage," *Eur. J. Oper. Res.*, vol. 145, no. 2, pp. 443–454, Mar. 2003.

[18] F. Huq, "Testing in the software development life-cycle: now or later," *Int. J. Project Management*, vol. 18, pp. 243–250, 2000.

[19] R. S. Pressman, *Software Engineering: A Practitioner's Approach*: McGraw-Hill, 2001.

[20] F. N. Parr, "An alternative to the Rayleigh curve for software development effort," *IEEE Trans. on Software Engineering*, vol. SE-6, pp. 291–296, 1980.

[21] T. DeMarco, *Controlling Software Projects: Management, Measurement and Estimation*: Prentice-Hall, 1982.

[22] M. Ohba, "Software reliability analysis models," *IBM Journal of Research and Development*, vol. 28, no. 4, pp. 428–443, 1984.

[23] P. K. Kapur and S. Younes, "Software reliability growth model with error dependency," *Microelectronics and Reliability*, vol. 35, no. 2, pp. 273–278, 1995.

[24] M. R. Lyu and A. Nikora, "Applying software reliability models more effectively," *IEEE Software*, pp. 43–52, Jul. 1992.

[25] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. on Software Engineering*, vol. 23, pp. 736–743, 1997.

[26] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *IEEE Trans. on Software Engineering*, vol. 21, no. 2, pp. 126–136, 1995.

[27] K. Pillai and V. S. Sukumaran Nair, "A model for software development effort and cost estimation," *IEEE Trans. on Software Engineering*, vol. 23, no. 8, pp. 485–497, Aug. 1997.

[28] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *IEEE Trans. on Software Engineering*, vol. 21, no. 2, pp. 126–136, 1995.

[29] K. Okumoto and A. L. Goel, "Optimum release time for software systems based on reliability and cost criteria," *J. System Software*, vol. 1, pp. 315–318, 1980.

[30] R. H. Hou, 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.

[31] ——, "Optimal release policy for hyper-geometric distribution software reliability growth model," *IEEE Trans. Rel.*, vol. 45, no. 4, pp. 646–651, Dec. 1996.

[32] M. Xie and B. Yang, "A study of the effect of imperfect debugging on software development cost," *IEEE Trans. on Software Engineering*, vol. 29, no. 5, pp. 471–473, May 2003.

[33] T. Dohi, "Software release games," *J. Optim. Theory Appl.*, vol. 105, no. 2, pp. 325–346, May 2000.

[34] 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, Sep. 1988.

[35] S. Zheng, "Dynamic release policies for software systems with a reliability constraint," *IIE Transactions*, vol. 34, no. 3, pp. 253–262, Mar. 2002.

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

[37] B. Boehm, *Software Engineering Economics*: Prentice-Hall, 1981.

[38] B. Boehm, C. Abts, and S. Chulani, "Software development cost estimation approaches—a survey," *Annals of Software Engineering*, vol. 10, no. 1–4, pp. 177–205, 2000.

[39] J. Fajardo. The Drawbacks of Developing Your Own Test Tools. [Online] http://www.stickyminds.com/sitewide.asp?ObjectId=6774&Function=DETAILBROWSE&ObjectType=ART#authorbio

Chin-Yu Huang is currently an Assistant Professor in the Department of Computer Science at National Tsing Hua University, Hsinchu, Taiwan. He received the MS (1994), and the Ph.D. (2000) in Electrical Engineering from National Taiwan University, Taipei. He was with the Bank of Taiwan from 1994 to 1999, and was a senior software engineer at Taiwan Semiconductor Manufacturing Company from 1999 to 2000. Before joining NTHU in 2003, he was a division chief of the Central Bank of China, Taipei. His research interests are software reliability engineering, software testing, software metrics, software testability, fault tree analysis, and system safety assessment, etc. He is a member of IEEE.

Michael R. Lyu received the B.S. (1981) in electrical engineering from the National Taiwan University; the M.S. (1985) in computer engineering from the University of California, Santa Barbara; and the Ph.D. (1988) in computer science from the University of California, Los Angeles. He is a Professor in the Computer Science and Engineering Department of the Chinese University of Hong Kong. He worked at the Jet Propulsion Laboratory, Bellcore, and Bell Labs; and taught at the University of Iowa. His research interests include software reliability engineering, software fault tolerance, distributed systems, image & video processing, multimedia technologies, and mobile networks. He has published over 200 papers in these areas. He has participated in more than 30 industrial projects, and helped to develop many commercial systems & software tools. Professor Lyu was frequently invited as a keynote or tutorial speaker to conferences & workshops in U.S., Europe, and Asia. He initiated the International Symposium on Software Reliability Engineering (ISSRE), and was Program Chair for ISSRE 1996, Program Co-Chair for WWW10 & SRDS 2005, and General Chair for ISSRE 2001 & PRDC 2005. He also received Best Paper Awards in ISSRE 98, and in ISSRE 2003. He is the editor-in-chief for two book volumes: Software Fault Tolerance (Wiley, 1995), and the Handbook of Software Reliability Engineering (IEEE & McGraw-Hill, 1996). He has been an Associate Editor of IEEE TRANSACTIONS ON RELIABILITY, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, and Journal of Information Science and Engineering. Professor Lyu is an IEEE Fellow.