Analysis of Incorporating Logistic Testing-Effort Function Into Software Reliability Modeling

Chin-Yu Huang and Sy-Yen Kuo, *Fellow, IEEE*

***Abstract*—This paper investigates a SRGM (software reliability growth model) based on the NHPP (nonhomogeneous Poisson process) which incorporates a logistic testing-effort function. SRGM proposed in the literature consider the amount of testing-effort spent on software testing which can be depicted as an exponential curve, a Rayleigh curve, or a Weibull curve. However, it might not be appropriate to represent the consumption curve for testing-effort by one of those curves in some software development environments. Therefore, this paper shows that a logistic testing-effort function can be expressed as a software-development/test-effort curve and that it gives a good predictive capability based on real failure-data. Parameters are estimated, and experiments performed on actual test/debug data sets. Results from applications to a real data set are analyzed and compared with other existing models to show that the proposed model predicts better. In addition, an optimal software release policy for this model, based on cost-reliability criteria, is proposed.**

***Index Terms*—Mean value function, nonhomogeneous Poisson process, optimal software release policy, software reliability growth model, testing-effort function.**

ACRONYMS[1]

AE	accuracy of estimation
BMMRE	balanced mean magnitude of RE
HGDM	hyper-geometric distribution model
HPP	homogeneous Poisson process
LOC	lines of source code
LSE	least squares estimation
MLE	maximum likelihood estimation
MRE	magnitude of RE
MSF	mean of square fitting faults
MVF	mean value function
NHPP	non-HPP
PE	prediction error
RE	relative error
RMS	root mean square
s-	implies: statistical(ly)
SD	software development
SRE	software reliability engineering
SRGM	software reliability growth model
TEF	testing-effort function

Manuscript received December 1, 1997; revised September 10, 2000. This work was supported by the National Science Council, TAIWAN, R.O.C., under Grant NSC 86-2213-E259-002. Responsible Editor: M. A. Vouk.

The authors are with the Electrical Engineering Department, National Taiwan University, Taipei, Taiwan (e-mail: CY2Huang@mail-cbc.gov.tw; SYKuo@cc.ee.ntu.edu.tw).

Publisher Item Identifier 10.1109/TR.2002.801847.

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

NOTATION

$m(t)$	mean number of faults detected in time $(0, t]$, an MVF
$\lambda(t)$	$dm(t)/dt$: failure intensity for $m(t)$
$w(t)$	current testing-effort consumption at time t
$W(t)$	cumulative $w(t)$
a	mean number of initial faults
r	fault detection rate per unit testing-effort
N	total testing-effort eventually consumed
α	consumption rate of testing-effort expenditures in the logistic TEF
A	constant parameter in the logistic TEF
β	scale parameter in the Weibull-type TEF
m	shape parameter in the Weibull-type TEF
$R(x\|t)$	conditional software reliability
L	likelihood function
W_k	cumulative testing-effort actually consumed in $(0, t_k]$
m_k	cumulative number of faults observed in $(0, t_k]$
T_{LC}	software life-cycle length
C_1	cost of correcting an error during testing
C_2	cost of correcting an error during operation, $C_2 > C_1$
C_3	cost of testing per unit testing-effort expenditures

I. INTRODUCTION

SOFTWARE reliability is the probability that a given software functions correctly under a given environment during a specified period of time. It is a key software-quality factor. Software reliability represents a customer-oriented view of software quality. It relates to practical operation rather than simply the design of a program. Therefore, it is dynamic rather than static. The aim and objective of software (reliability) engineers are to increase the probability that a designed program works as intended by the customers. Hence, measuring and computing the reliability of software systems are very important. They can be used for planning and controlling all testing resources during development, and can assure us about the correctness of software. A common approach for measuring software reliability is by using an analytic model whose parameters are generally estimated from available data on software failures. However, research activities in SRE have been conducted over the past 2 decades extensively, and many SRGM have been proposed [1]. SRGM are successful for estimating software reliability and the number of faults remaining in the software systems. They can be used to evaluate SD status and SRE technology quantitatively [2], [3].

The testing phase is an important and expensive part of SD. Many research studies assume that the consumption rate of testing resources during the testing phase is constant, or do not even consider such testing-effort. References [2]–[4] show that the effort index (execution time) is a better exposure indicator for software reliability modeling than calendar time because the shape of the observed reliability growth curve depends strongly on the time distribution of the testing-effort. References [2], [3], [5], [6] propose a SRGM which describes the relationship among the calendar testing, the amount of testing-effort, and the number of software faults detected by testing. The testing-effort can be represented as the number of CPU hours, the number of executed test cases, etc. Sometimes the testing time can be represented by the number of tests instead of the execution time [4]. In the area of software reliability modeling, SD effort was often described by the traditional exponential, Rayleigh, or Weibull curves [5]–[7]. However, in many software testing environments it is difficult to describe the testing-effort function by these 3 effort consumption curves only. In this paper, we show that a logistic TEF can be expressed as a SD/test-effort curve. Experiments have been performed based on real test/debug data sets. Comparisons of predictive capabilities between various models are presented. The results show that the SRGM with a logistic TEF can estimate the number of initial faults better than the previous approaches.

Section II briefly describes existing TEF in the literature, and discusses the proposed logistic TEF. Section III investigates the SRGM with logistic TEF. The parameters of an SRGM are estimated with logistic TEF by using the LSE and MLE. Section IV applies this model to actual software failure data, and shows the numerical results. Section V is concerned with the applications of this model to an optimum release policy based on the cost-reliability criterion.

II. Testing-Effort Functions

This section briefly reviews some TEF which were developed to estimate the SD effort. Most of them are parametric because they predict development effort using a formula of fixed form that is parameterized from historical data records. During the software testing phase, many testing-efforts, such as the man power, the number of executed test cases, and the CPU time, are consumed.

A. A Brief Review of TEF

1) COCOMO[2] Effort Algorithms: COCOMO is one of the best known SD models and was developed by Boehm based on a regression analysis of 63 completed projects. COMOCO relates the effort to Delivered Source Instructions [8]. COCOMO provides a combination of various functional forms made accessible to the user in a structured manner. The COCOMO effort algorithms all have the basic form:

$$E = a \cdot S^b; \tag{1}$$

$E \equiv$ the effort,

$S \equiv$ the size, typically measured as LOC,

$a \equiv$ the productivity parameter,

$b \equiv$ the scale parameter.

[2]COnstructive COst MOdel.

2) Analogies: Reference [9] proposes an alternative approach to estimation, based on the use of analogies. The underlying principle is to characterize projects in terms of features. However, estimation of software-project effort by analogy has an advantage in that it is very intuitive.

3) Machine Learning Approaches: Reference [10] proposes machine learning approaches to estimating SD effort using an algorithm for building regression trees, and neural-network learning approach known as back-propagation. The advantage of learning approaches is that they are adaptable and nonparametric.

4) Norden/Rayleigh Model: References [11], [12] observe that the Rayleigh distribution provides a good approximation of the manpower curve for various hardware development processes. Then, the Rayleigh distribution is used as an approximation to the smoothed labor distribution curve and is applied to several software projects

$$P = K \cdot (1 - \exp(=a \cdot t^2)); \tag{2}$$

$K \equiv$ the total consumed manpower,

$a \equiv \dfrac{1}{2t_d^2}$,

$t_d \equiv$ the time for manpower to peak.

5) Chatterjee TEF: Reference [13] proposes a TEF to describe the resource consumption during SD. It considered that the test-effort and learning-factor depend on each other; i.e., it assumes that test-effort and learning-factor are inversely proportional to each other, and that they have a joint effect on SD. Therefore, the Chatterjee TEF can be written as:

$$w(t) = \frac{k}{[f(t)]^p} \tag{3}$$

$k \equiv$ the proportionality constant,

$f(t) \equiv$ the learning factor;

an increasing function of t.

6) Pillai and Nair Gamma Model: Reference [14] proposes a gamma model for SD-effort and cost-estimation. This model is based on the Gamma distribution and can be represented as

$$w(t) = \frac{4N}{t_d^3} \cdot t^2 \cdot \exp\left(-\frac{2t}{t_d}\right) \tag{4}$$

$t_d \equiv$ the time for SD effort to peak,

$N \equiv$ the total SD effort expended for the project.

7) Thoma Test-Instance Functions: HGDM was first proposed by Thoma and has been developed to estimate the number of remaining software faults after the test/debug phase [15], [16]. The collection of test operations performed in a day or a week is called a test instance. Therefore, various functions were proposed to describe the test instance, such as:

$$w(i) = \text{tester}(i) \cdot (a \cdot i + b), \tag{5}$$

$$w(i) = \text{tester}(i) \cdot (a \cdot i^2 + b \cdot i + c), \tag{6}$$

$$w(i) = [\text{tester}(i)]^p \cdot a, \tag{7}$$

$\text{tester}(i) \equiv$ the number of workers involved in $t(i)$

8) Hou and Kuo Learning-Factor Functions: References [17]–[19] propose two new functions based on the exponential,

TABLE I
SUMMARY OF REAL DATA SETS STUDIED

Data Set [Reference]	No. of Faults Detected	Observation Period	Software Project Program Descriptions & Characteristics
DS1 [2]	136	21 weeks	Real-time Command and Control Application (System T1), 9 Programmers, Software Size: 21,700 LOC, Execution Time: 25.3 CPU hours.
DS2 [4]	328	19 weeks	PL/I application Program, Software Size: 1,317KLOC, Execution Time: 47.65 CPU hours.
DS3 [16]	86	22 days	The data set is the pattern of discovery of faults by Thoma. Execution Time: 93 CPU hours
DS4 [25]	227	38 weeks	Real-time command and control application software that supported Space Shuttle flights STS2, STS3, STS4 at the Johnson Space Center. Execution Time: 2456.9 CPU hours.

and on the S-shaped learning curve, to enhance the HGDM and make it more realistic.

Exponential learning curve:

$$w(i) = p_{\mathrm{LT}} \cdot [1 - \exp(-u_i \cdot a \cdot i)], \quad a > 0, \qquad 0 < p_{\mathrm{LT}} \le 1. \tag{8}$$

Logistic learning curve:

$$w(i) = \frac{p_{\mathrm{LT}}}{1 + b \cdot \exp(-u_i \cdot a \cdot i))}, \quad a > 0, \quad b > 0, \quad 0 < p_{\mathrm{LT}} \le 1; \tag{9}$$

$p_{\mathrm{LT}} \equiv$ the limit value of learning factor.

9) Yamada Weibull-Type TEF: According to [2], [3], [5], [6], [11], testing-effort should not be assumed constant throughout the testing phase. Instantaneous testing-effort ultimately decreases during the testing life-cycle because the cumulative testing-effort approaches a finite limit. This assumption is reasonable because no software company will spend infinite resources on software testing. Hence, [5], [6] show that the testing-effort can be described by a Weibull-type distribution and have the following 3 cases.

1) Exponential curve: The cumulative testing-effort consumed in $(0, t]$ is

$$W(t) = N \cdot [1 - \exp(-\beta \cdot t)]. \tag{10}$$

2) Rayleigh curve: The cumulative testing-effort consumed is

$$W(t) = N \cdot \left[1 - \exp\left(-\frac{\beta}{2} \cdot t^2\right)\right]. \tag{11}$$

3) Weibull curve: The cumulative testing-effort consumed is

$$W(t) = N \cdot [1 - \exp(-\beta \cdot t^m)]. \tag{12}$$

For the Weibull-type curves (12), when $m = 1$ or $m = 2$, the result is the exponential or Rayleigh curve respectively; therefore, they are special cases of the Weibull TEF. For the Weibull-type curves, if $m = 3, 4$, or 5, these testing-effort curves have an apparent peak phenomenon (nonsmoothly increasing and degrading consumption curve) during the SD process; i.e., a peak work-rate occurs. This phenomenon seems not so realistic because it is not commonly used to interpret the actual SD/test process [20]. Hence, the Weibull function might not be suitable for modeling the testing-effort consumption curve, although it can be made to fit or approximate many distributions and represents flexible testing-effort by controlling the shape parameter.

B. Logistic TEF

Because actual testing-effort data represent various expenditure patterns, sometimes the testing-effort expenditures are difficult to describe only by an exponential or a Rayleigh curve. Although a Weibull-type curve can fit the data well under the general SD environment and is widely used in software reliability modeling, it has the apparent peak phenomenon when $m > 3$ [20]–[22]. An alternative is the logistic TEF, first presented in [23]. This function was fairly accurate as reported by the Yourdon 1978–1980 project survey [24]. The cumulative testing-effort consumption in time $(0, t]$ is

$$W(t) = \frac{N}{1 + A \cdot \exp(-\alpha \cdot t)}; \tag{13}$$

the current testing-effort consumption is

$$w(t) = \frac{dW(t)}{dt} = \frac{N \cdot A \cdot \alpha \cdot \exp(-\alpha \cdot t)}{[1 + A \cdot \exp(-\alpha \cdot t)]^2} = \frac{N \cdot A \cdot \alpha}{\left[\exp\left(\frac{\alpha \cdot t}{2}\right) + A \cdot \exp\left(-\frac{\alpha \cdot t}{2}\right)\right]^2}. \tag{14}$$

Therefore, $w(t)$ is a smooth bell-shaped function, and reaches its maximum value at

$$t_{\max} = \frac{1}{\alpha} \cdot \log(A). \tag{15}$$

In contrast with the Weibull-type TEF in the initial point, the logistic TEF $W(0) \neq 0$. The discrepancies between the Weibull-type curve and the $W(t)$ exist in the earlier stages of SD where progress is often least visible, and where formal accounting procedures for recording the amount of applied testing-effort might not have been instituted. It is possible to judge between these models using some statistical test of their relative ability to fit actual failure data, such as adjusting the origin and scales linearly [23].

C. Comparisons Between Different TEF

To check the performance of the logistic TEF and to compare fairly with other TEF, especially the Rayleigh distribution, this paper applies 4 real data-sets to these proposed models. These

TABLE II
COMPARISON RESULTS FOR DIFFERENT TEF BASED ON DS1

Parameter	Distribution		
	Logistic	Rayleigh	Exponential
Bias	0.0548	-1.1496	-16.5313
Variation	0.3508	3.4579	6.3495
RMS-PE	0.3551	3.6440	17.7087
MRE	-0.0040	0.2384	-0.5254
$PE_{\text{end of testing}}$	-0.1022	6.0332	-13.2936
BMMRE	24.3718	65.2980	81.4501

TABLE III
COMPARISON RESULTS FOR DIFFERENT TEF BASED ON DS2

Parameter	Distribution			
	Logistic	Exponential	Rayleigh	Weibull
Bias	-0.0317	-0.3938	0.8302	0.0342
Variation	1.1069	1.3685	2.1690	0.9559
RMS-PE	1.9105	1.4241	2.0227	0.9565
MRE	0.0022	0.0266	0.0525	-0.0079
$PE_{\text{end of testing}}$	1.0510	1.2688	2.5038	-0.3775
BMMRE	6.6707	7.1467	63.7387	7.0054

TABLE IV
COMPARISON RESULTS FOR DIFFERENT TEF BASED ON DS3

Parameter	Distribution		
	Logistic	Rayleigh	Weibull
Bias	-0.1221	0.3242	0.1824
Variation	2.2754	2.3824	2.3666
RMS-PE	2.2853	2.4044	2.3737
MRE	0.0209	0.0171	-0.4352
$PE_{\text{end of testing}}$	0.1948	0.1515	-0.4352
BMMRE	15.5306	18.4428	13.7032

data sets are in Table I, and the comparison criteria for evaluation are described here [9], [10], [14]:

1) Prediction Error (PE_i)
$$= \text{Actual(Observed)}_i - \text{Predicted(Estimated)}_i$$

2) $$\text{Variation} = \sqrt{\frac{\sum_{i=1}^{n} (\text{PE}_i - \text{Bias})^2}{n-1}}$$

3) $$\text{Bias} = \frac{1}{n} \cdot \sum_{i=1}^{n} \text{PE}_i$$

4) $$\text{RMS-PE} = \sqrt{\text{Bias}^2 - \text{Variation}^2}$$

5) $$\text{MRE} = \left| \frac{M_{\text{estimated}} - M_{\text{actual}}}{M_{\text{actual}}} \right|$$

6) $$\text{BMMRE} = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{|M_{\text{estimated}} - M_{\text{actual}}|}{\min(M_{\text{estimated}}, M_{\text{actual}})}.$$

The comparisons between the proposed logistic TEF and the Weibull-type TEF are illustrated in Tables II–V and Figs. 1–4. Figs. 1–4 illustrate the comparisons between the observed failure data and the data estimated by various TEF. The computed RMS-PE, the bias, the Variation, the MRE, and the BMMRE based on different data sets are shown in Tables II–V. Figs. 1–4 and Tables II–V show that: 1) the logistic TEF yields a better fit for the data sets chosen, and 2) the structure of logistic TEF is very flexible for various testing environments.

TABLE V
COMPARISON RESULTS FOR DIFFERENT TEF BASED ON DS4

Parameter	Distribution	
	Logistic	Rayleigh
Bias	-7.09	44.7144
Variation	96.4962	166.7640
RMS-PE	96.7563	172.6550
MRE	0.0753	0.05221
$PE_{\text{end of testing}}$	3.4371	-52.4425
BMMRE	25.0964	100.7361

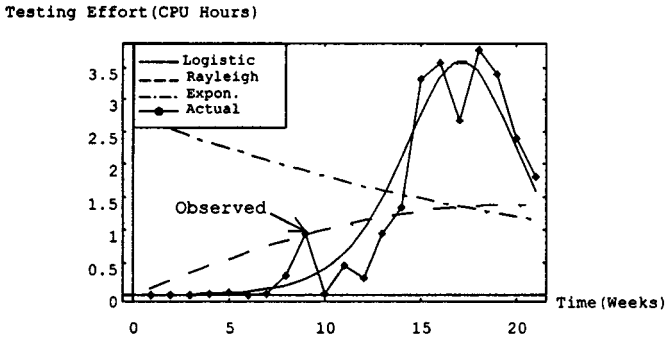

Fig. 1. Observed/estimated TEF for DS1.

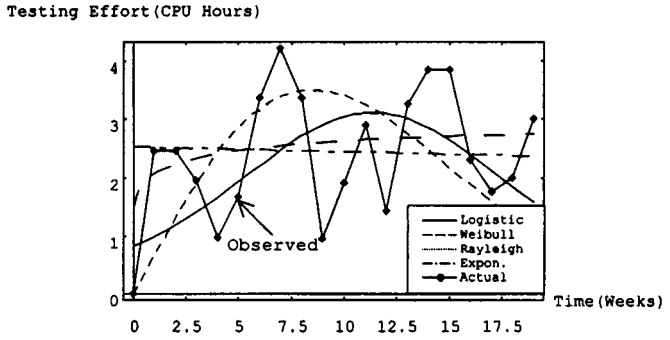

Fig. 2. Observed/estimated TEF for DS2.

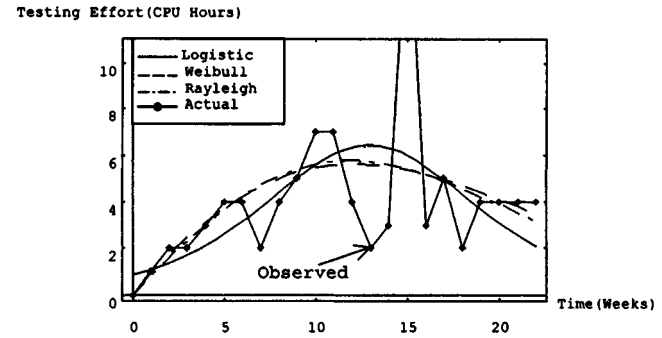

Fig. 3. Observed/estimated TEF for DS3.

III. SOFTWARE RELIABILITY GROWTH MODEL

A basic SRGM is based on the assumptions:

1) The fault removal process follows the NHPP.

2) The software system is subject to failures at random times caused by faults remaining in the system.

3) The mean number of faults detected in $(t, t+\Delta t]$ by $w(t)$ is proportional to the mean number of faults remaining in the system.

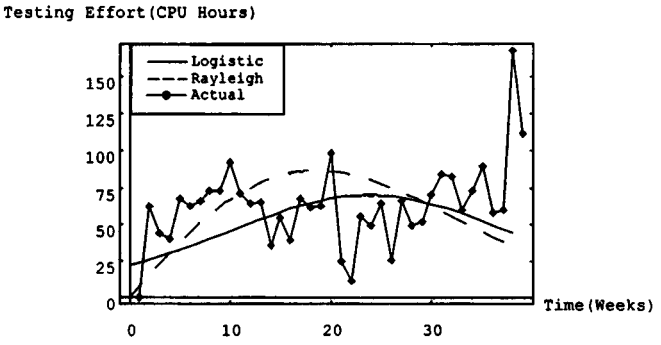

Fig. 4. Observed/estimated TEF for DS4.

4) The proportionality constant does not change with respect to time.

5) The testing-effort consumption with exposure is modeled by a logistic function.

6) Each time a failure occurs, the fault that caused it is immediately and perfectly removed without new faults being introduced.

7) Correction of faults takes negligible time, and a detected error is removed with certainty.

Because the s-expected current detected fault content is finite at any time, $m(t)$ is an increasing function of t; $m(0) = 0$. From these assumptions:

$$\frac{m(t+\Delta t) - m(t)}{w(t)} = r \cdot [a - m(t)] \cdot \Delta t;$$

$$r = \lim_{\Delta t \to 0} \left(\frac{m(t+\Delta t) - m(t)}{[a - m(t)] \cdot w(t) \cdot \Delta t} \right). \quad (16)$$

Consequently, if the number of detected faults due to the current testing-effort expenditures is proportional to the number of remaining faults, then

$$\frac{dm(t)}{dt} \cdot \frac{1}{w(t)} = r \cdot [a - m(t)], \qquad a > 0, \quad 0 < r < 1. \quad (17)$$

Solving (17) under the boundary condition $m(0) = 0$ gives:

$$\begin{aligned} m(t) &= a \cdot (1 - \exp[-r(W(t) - W(0))]) \\ &= a \cdot (1 - \exp[-r \cdot W^*(t)]); \end{aligned} \quad (18)$$

$$W^*(t) \equiv \frac{N}{1 + A \cdot \exp(-\alpha \cdot t)} - \frac{N}{1 + A}.$$

Therefore,

$$\begin{aligned} \lambda(t) &= \frac{dm(t)}{dt} = a \cdot r \cdot w(t) \cdot \exp[-r \cdot W^*(t)] \\ &= a \cdot r \cdot w(t) \exp\left(-\frac{N}{1 + A \cdot \exp(-\alpha \cdot t)} + \frac{N}{1 + A} \right). \end{aligned} \quad (19)$$

$$m(\infty) = a \cdot \left(1 - \exp\left[\frac{N}{1 + A} \right] \right). \quad (20)$$

Thus, the mean number of undetected faults, if a test is applied for an infinite amount of time, is

$$\begin{aligned} a - m(\infty) &= a \cdot \exp\left[-r \cdot \frac{N \cdot A}{1 + A} \right] \\ &\cong a \cdot \exp(-r \cdot N), \qquad \text{if } A \gg 1. \end{aligned} \quad (21)$$

Hence, not all the original faults in a software system can be fully detected with a finite testing-effort because the effort to be eventually used during the testing phase is limited to N.

IV. Evaluation of SRGM With Logistic TEF and Performance Analysis

A. Data Description

This section evaluates the performance of SRGM with logistic TEF. The real data set is the System T1 data of the Rome Air Development Center (RADC) projects in [2], and the failure data are generally of the best quality. System T1 is used for a real-time command and control application. The size of software is about 21 700 object instructions. It took 21 weeks and 9 programmers to complete the test. During the test phase, about 25.3 CPU hours were consumed and 136 software faults were removed.

B. Criteria for Model Comparison

The 4 performance-comparison criteria are given here.

1) AE is defined as [26]:

$$\left| \frac{M_a - a}{M_a} \right| \quad (22)$$

M_a is the actual cumulative number of detected faults after the test, and a is the estimated number of initial faults. For practical purposes, M_a is obtained from software fault tracking after software testing.

2) RE is defined as [2]:

$$\frac{m(t_q) - q}{q}. \quad (23)$$

If q failures are observed in test-time t_q, use the failure data up to time t_e ($t_e < t_q$) to estimate the parameters of $m(t)$: $m(t_q)$. The estimate is compared with the actual number q. The procedure is repeated for various t_e. The predictive validity is checked by plotting RE versus t_q.

3) Noise is defined as [27]:

$$\sum_{i=1}^{n} \left| \frac{r_i - r_{i-1}}{r_{i-1}} \right| \quad (24)$$

$r_i \equiv$ predicted failure rate.

Small values represent less noise in the model's prediction behavior, indicating more smoothness. A noise measure of ∞ indicates that the model has predicted a zero failure rate [27].

4) MSF (for long-term prediction) is defined as [27]

$$\frac{1}{k} \cdot \sum_{i=1}^{k} [m(t_i) - m_i]^2. \quad (25)$$

MSF is used for quantitative comparisons for long-term predictions, because it provides a better-understood measure of the differences between actual and predicted values. A smaller MSF indicates a smaller fitting error and better performance [27].

C. Performance Analysis

This section evaluates the proposed model and several existing NHPP models. First, parameters of all selected models are estimated and the related mean value functions are obtained. Second, all the selected models are compared with each other based on objective criteria.

N, A, α in logistic TEF are estimated using LSE. Using the estimated TEF, the other parameters a, r in (18) can be solved numerically by MLE. Therefore, $a = 138.165$, $r = 0.145\,098$. Table VI compares the performance of various SRGM for the data set investigated in this paper. Due to the limitations of paper

TABLE VI
COMPARISON RESULTS BETWEEN DIFFERENT SRGM

Model	a	r	AE (%)	MSF	Noise	RE (%)	KD
Proposed	138.026	0.14509	26.58	62.41	2.11638	0.1958	0.0644
Yamada Rayleigh-type	140.782	0.141568	25.11	89.24	2.80449	0.2529	0.1161
Exponential	137.2	0.1560	27.12	3019.66	3.12547	0.303	0.0653
Inflection S-Shaped	159.11	0.0765	15.36	118.3	3.5489	16.99	0.1844
Delayed S-Shaped	237.196	0.09634	26.16	245.25	2.33098	0.2420	0.1729
Kapur Error Dependency	209.12	0.303	11.17	13.6	2.91837	0.2743	0.1677

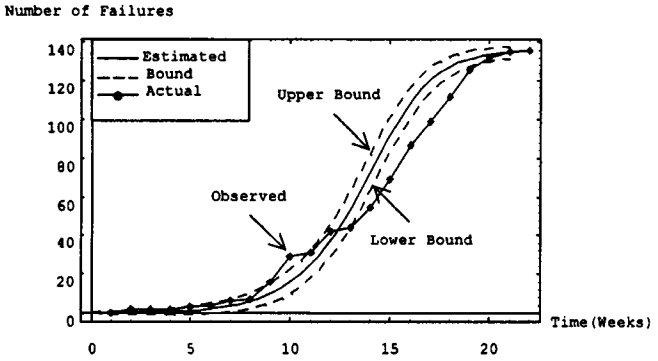

Fig. 5. $m(t)$ of the proposed model and the 90% s-confidence bounds.

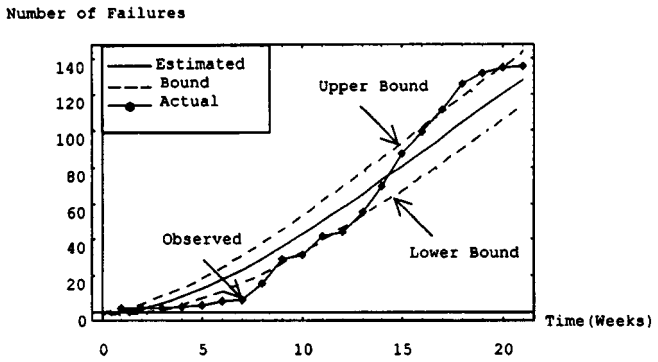

Fig. 6. $m(t)$ of the Delayed S-Shaped model and the 90% s-confidence bounds.

size, only 3 pre-eminent models are used for detailed discussions:

- proposed model in this paper,
- Delayed S-Shaped Model,
- Yamada Rayleigh-type model.

These have better performance as shown in Table VI. Figs. 5–7 show the actual (observed), fitted software failures, and the 90% s-confidence bounds [3] respectively. A good SRGM should be able to predict well the behavior of future failures. In general this can be evaluated by considering the estimated probability distribution of the next failure-time. Thus Fig. 8 shows a u-plot analysis of predictions from the selected models on the Musa system T1 data set. The u-plot of the proposed model is close to the line of unit slope; and the proposed model has a smaller Kolmogorov-Distance, which is defined as the maximum vertical derivation between the plot and the line of unit slope, when comparing with other existing SRGM, as shown in Table VI. Table VII shows the values of $\mathrm{Var}[a]$ and $\mathrm{Var}[r]$ for 3 selected models [28]. Estimates for system T1 between selected models are compared in Table VIII. The RE in prediction is cal-

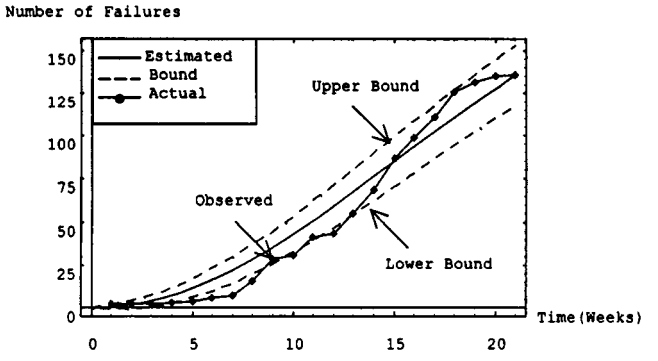

Fig. 7. $m(t)$ of the Yamada Rayleigh-type model and the 90% s-confidence bounds.

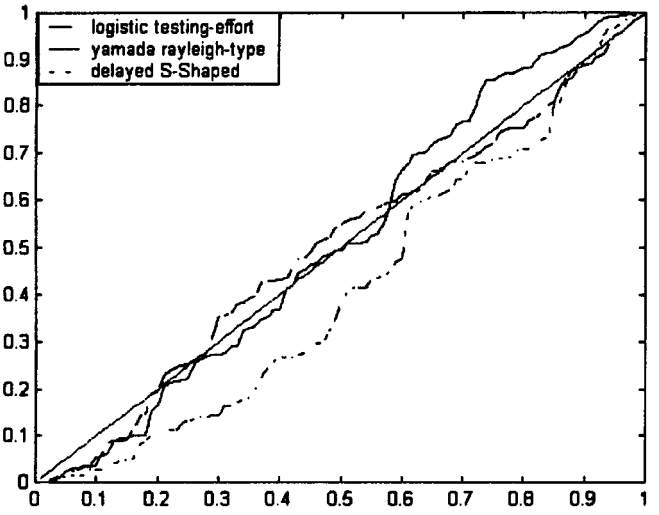

Fig. 8. u-plots for predictions of T_i, $i = 36, \ldots, 136$ for Musa system T1 data-set.

TABLE VII
$\mathrm{Var}[a]$ AND $\mathrm{Var}[r]$ FOR SELECTED MODELS

Model	a	r	Var[a]	Var[r]
Proposed	138.026	0.14509	11.989	0.00024695
Delayed S-Shaped	237.196	0.09634	40.188	0.00032609
Yamada Rayleigh-type	140.782	0.141568	12.173	0.00024752

TABLE VIII
ESTIMATES FOR SYSTEM T1 FROM SELECTED MODELS

	Proposed		Delayed S-Shaped		Yamada Rayleigh-type	
	Lower	Upper	Lower	Upper	Lower	Upper
a	132.112	143.939	226.368	248.023	134.822	146.741
r	0.11824	0.17193	0.065497	0.12718	0.11469	0.16843

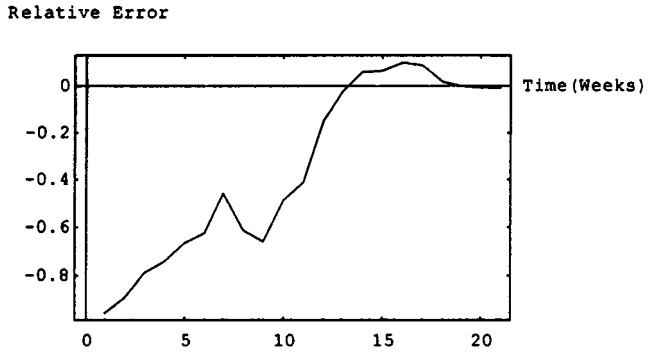

Fig. 9. RE curve for the Delayed S-shaped model, based on Musa's system T1 data set.

Fig. 10. RE curve for the Yamada Rayleigh-type model based on Musa's system T1 data set.

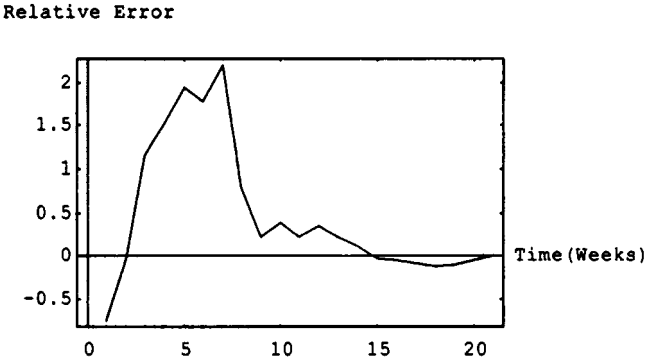

Fig. 11. RE curve for the proposed model, based on Musa's system T1 data set.

culated for this data set at the end of testing. The results are in Figs. 9–11. From these simulation/comparison results, the proposed model performs appreciably better than the others. This model fits the observed data better, and predicts the future behavior well for this data set.

V. Optimal Software-Release Policy

A. Software Release-Time Based on Reliability Criteria

In general, the software-release time problem is associated with the reliability of a software system. The release policy based on the reliability criterion is discussed first. If the reliability of a software system is known to have reached an acceptable level, then we the right time to release this software can be determined. References [29]–[31] first discussed the release problem by considering the software cost-benefit. The conditional reliability function after the last failure occurs at time t is:

$$\begin{aligned} R &= R(\Delta t|t) = \exp[-(m(t+\Delta t) - m(t))] \\ &= \exp[-a \cdot (\exp[-r \cdot W^*(t)] - \exp[-r \cdot W^*(t+\Delta t)])]. \end{aligned} \tag{26}$$

Differentiate $R(\Delta t|t)$ with respect to t, then $\partial R(\Delta t|t)/\partial t \geq 0$. Hence $R(\Delta t|t)$ is a monotonic increasing function of t. Take the logarithm of (26):

$$\log[R(t)] = -[m(t+\Delta t) - m(t)]. \tag{27}$$

Solving (27) and (18) determines the testing time needed to reach a desired R. $R(t)$ is increasing in t $(0 \leq t < T_{\mathrm{LC}})$. Using (27), one can get the required testing-time needed to reach the reliability objective R or decide whether R is reached or not in a specified time interval.

B. Optimal Release-Time Based on Cost-Reliability Criterion

This section discusses the cost model and release policy based on the cost-reliability criterion. Using the total software cost evaluated by the cost criterion, the cost of testing-effort expenditures during the software testing/development phase and the cost of correcting errors before and after release are [5], [6], [20]:

$$\begin{aligned} C_1(T) = C_1 \cdot m(T) + C_2 \cdot [m(T_{\mathrm{LC}}) - m(T)] \\ + C_3 \cdot \int_0^T w(x)\, dx. \end{aligned} \tag{28}$$

From [8], $C_2 > C_1$ because C_2 is usually an order of magnitude greater than C_1. Differentiate (28) with respect to T and set it to 0:

$$\begin{aligned} \frac{d}{dT} C(T) &= C_1 \cdot \frac{dm(T)}{dT} - C_2 \cdot \frac{dm(T)}{dT} + C_3 \cdot w(T) = 0 \\ &= w(T) \cdot (-(C_2 - C_1) \cdot a \cdot r \\ &\qquad \cdot \exp[-r \cdot W^*(T)] + C_3); \end{aligned} \tag{29}$$

$$\begin{aligned} \frac{1}{w(T)} \cdot \frac{dm(T)}{dT} &= \frac{\lambda(T)}{w(T)} = \frac{C_3}{C_2 - C_1} \\ &= \frac{a \cdot r \cdot w(T) \cdot \exp(-r \cdot W^*(T))}{w(T)} \\ &= a \cdot r \cdot \exp(-r \cdot W^*(T)) \\ &= r \cdot (a - m(T)). \end{aligned} \tag{30}$$

Case 1) $T = 0$; then $m(0) = 0$, and

$$\frac{1}{w(T)} \cdot \frac{dm(T)}{dT} = \frac{\lambda(T)}{w(T)} = a \cdot r.$$

Case 2) If $T \to \infty$; then $w(\infty) = N$, and

$$\begin{aligned} m(\infty) &= a \cdot \left[1 - \exp\left(-r \cdot \frac{N \cdot A}{1+A}\right)\right]; \\ \frac{\lambda(T)}{w(T)} &= a \cdot r \cdot \exp\left(-r \cdot \frac{N \cdot A}{1+A}\right). \end{aligned}$$

Therefore, $\lambda(T)/w(T)$ is monotonically decreasing in T. If

$$\frac{\lambda(0)}{w(0)} = a \cdot r \leq \frac{C_3}{C_2 - C_1},$$

$$\text{then } \frac{\lambda(T)}{w(T)} \leq \frac{C_3}{C_2 - C_1} \quad \text{for } 0 < T < T_{\mathrm{LC}}.$$

Hence, for this case, the optimal software release time $T^* = 0$. If

$$\frac{\lambda(0)}{w(0)} = a{\cdot}r > \frac{C_3}{C_2 - C_1} > \frac{\lambda(T)}{w(T)} = a{\cdot}r{\cdot}\exp\left(-r \cdot \frac{N \cdot A}{1+A}\right),$$

then there exists a finite and unique solution, T_0 satisfying

$$\frac{\lambda(T)}{w(T)} = \frac{C_3}{C_2 - C_1} = r \cdot (a - m(T))$$
$$= a \cdot r \cdot \exp\left[-r \cdot \left(\frac{N}{1 + A \cdot \exp(-\alpha \cdot T)} - \frac{N}{1+A}\right)\right].$$

Rearranging this equation, gives:

$$T_0 = \frac{1}{a} \cdot \log\left(\frac{A \cdot \Theta}{N - \Theta}\right)$$

minimizes $C(T)$

$$\Theta \equiv \frac{1}{r} \cdot \log\left(a \cdot r \cdot \frac{C_2 - C_1}{C_3}\right) + \frac{N}{1+A}. \qquad (31)$$

Because

$$\frac{dC(T)}{dt} < 0, \qquad \text{for } 0 < T < T_0,$$
$$\frac{dC(T)}{dt} > 0, \qquad \text{for } T > T_0,$$

the minimum of $C(T)$ is at $T = T_0$ for $T_0 \leq T$. [Because $d^2C(T)/dT^2 > 0$, then $C(T)$ is a convex function.]

Section V-A provides the information to get the required testing time needed to reach R_0. The goal is to minimize the total software cost to achieve the desired software reliability, and then the optimal software release time is obtained. That is, mathematically minimize $C(T)$ subject to $R(t + \Delta t|t) \geq T_0$, $0 < R_0 < 1$.

T^* = optimal software release time or total testing time = $\max[0, T_0, T_1]$, where T_0 is finite and the unique solution T of (31), T_1 is finite and unique T satisfying $R(t + \Delta t|t) = R_0$, $0 < R_0 < 1$.

This analysis is summarized in the relationships:

Given: $C_1 > 0$, $C_2 > C_1$, $C_3 > 0$, $\Delta t > 0$, $0 < R_0 < 1$:

1) If $\lambda(0)/w(0) > C_3/(C_2 - C_1)$ and $\lambda(T)/w(T) = a \cdot r \cdot \exp(-r \cdot (N \cdot A/(1+A))) \leq C_3/(C_2 - C_1)$, then $T^* = \max[0, T_0, T_1]$ for $R(\Delta t|0) < R_0 < 1$, or $T^* = T_0$ for $0 < R_0 < R(\Delta t|t = 0)$.
2) If $\lambda(0)/w(0) \geq C_3/(C_2 - C_1)$ then $T^* = T_1$ for $R(\Delta t|0) < R_0 < 1$, or $T^* \geq 0$ for $0 < R_0 \leq R(\Delta t|0)$.
3) If $\lambda(0)/w(0) \leq C_3/(C_2 - C_1)$ then $T^* - T_1$ for $R(\Delta t|0) < R_0 < 1$, or $T^* = 0$ for $0 < R_0 \leq R(\Delta t|0)$.

To illustrate items 1)–3), use again the first real data-set in Section II-C for a numerical example on the optimal software release problem in Section V-C.

C. Numerical Example

From the previously estimated parameters:

$$N = 29.1095, \quad A = 4624.89, \quad \alpha = 0.493\,515,$$
$$a = 138.165, \quad r = 0.145\,098.$$

Also assume $C_1 = 1$, $C_2 = 100$, $C_3 = 50$, $T_{\mathrm{LC}} = 100$, $R_0 = 0.95$, $\Delta t = 1$.

Then T_0 is estimated as 20.98, based on minimizing $C(T)$ of (31), and T_1 is estimated as 12.79, based on satisfying the reliability criterion of $R(t + \Delta t|t) = R_0$.

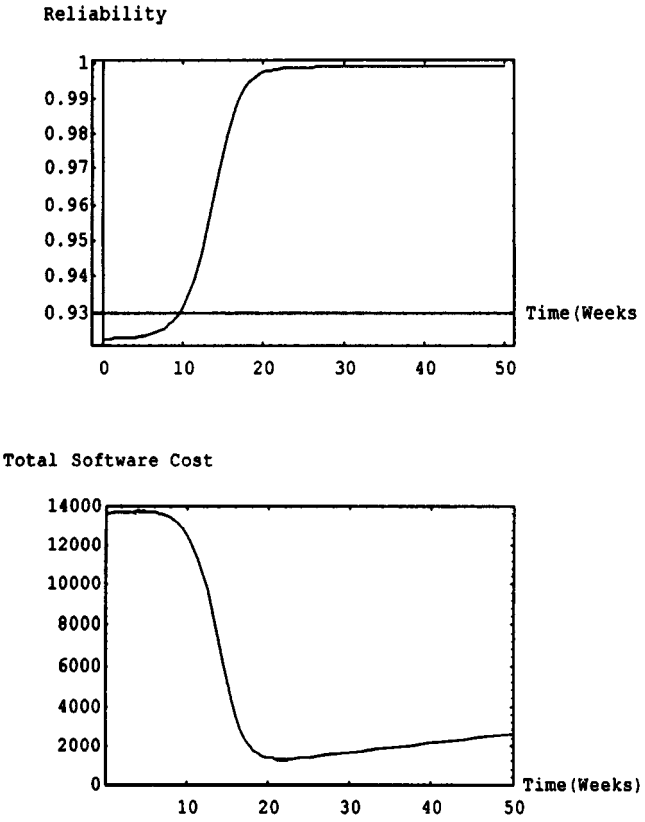

Fig. 12. Total software cost and reliability estimation for the System T1 data-set.

Because

$$\frac{\lambda(0)}{w(0)} > \frac{C_3}{C_2 - C_1}, \quad \frac{\lambda(T)}{w(T)} < \frac{C_3}{C_2 - C_1}, \quad R(\Delta t|0) < R_0. \qquad (32)$$

the T^* is estimated as $\max[0,\ 20.98,\ 12.79] = 20.98$. $C(T*) = 1329.44$ (the plots are shown in Fig. 12).

APPENDIX

To validate the proposed SRGM with a logistic TEF in (18), experiments on 3 real test/debug data sets were performed. Two popular estimation techniques are MLE and LSE [1]–[3]. The MLE estimates parameters by solving a set of simultaneous equations and is better in deriving s-confidence intervals. But the equation sets are very complex and usually must be solved numerically. The LSE minimizes the sum of squares of the deviations between what is actually observed/gotten and what is anticipated. LSE is generally considered to be the best for medium sample sizes and provides the best point estimates [20]. LSE is used here to fit the logistic curve with the real data set. For the method of "least square sum," the evaluation formula S1(N, A, α) is:

$$\text{Minimize S1}(N, A, \alpha) = \sum_{k=1}^{n} [W_k - W(t_k)]^2, \qquad \text{(A-1)}$$

$W_k \equiv$ cumulative testing-effort actually consumed in time $(0, t_k]$, $W(t_k) \equiv$ cumulative testing-effort estimated by the logistic testing function in (13).

Differentiate S1 with respect to N, A, and α, set the partial derivatives to zero, and rearrange these terms, to solve this type of nonlinear least-squares problems

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

The LSE, N, is found by solving (A-1) and (A-2):

$$N = \frac{\sum_{k=1}^{n} 2\left(\frac{W_k}{1+A\cdot\exp(-\alpha\cdot t)}\right)}{\sum_{k=1}^{n} 2\left(\frac{1}{1+A\cdot\exp(-\alpha\cdot t)}\right)^2}. \quad \text{(A-3)}$$

Then,

$$\frac{\partial \mathrm{S1}}{\partial A} = \sum_{k=1}^{n} 2\left(W_k - \frac{N}{1+A\cdot\exp(-\alpha\cdot t)}\right) \cdot \frac{N\cdot\exp(-\alpha\cdot t)}{[1+A\cdot\exp(-\alpha\cdot t)]^2} = 0 \quad \text{(A-4)}$$

$$\frac{\partial \mathrm{S1}}{\partial \alpha} = \sum_{k=1}^{n} 2\left(W_k - \frac{N}{1+A\cdot\exp(-\alpha\cdot t)}\right) \cdot \frac{N\cdot A\cdot t\cdot\exp(-\alpha\cdot t)}{[1+A\cdot\exp(-\alpha\cdot t]^2)} = 0. \quad \text{(A-5)}$$

The other parameters A, α can be obtained by substituting the LSE N into (A-4) and (A-5).

The likelihood function for a and r in the NHPP model with $m(t)$ in (18), is:

$$L \equiv \Pr\{N(t_i) = m_i,\ i = 1,\ \ldots,\ n\} = \prod_{k=1}^{n} \frac{m(t_k - m(t_{k-1})}{m_k - m_{k-1}!}\cdot\exp[-(m(t_k) - m(t_{k-1}))], \quad \text{(A-6)}$$
$$m_0 \equiv 0 \qquad \text{for } t_0 = 0.$$

Take the logarithm of the likelihood function in (A-6),

$$\log(L) = \sum_{k=1}^{n} (m_k - m_{k-1})\cdot\log[m(t_k) - m(t_{k-1})] - \sum_{k=1}^{n}[m(t_k) - m(t_{k-1})] - \sum_{k=1}^{n}\log[m(t_k) - m(t_{k-1})!].$$

From (18):

$$m(t_k) - m(t_k - 1) = a\cdot(\exp[-r\cdot W^*(t_{k-1})] - \exp[-r\cdot W^*(t_k)]).$$

Thus,

$$\log L = \sum_{k=1}^{n} (m_k - m_{k-1})\cdot\log(a) + \sum_{k=1}^{n}(m_k - m_{k-1}) \cdot\log(\exp[-r\cdot W^*(t_{k-1})] - \exp[-r\cdot W^*(t_k)]) - a\cdot(1 - \exp[-r\cdot W^*(t_n)]) - \sum_{k=1}^{n}\log[(m_k - m_{k-1})!]. \quad \text{(A-7)}$$

Consequently, the MLE, a and r are obtained by solving

$$\frac{\partial\log(L)}{\partial a} = \frac{\partial\log(L)}{\partial r} = 0,$$
$$a = \frac{\sum_{k=1}^{n}(m_k - m_{k-1})}{1-\phi_n} = \frac{m_n}{1-\phi_n}; \quad \text{(A-8)}$$

$$a\cdot W*(t_n)\cdot\exp[-r\cdot W^*(t_n)] = \sum_{k=1}^{n}(m_k - m_{k-1})\cdot\frac{-W^*(t_{k-1})\cdot\phi_{k-1}) + W^*(t_k)\cdot\phi_k}{\phi_{k-1} - \phi_k};$$
$$\phi_k \equiv \exp[-r\cdot W^*(t_k)] \quad \text{(A-9)}$$

The a and r are solved by numerical methods.

If n of $(t_k,\ m_k)$ is sufficiently large, then the MLE $\hat{a}$ and $\hat{r}$ asymptotically follow a BVN (bivariate *s*-normal distribution) [29], [32]:

$$\begin{pmatrix}\hat{a}\\ \hat{r}\end{pmatrix} \sim \mathrm{BVN}\left[\begin{pmatrix}\hat{a}\\ \hat{r}\end{pmatrix}, \Sigma\right] (n\to\infty).$$

The mean values of $\hat{a}$ and $\hat{r}$ are the true values of a and r, respectively, and the variance–covariance matrix Σ is given by the inverse matrix of the Fisher information matrix [5], [6], [26].

The Fisher information matrix $\mathbf{F}$ for $\hat{a}$ and $\hat{r}$ can be derived from $\log(L)$ as

$$\mathbf{F} = \begin{bmatrix} \mathrm{E}\left[-\frac{\partial^2\log(L)}{\partial\alpha^2}\right] & \mathrm{E}\left[-\frac{\partial^2\log(L)}{\partial\alpha\partial r}\right] \\ \mathrm{E}\left[-\frac{\partial^2\log(L)}{\partial\alpha\partial r}\right] & \mathrm{E}\left[-\frac{\partial^2\log(L)}{\partial r^2}\right] \end{bmatrix} = \begin{bmatrix} \frac{f_n}{a} & g_n \\ g_n & a\cdot\sum_{k=1}^{n}\frac{(g_k - g_{k-1})^2}{f_k - f_{k-1}} \end{bmatrix}; \quad \text{(A-10)}$$
$$f_k \equiv 1 - \exp[-r\cdot W^*(t_k)],$$
$$g_k \equiv W^*(t_k)\cdot\exp[-r\cdot W^*(t_k)].$$

Apply a and r to (A-10) and calculate F^{-1}. The estimated asymptotic variance–covariance matrix is

$$\hat{\Sigma} = F^{-1} = \begin{bmatrix} \mathrm{Var}[\hat{a}] & \mathrm{Cov}[\hat{a},\ \hat{r}] \\ \mathrm{Cov}[\hat{a},\ \hat{r}] & \mathrm{Var}[\hat{r}] \end{bmatrix}.$$

The Σ is useful in quantifying the variability of the estimated parameters.

ACKNOWLEDGMENT

The authors are pleased to thank Professor M. A. Vouk for his constructive and insightful suggestions for improving the details of this manuscript.

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, 1999.
[4] M. Ohba, "Software reliability analysis models," *IBM J. Research & Development*, vol. 28, no. 4, pp. 428–443, Jul. 1984.

[5] S. Yamada, H. Ohtera, and H. Narihisa, "Software reliability growth models with testing-effort," *IEEE Trans. Reliability*, vol. R-35, no. 1, pp. 19–23, Apr. 1986.
[6] S. Yamada, J. Hishitani, and S. Osaki, "Software reliability growth model with Weibull testing effort: A model and application," *IEEE Trans. Reliability*, vol. 42, pp. 100–105, 1993.
[7] P. K. Kapur and S. Younes, "Modeling an imperfect debugging phenomenon with testing effort," in *Proc. 5th Int. Symp. Software Reliability Engineering (ISSRE'1994)*, pp. 178–183.
[8] B. W. Boehm, *Software Engineering Economics*: Prentice Hall, 1981.
[9] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. Software Engineering*, vol. 23, pp. 736–743, 1997.
[10] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *IEEE Trans. Software Engineering*, vol. 21, no. 2, pp. 126–136, 1995.
[11] L. H. Putnam, "A general empirical solution to the macro software sizing and estimating problem," *IEEE Trans. Software Engineering*, vol. 4, pp. 345–367, 1978.
[12] J. Tian, P. Lu, and J. Palma, "Test-execution-based reliability measurement and modeling for large commercial software," *IEEE Trans. Software Engineering*, vol. 21, no. 5, pp. 405–414, May 1995.
[13] R. Chatterjee, B. Misra, and S. S. Alam, "Joint effect of test effort and learning factor on software reliability and optimal release policy," *Inter'l. J. Systems Science*, vol. 28, no. 4, pp. 391–396, 1997.
[14] K. Pillai and V. S. S. Nair, "A model for software development effort and cost estimation," *IEEE Trans. Software Engineering*, vol. 23, no. 8, Aug. 1997.
[15] Y. Tohma, K. Tokunaga, S. Nagase, and Y. Murata, "Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution," *IEEE Trans. Software Engineering*, vol. 15, no. 3, pp. 345–355, Mar. 1989.
[16] Y. Tohma, R. Jacoby, Y. Murata, and M. Yamamoto, "Hyper-geometric distribution model to estimate the number of residual software faults," in *Proc. COMPSAC-1989*, pp. 610–617.
[17] R. H. Hou, S. Y. Kuo, and Y. P. Chang, "Applying various learning curves to hyper-geometric distribution software reliability growth model," in *Proc. 5th Int. Symp. Software Reliability Engineering (ISSRE'1994)*, pp. 8–17.
[18] ——, "Optimal release policy for hyper-geometric distribution software reliability growth model," *IEEE Trans. Reliability*, vol. 45, no. 4, pp. 646–651, Dec. 1996.
[19] T. Rivers, "Modeling software reliability during nonoperational testing," Ph.D. dissertation, Department Computer Science, NC State University, Raleigh, 1998.
[20] C. Y. Huang, S. Y. Kuo, and I. Y. Chen, "Analysis of a software reliability growth model with logistic testing-effort function," in *Proc. 8th Int. Symp. Software Reliability Engineering (ISSRE'1997)*, pp. 378–388.
[21] 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 *Proc. 10th Int. Symp. Software Reliability Engineering (ISSRE'1999)*, pp. 62–72.
[22] C. Y. Huang, S. Y. Kuo, and M. R. Lyu, "Effort-index-based software reliability growth models and performance assessment," in *Proc. 24th Ann. Int. Computer Software and Applications Conf. (COMPSAC2000)*, pp. 454–459.
[23] F. N. Parr, "An alternative to the Rayleigh curve for software development effort," *IEEE Trans. Software Engineering*, vol. SE-6, pp. 291–296, 1980.
[24] T. DeMarco, *Controlling Software Projects: Management, Measurement and Estimation*: Prentice-Hall, 1982.
[25] P. N. Misra, "Software reliability analysis," *IBM Systems J.*, vol. 22, no. 3, pp. 262–279, 1983.
[26] S. Yamada, J. Hishitani, and S. Osaki, "Software reliability growth modeling: Models and applications," *IEEE Trans. Software Engineering*, vol. SE-11, no. 12, pp. 1431–1437, 1985.
[27] M. R. Lyu and A. Nikora, "Applying software reliability models more effectively," *IEEE Software*, pp. 43–52, Jul., 1992.
[28] Y. Liang and K. S. Trivedi, "Confidence interval estimation of NHPP-based software reliability models," in *Proc. 10th Int. Symp. Software Reliability Engineering (ISSRE'1999)*, pp. 6–11.
[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] M. Xie, "On the determination of optimum software release time," in *Proc. 2nd Int. Symp. on Software Reliability Engineering*, 1991, pp. 218–224.
[31] P. K. Kapur and R. B. Garg, "Cost-reliability optimum release policies for a software system with testing effort," *OPSEARCH*, vol. 27, no. 2, pp. 109–116, 1990.
[32] W. Nelson, *Applied Life Data Analysis*: Wiley, 1982.

Chin-Yu Huang was awarded an honorary degree (1989) in electronic engineering from Hocking Technical College, Ohio. From 1990 to 1992, he studied in the Department of Transportation Engineering and Management at National Chiao Tung University. He received the M.S. (1994) and the Ph.D. (2000) in electrical engineering from National Taiwan University. He was with the Bank of Taiwan from 1994 to 1999; and was a senior software engineer at Taiwan Semiconductor Manufacturing Company from 1999–2000. He is now a senior specialist at the Central Bank of China, Taipei. His research interests are software reliability, software testing, and fault-tolerant computing.

Sy-Yen Kuo received the B.S. (1979) in electrical engineering from National Taiwan University, the M.S. (1982) in electrical and computer engineering from the University of California at Santa Barbara, and the Ph.D. (1987) in computer science from the University of Illinois at Urbana-Champaign. Since 1991 he has been with National Taiwan University, where is currently Professor and Chairman in the Department of Electrical Engineering. He was Chairman of the Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan from 1995 to 1998, a faculty member in the Department of Electrical and Computer Engineering at the University of Arizona from 1988 to 1991, and an engineer at Fairchild Semiconductor and Silvar-Lisco, both in California, from 1982 to 1984. In 1989, he also worked as a summer faculty fellow at Jet Propulsion Laboratory of California Institute of Technology. His current research interests include dependable distributed systems and networks, software reliability engineering, and optical WDM networks. He received the distinguished research awards (1997–2001) from the National Science Council, Taiwan. He also received the Best Paper Award in the 1996 International Symposium on Software Reliability Engineering, the Best Paper Award in the simulation and test category at the 1986 IEEE/ACM Design Automation Conference (DAC), the National Science Foundation's Research Initiation Award in 1989, and the IEEE/ACM Design Automation Scholarship in 1990 and 1991.