Framework for Modeling Software Reliability, Using Various Testing-Efforts and Fault-Detection Rates

Sy-Yen Kuo, *Fellow, IEEE*, Chin-Yu Huang, and Michael R. Lyu, *Senior Member, IEEE*

***Abstract*—This paper proposes a new scheme for constructing software reliability growth models (SRGM) based on a nonhomogeneous Poisson process (NHPP). The main focus is to provide an efficient parametric decomposition method for software reliability modeling, which considers both testing efforts and fault detection rates (FDR). In general, the software fault detection/removal mechanisms depend on previously detected/removed faults and on how testing efforts are used. From practical field studies, it is likely that we can estimate the testing efforts consumption pattern and predict the trends of FDR. A set of time-variable, testing-effort-based FDR models were developed that have the inherent flexibility of capturing a wide range of possible fault detection trends: increasing, decreasing, and constant. This scheme has a flexible structure and can model a wide spectrum of software development environments, considering various testing efforts. The paper describes the FDR, which can be obtained from historical records of previous releases or other similar software projects, and incorporates the related testing activities into this new modeling approach. The applicability of our model and the related parametric decomposition methods are demonstrated through several real data sets from various software projects. The evaluation results show that the proposed framework to incorporate testing efforts and FDR for SRGM has a fairly accurate prediction capability and it depicts the real-life situation more faithfully. This technique can be applied to a wide range of software systems.**

***Index Terms*—Fault detection, nonhomogeneous Poisson process (NHPP), software faults and failures, software reliability growth model (SRGM), testing-effort functions.**

ACRONYMS[1]

AE	accuracy of estimation
DS	data set
FDR	fault detection rate
HPP	homogeneous Poisson process
K–S	Kolmogorov–Smirnov
LOC	lines of source code
LSE	least squares estimate
MLE	maximum likelihood estimate
MSF	mean of square fitting faults
NFD	number of faults detected
NHPP	non-HPP
RE	relative error
SRGM	software reliability growth model
TE	testing effort

Notation

$m(t)$	mean number of faults detected in time $(0, t]$: mean value function
$\lambda(t)$	$dm(t)/dt$: failure intensity for $m(t)$
$w(t)$	current TE consumption at time t
$W(t)$	cumulative TE consumption at time t
$\Delta W(\cdot)$	$W(\cdot) - W(0)$
a	anticipated number of initial faults
r	FDR per unit of TE
r_0	initial value of r
r_f	final value of r
w_0	initial TE
$g(t)$	consumption rate of the TE expenditures for a Weibull-type curve at time t
N	total amount of TE eventually used
α	consumption rate of TE expenditures in the logistic TE function
A	constant parameter in the logistic TE function
β, m	[scale, shape] parameter in the Weibull TE function
$d(t)$	detectability of an error for the current error content
$U(t)$	Laplace trend factor.

Manuscript received February 6, 1999; revised February 28, 2000. This work was supported by the National Science Council, Taiwan, ROC and the Taiwan Power Company, under Grant NSC 87-TPC-002-017. It was also partially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, under Project CUHK4432/99E.

Responsible Editor: K. Kanoun

S.-Y. Kuo and C.-Y. Huang are with the Electrical Engineering Department, National Taiwan University, Taipei, Taiwan (e-mail: SYKuo@cc.ee.ntu.edu.tw; CY2Huang@mail.cbc.gov.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).

Publisher Item Identifier S 0018-9529(01)11175-9.

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

I. INTRODUCTION

DUE to the rapid development of computer and information technology, society increasingly depends on software-intensive systems. Software is embedded in many modern systems, including expensive scientific computing systems, financial banking systems, industrial applications, university computer centers, and home personal computers. Since the demands for complex and large-scale software systems are increasing more rapidly, the possibility of programmers' design errors in the systems will grow appreciably. Consequently, the possibility of crises due to software failures will continue to increase. These failures can generate enormous losses of revenue for many enterprises. Therefore, to determine system reliability, the software reliability must be carefully evaluated.

Software reliability is similar to hardware reliability because both can be described by probability distributions. However, software faults are harder to visualize, detect, and correct, compared with physical hardware faults. The reliability of any system depends on the correctness of the system design,

the correctness of the mapping of the system design to implementation, and the reliability of the system-components. Software systems are quite different from hardware systems because the former do not wear out, at least in a physical sense. Therefore, fault and failure patterns for software and hardware are different, and the difference is fundamental Thus, the same models used for measuring hardware reliability cannot be used for measuring software reliability.

Hardware has mixtures of decreasing and increasing failure rates [1], [2]. The decreasing failure rate is caused by repairing original design-related hardware failures. The increasing failure rate is due to hardware component aging or wear-out. On the other hand, software systems usually have a decreasing failure rate [1]–[3]. And there is some confusion as to whether software reliability is probabilistic or deterministic. The answer depends on how we view software reliability. However, software has to be considered with its environment. Various software reliability measurement indicate different values for the same software under different testing/operating environments. According to the ANSI definition [4]: "Software reliability is the probability of failure-free software operation for a specified period of time in a specified environment." Hence, accurately modeling software reliability, and predicting its possible trends, are essential in determining the system reliability. To achieve a highly reliable software system, many software fault detection/removal techniques can be used by the program developers or testing teams. In applying these techniques, the SRGM are important, because they can provide quite useful information for developers and testers during the testing/debugging phase.

Numerous fault-prediction models are published, and many efforts were made to estimate software reliability from real DS. Most of them are based on

- calendar-time, e.g., the Jelinski–Moranda Model [4],
- staff-time, e.g., Shooman Model [4], or
- computer-time, e.g., Musa Model [4], [5].

Musa [5], [6] first discussed the validity of execution-time theory by taking DS from real software systems. However, most existing DS are not based on the execution-time concept. Recently, [1]–[3] and [7]–[9] proposed two simple SRGM with Weibull-type and logistic TE functions, respectively. These models attempt to account for the relationship among the calendar testing, the amount of TE, and the number of software faults detected during testing. The TE can be measured by the human power, the number of test cases, the number of CPU hours, etc. When applied extensively to real software development projects, these models provide a reasonable fit to the observed data and give insightful interpretations for the resource consumption process during the software development phase [7]–[9].

In general, among various SRGM, two most important factors affect reliability: the number of initial faults and the FDR. The number of initial faults is the number of faults in the software at the beginning of the test. This number is usually a representative measure of software reliability. Knowing the number of residual faults can help to determine whether the software is suitable for customers to use or not, and how much more testing resources are required. It can provide an estimate of the number of failures that will eventually be encountered by the customers. The FDR, on the other hand is used to measure the effectiveness of fault detection by test techniques and test cases. In the vast literature [1]–[6], and [10]–[19], most researchers assume a constant FDR per fault in deriving their SRGM. That is, they assume that all faults have equal probability of being detected during the software testing process, and the rate remains constant over the intervals between fault occurrences.

In reality, the FDR strongly depends on the skill of test teams, program size, and software testability. Through real data experiments and analyzes on several software development projects [20], the FDR has 3 possible trends as time progresses: increasing, decreasing, or constant. We thus treat the FDR as a function of time to interpret these possible trends; *viz*, we assume a time-variable fault detection function. And, we consolidate these 2 concepts, TE function and time-variable FDR, into a combined analysis for software reliability modeling.

This paper estimates the SRGM parameters by MLE and LSE methods. We take the estimated parameters into the proposed software-fault prediction model and compare the predicted results with other existing SRGM. From the comparison results, our analysis determines the reasons why the predicted results agree or disagree with the actual results. Experimental results show that the combined model gives a more accurate prediction, and depicts the real-life situation more faithfully.

Section II describes a basic SRGM that combines the TE function and the time-variable FDR.

Section III extends the basic SRGM to consider various time-dependent FDR, and discusses several modeling propositions.

Section IV estimates these parameters of the proposed SRGM based on the actual observed software failure data, plots their mean value functions, and fairly compares them with other existing models.

Section V discusses FDR and the reliability trend.

II. TE-Based Software Reliability Modeling

The mathematical expression of TE-based is:

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

This basic SRGM (1) is based on the assumptions:

1) The fault removal process is 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 $(t, t+\Delta t]$, $dm(t)/dt$ by the current $w(t)$, is proportional to the mean number of remaining faults in the system.
4) The $r(t)$ is a function of time (not just a constant).
5) The time-dependent behavior of TE can be modeled by a Logistic or a Weibull distribution.
6) Each time a failure occurs, the fault that caused it is immediately and perfectly removed, and no new faults are introduced.

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

Eq. (1) has 2 elements which influence the number of faults detected: $w(t)$ and $r(t)$. If $r(t)$ is constant over time, then

$$\frac{dm(t)}{dt} = w(t) \cdot r \cdot [a - m(t)]. \tag{2}$$

Solve (2) using the boundary condition $m(0) = 0$:

$$m(t) = a \cdot [1 - \exp(-r \cdot [w(t) - w(0)])]. \tag{3}$$

A. The TE Function

The first component in (1) is the TE function. During the software testing/debugging phase, appreciable TE, e.g., number of test cases, human Power, and CPU time, is consumed. The consumed TE can indicate how effective software faults are detected. Hence, resource consumption or allocation of human power can be modeled by various distributions. From the studies in [1]–[3], [7]–[9], [20], [21], several TE pattern expressions exist, as shown in this Section II-A.

1) Constant TE Consumption: In the derivation of classical SRGM [4]–[6], [11]–[14], [17], [22], most researchers assumed that the TE (workload) of a software system is constant:

$$w(t) = w_0. \tag{4}$$

2) Weibull-Type TE Function: According to [1]–[3], [5], [23], TE should not be assumed constant throughout the testing phase. In fact, instantaneous TE ultimately decreases during the testing life cycle so that the cumulative TE approaches a finite limit. This analysis is reasonable because no software company will spend infinite resources on software testing. Thus we describe TE a Weibull distribution:

$$W(t) = N \cdot \left[1 - \exp\left(-\int_0^t g(\tau)\, d\tau\right)\right] \tag{5}$$

$$W(t) \equiv \int_0^t w(\tau)\, d\tau. \tag{6}$$

There are 3 cases.

1) $g(t) = \beta$. Then $w(t) = N \cdot \beta \cdot \exp(-\beta \cdot t])$. There is an exponential curve, and the cumulative TE in $(0, t]$ is

$$W(t) = N \cdot [1 - \exp(-\beta \cdot t)]. \tag{7}$$

2) $g(t) = \beta \cdot t$. Then

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

There is a Rayleigh curve and the cumulative TE is

$$W(t) = N \cdot \left(1 - \exp\left[-\frac{\beta}{2} \cdot t^2\right]\right). \tag{8}$$

3) $w(t) = N \cdot \beta \cdot m \cdot t^{m-1} \cdot \exp[-\beta \cdot t^m]$. There is a Weibull curve, and the cumulative TE is

$$W(t) = N \cdot (1 - \exp[-\beta \cdot t^m]). \tag{9}$$

3) Logistic TE Function: When $m > 3$, the Weibull-type TE curves have an apparent peak phenomenon [7]–[9], [20]. This phenomenon does not seem realistic because it is not commonly used in an actual software development/test process. Therefore, we suggest a logistic TE function to describe the test effort patterns. Also, [24] reported that this function was fairly accurate in the 1978–1980 project survey. Therefore

$$W(t) = \frac{N}{1 + A \cdot \exp(-\alpha \cdot t)}, \tag{10}$$

$$w(t) = \frac{dW(t)}{dt} = \frac{N \cdot A \cdot \alpha \cdot \exp(-\alpha \cdot t)}{[1 + A \cdot \exp(-\alpha \cdot t)]^2} = \frac{N \cdot A \cdot \alpha}{\left[\exp\left(\frac{\alpha \cdot t}{2}\right) + A \cdot \exp\left(-\frac{\alpha \cdot t}{2}\right)\right]^2}. \tag{11}$$

B. The FDR

The second component in (1) is the FDR: the rate of discovering new faults in software during the testing phase. Typically, whether the software faults can be detected or not, depends on the ability of programmers/debuggers, the software structure, the maturity of software development procedure, and the correlation among program modules.

- At the beginning of the testing phase, many faults can be discovered by inspection, and the FDR depends on the fault-discovery efficiency, the fault density, the TE, and the inspection rate.
- In the middle stage of the testing phase, the FDR usually depends on other parameters such as the execution rate of CPU instruction, the failure-to-fault relationship, the code expansion factor, and the scheduled CPU execution hours per calendar day [4].

Consequently, the FDR can be calculated. We use this rate to track the progress of checking activities, to evaluate the effectiveness of test planning, and to assess the checking methods we adopted.

1) Constant Proportionality: Most existing SRGM assume that the mean number of faults detected in $[t, t + \Delta t)$ is proportional to the number of remaining faults [4], [11]–[14], [17], [22]:

$$m(t + \Delta t) = r \cdot w(t) \cdot [a - m(t)] \cdot \Delta t, \tag{12}$$

r is a constant proportionality.

2) Time-Variable FDR: In our experiments, the FDR is measured by the "average number of faults detected per TE expenditure" or by the "number of faults detected by special checking activities." This information is helpful for the system developers to plan the checking activities, diagnose problems, and assess the effects of changes. And, it indicates cost-effectiveness of various checking activities in the long run. To interpret the possible variation in FDR with time, we survey some real test/debugging DS [4], [5], [11], [12], [15], [17], [18], [22], [25]–[27]. We analyze those fault-detection processes and observe various fault-detection behaviors. Most of the grouped DS have the form:

$$(t_0, m_0), (t_1, m_1), \ldots, (t_n, m_n);$$

$m_j \equiv$ total number of faults detected by t_j.

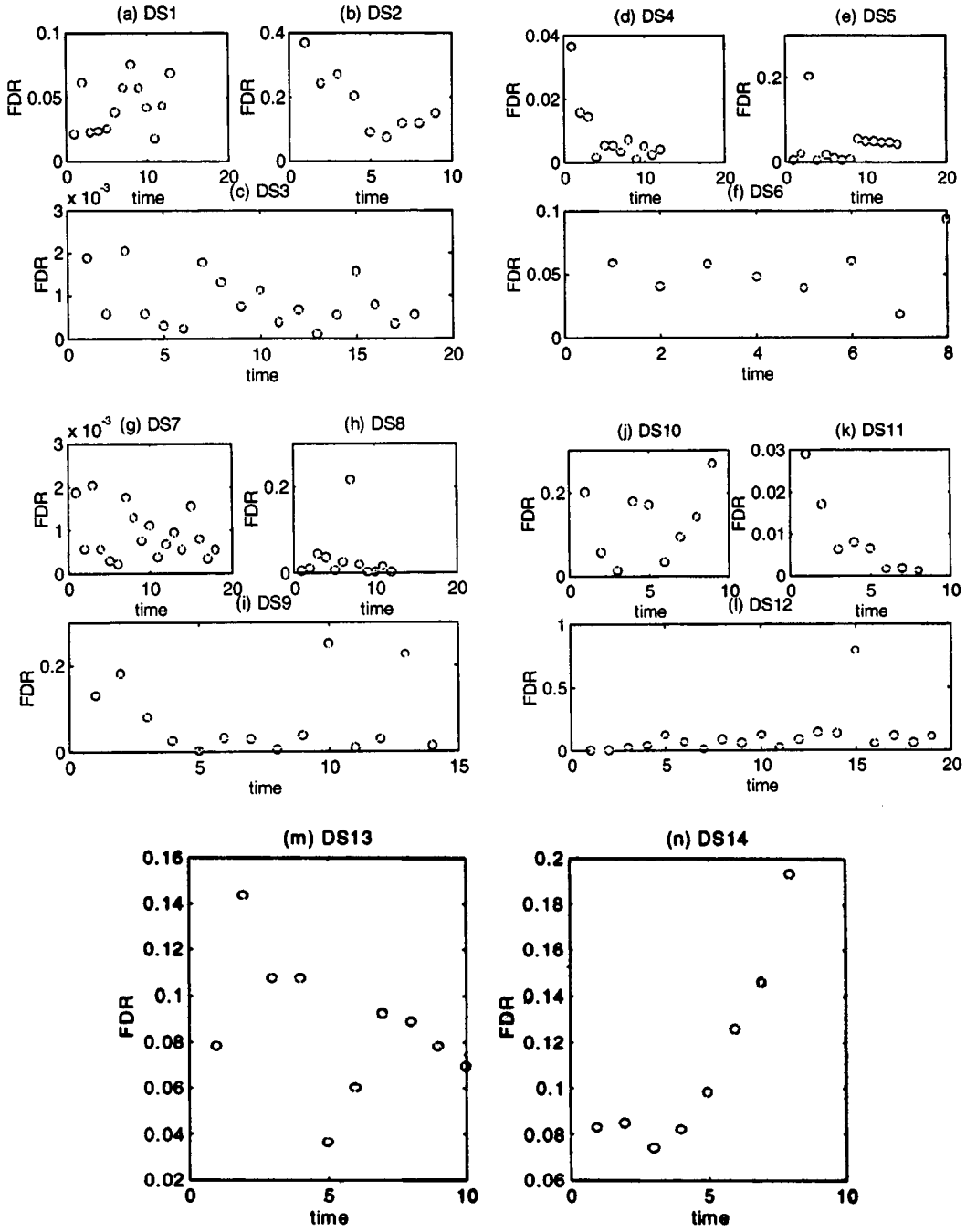

Fig. 1. Variation of FDR with time.

Generally speaking, the data obtained based on calendar time tends to be noisy (short-term randomness) and might not comply with most assumptions for existing SRGM [4], [5], [28]. One way of interpreting FDR at various times is to use the computation approach in [28]. From (3), and using $m(t_i)$, $m(t_{i+1})$, the FDR between t_i and t_{i+1} is estimated by:

$$\frac{m(t_i)}{m(t_{i+1})} = \frac{a \cdot [1 - \exp(-r \cdot \Delta W(t_i))]}{a \cdot [1 - \exp(-r \cdot \Delta W(t_{i+1}))]}, \tag{13}$$

or

$$m(t_i) \cdot [1 - \exp(-r \cdot \Delta W(t_{i+1}))] - m(t_{i+1}) \cdot [1 - \exp(-r \cdot \Delta W(t_i))] = 0. \tag{14}$$

Eq. (14) is solved numerically [29] (usually on a computer). Fig. 1 shows that the FDR varies with time for various real DS:

- Fig. 1(a), (e), (f), (j), (n) show that FDR has an increasing trend as time progresses;
- Fig. 1(b)–(d), (g), (k), (m) show that FDR is nonincreasing over time t;
- Fig. 1(h), (i), (l) show that FDR seems to be a constant.

There are some peaks and valleys in describing the possible FDR states. This might be due to sudden changes of test schemes, test teams, or the software under test.

A software testing process consists of several testing stages, including unit testing, integration testing, system testing, and

installation testing. If the software system is very large and complex, programmers can usually remove the easy-to-detect faults in their programs through inspection at the early stage of software testing. As time passes, the testing phase proceeds to the integration testing and the system testing phases, and it becomes more difficult for the programmers to detect the remaining faults. In this case, initially FDR is increasing and then it is decreasing [see Fig. 1(b), (k), or (m)]. In other words, the length of time between fault-detection will increase. On the other hand, if the software are not large and do not have many program modules, then the testing skills of programmers will improve as time progresses, and more efficient testing schemes can be conducted. Accordingly, the FDR can have an increasing trend [see Fig. 1(f), (j), or (n)]. If the requirements are changed and new features are added, or if new faults are introduced during debugging, the FDR will also increase. In any case, the FDR has three possible trends as time progresses: increasing, decreasing, and constant, as shown in the real project data. Section III discusses software reliability modeling for various FDR.

III. Investigation on FDR

We examine specific expressions of FDR and make several propositions. Because the fault-detection task is performed by programmers after coding, they will analyze the source-code or the results from object-code executions. Section II-B-2 suggests that a time-dependent coefficient can replace the constant FDR assumption [20]. To interpret the results, we assume that the FDR is a function of $m(t)$. That is, there is some relation among the number of initial faults, the number of detected faults, and the FDR. By re-arranging (1), the FDR per remaining fault at testing time t can be described. This represents the detectability of a fault for the current fault content [3], [17], [36]:

$$d(t) = \frac{dm(t)}{dt} \cdot \frac{1}{a - m(t)} = r(t) \cdot w(t), \tag{15}$$

i.e.,

$$r(t) = \frac{d(t)}{w(t)},$$

Eq. (15) implies that $d(t) \propto r(t)$: $r(t) \uparrow$, $d(t) \uparrow$ and $r(t) \downarrow$, $d(t) \downarrow$; and implies that the FDR per remaining fault is a function of the current $w(t)$. We view $d(t)$ as a software reliability-growth index. Most software reliability models assume that $w(t)$ is a constant (some do not even consider it, and they set r as a constant rate [4], [5], [11]–[13]). For $w(t) =$ constant, then $d(t) =$ constant, which indicates that these models have a homogeneous FDR. In the real projects, however, (15) gives a more precise description about the behavior of $d(t)$. Several scenarios are discussed in Sections III-A–III-C.

A. Proposition 1: Constant FDR for $r(t)$

If $r(t)$ is constant in t, then $m(t)$ is a constant FDR.

Case 1:

$$r(t) = r. \tag{16}$$

Eq. (16) shows that all faults are equally detectable during testing. Under this assumption, the FDR per unit TE is constant. Substitute (16) into (1) and solve the differential equation under the boundary condition, $m(0) = 0$ [8], [9]:

$$m(t) = a \cdot [1 - \exp(-r \cdot \Delta W(t))]. \tag{17}$$

From (15), the FDR per remaining fault at testing time t is

$$d(t) = r \cdot w(t). \tag{18}$$

Eq. (18) indicates that $d(t)$ is dominated by $w(t)$, whether $d(t)$ is homogeneous or nonhomogeneous.

B. Proposition 2: Non-Decreasing $r(t)$

If $r(t)$ is nondecreasing in t, then $m(t)$ is an increasing fault detection function.

Case 2.1:

$$r(t) = r(0) + k \cdot \frac{m(t)}{a}, \qquad k > 0. \tag{19}$$

Under this assumption, we use a linear model to describe the FDR. In (19), r_0 is the initial FDR and k is the slope (model parameter) which can be estimated by LSE. The k is used to track and predict the increasing FDR trends. Substitute (19) into (1), and solve the differential equation:

$$m(t) = a \cdot \left(1 - \frac{r_0 + k}{r_0 \cdot \exp[(r_0 + k) \cdot \Delta W(t)] + k}\right). \tag{20}$$

From (15), the FDR per remaining fault is

$$f(t) = w(t) \cdot \left(1 - \frac{k}{\exp[(r_0 + k) \cdot \Delta W(t)] + k}\right), \tag{21}$$

and is monotonically increasing with time t. That is, (21) implies that (20) describes a fault detection process in which the detectability of a fault increases with the progress of software testing.

Case 2.2:

$$r(t) = r_0 + (r_f - r_0) \cdot \frac{m(t)}{a}, \qquad 0 < r_0 < r_f. \tag{22}$$

Substitute (22) into (1); the result is a Riccati differential equation with solution:

$$m(t) = a \cdot \left(1 - \frac{r_f}{r_0 \cdot \exp[r_f \cdot \Delta W(t)] + r_f - r_0},\right) \qquad 0 < r_0 < r_f. \tag{23}$$

Similarly, from (15), the FDR per remaining fault at testing time t is

$$d(t) = w(t) \cdot r_f \cdot \left(1 - \frac{r_f - r_0}{r_0 \cdot \exp[r_f \cdot \Delta W(t)] + r_f - r_0}\right), \qquad 0 < r_0 < r_f \tag{24}$$

which is monotonically increasing. Similarly, (24) means that (23) describes a fault-detection process in which the detectability of a fault increases with the progress of software testing.

C. Proposition 3: Non-Increasing $r(t)$

If $r(t)$ is nonincreasing with time t, then $m(t)$ is a decreasing FDR function. This case describes the situation that many easy faults are effectively detected in the beginning and the last few faults are more difficult to detect.

Case 3.1:

$$r(t) = r_0 \cdot \left(1 - \frac{m(t)}{a}\right). \tag{25}$$

Substitute (25) into (1); the solution is:

$$m(t) = a \cdot \left(1 - \frac{1}{r_0 \cdot \Delta W(t) + 1}\right), \qquad 0 < r_0 < r_f. \quad (26)$$

From (15), the FDR per remaining fault at test time t is

$$d(t) = \frac{r_0 \cdot w(t)}{r_0 \cdot \Delta W(t) + 1}. \quad (27)$$

Case 3.2:

$$r(t) = r_0 + k \cdot \frac{m(t)}{a}, \qquad k < 0. \quad (28)$$

Substitute (28) into (1), the solution is:

$$m(t) = a \cdot \left(1 - \frac{r_0 + k}{r_0 \cdot \exp[(r_0 + k) \cdot \Delta W(t)] + k}\right). \quad (29)$$

From (15), the FDR per remaining fault at test time t is

$$d(t) = w(t) \cdot \left(1 - \frac{k}{\exp[(r_0 + k) \cdot \Delta W(t)] + k}\right). \quad (30)$$

Case 3.3:

$$r(t) = r_0 + (r_f - r_0) \cdot \frac{m(t)}{a}, \qquad r_0 < r_f. \quad (31)$$

Substitute (31) into (1), the solution is:

$$m(t) = a \cdot \left(1 - \frac{r_f}{r_0 \cdot \exp[(r_f) \cdot \Delta W(t)] + r_f - r_0}\right), \qquad r_0 > r_f. \quad (32)$$

From (15), the FDR per remaining fault at test time t is

$$d(t) = w(t) \cdot r_f \cdot \left(1 - \frac{k}{r_0 \cdot \exp[r_f \cdot \Delta W(t)] + r_f - r_0}\right). \quad (33)$$

D. Model Classification

Table I divides the propositions in Sections III-A–III-C into 4 groups.

Various TE functions can be applied in each of these 4 groups to form a particular software reliability model. Consider 4 TE functions: Logistic, Weibull, Rayleigh, and Exponential; then there is a maximum of 16 models based on our approach.

IV. Experimental Studies and Results

To check the performance of our models in Section III, and to make a fair comparison with other existing SRGM, we apply 3 DS to our models. These DS are in Table II. The 2 comparison criteria for evaluations among accepted models are:

1) AE [5], [8], [10], [14]

$$AE = \left|\frac{M_a - a}{M_a}\right|. \quad (34)$$

TABLE I
Classification of the Models

Group	FDR	Case
A	$r(t) = r$	1
B	$r(t) = r_0 + (r_f - r_0) \cdot \frac{m(t)}{a}$; $0 < r_0 < r_f$ or $0 < r_f < r_0$	2.2, 3.3
C	$r(t) = r_0 + k \cdot \frac{m(t)}{a}$; $k > 0$ or $k < 0$	2.1, 3.2
D	$r(t) = r_0 \cdot \left(1 - \frac{m(t)}{a}\right)$; $r_0 > 0$	3.1

$M_a \equiv$ actual cumulative number of detected faults during the test and after the test.

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

2) MSF

$$MSF = \frac{1}{k} \cdot \sum_{i=1}^{k} [m(t_i) - m_i]^2. \quad (35)$$

A smaller MSF indicates a smaller fitting error and better performance.

Other useful quantitative measures to assist in determining the number of residual faults and the probability that a software system can survive up to a certain time, are:

1) MF (maximum faults): total number of initial faults, $m(\infty)$;
2) RF (remaining faults in the system at test time t): $m(\infty) - m(t)$, an important indicator of software reliability and a useful measure for planning maintenance activities;
3) MTTF (mean time to failure);
4) SR (software reliability) [8], [9], [19], [30].

A. DS #1

The system is a PL/I database application software. The α, β, m of the Weibull-type TE function in (7)–(9), and N, A, α of the Logistic TE function in (10) can be derived using MLE and LSE. Similarly, a, r_0, r_f, k of the mean value function can also be solved numerically. Fig. 2 shows the fitting of the estimated TE by using (7)–(10). Table III summarizes the estimated parameters for various TE functions, mean value function, and the comparison criteria. Our proposed software reliability growth function fits pretty well at the 5% s-significance level through the K–S goodness-of-fit [4]. From Table III, both MSF and AE in group B are less than those in other groups and the existing SRGM; therefore, group B could have a better goodness-of-fit. This performance improvement is achieved by the using 2 parameters to interpret the FDR patterns instead of the traditional assumption of constant FDR.

By adding an extra parameter in modeling the fault detection phenomenon, the estimation becomes more tedious because more numerical calculations are involved. However, the additional calculations can be fully automated. If high reliability is

TABLE II
SUMMARY OF REAL DS THAT WERE STUDIED

DS	NFD	Observation Period	Software Project/Program Descriptions and Characteristics
1 [11]	328	19 weeks	PL/I application Program, Execution Time: 47.65 CPU hours, Software Size: 1317k LOC
2 [27]	86	22 days	Execution Time: 93 CPU hours
3 [25]	136	21 weeks	Real-time Command & Control Application (System T1), Execution Time: 25.3 CPU hours, 9 Programmers, Software Size: 21.7k LOC

TABLE III
SUMMARY OF MODEL PARAMETERS AND COMPARISONS FOR DS #1

Model (Group A)	a	r		MSF	AE (%)
with Logistic function	394.076	0.0427223		118.29	10.07
with Weibull function	565.35	0.0196597		122.09	57.91
with Rayleigh function	459.08	0.0273367		268.42	28.23
with Exponential function	828.252	0.0117836		140.66	131.35
Model (Group B)	a	r_0	r_f	MSF	AE (%)
with Logistic function	337.41	0.018962	0.113343	163.095	5.75
with Weibull function	345.686	0.0125642	0.106949	91.0226	3.43
with Rayleigh function	371.438	0.0137198	0.0805023	158.918	3.75
with Exponential function	352.521	0.0108438	0.108197	83.9989	1.53
Model (Group C)	a	r_0	k	MSF	AE (%)
with Logistic function	430.662	0.0409427	−0.0146536	103.0326	20.11
with Weibull function	385.392	0.0229036	0.0393828	87.5831	7.65
with Rayleigh function	379.947	0.0239006	0.0385439	406.7115	6.13
with Exponential function	385.179	0.0180857	0.0547021	83.3452	7.69
Model (Group D)	a	r_0		MSF	AE (%)
with Logistic function	582.538	0.0308452		96.9321	62.72
with Weibull function	958.718	0.0118215		124.3994	167.79
with Rayleigh function	702.693	0.0191208		247.84	96.09
with Exponential function	1225.66	0.0082272		169.7194	242.36
Existing SRGM	a	r		MSF	AE (%)
G-O Model	760.0	0.0322688		139.815	112.29
Inflection S-Shaped Model	389.1	0.0935493		133.53	8.69
Delayed S-Shaped Model	374.05	0.197651		168.67	4.48
Exponential Model	455.371	0.0267368		206.93	27.09
HGDM	387.71	*		138.12	8.30

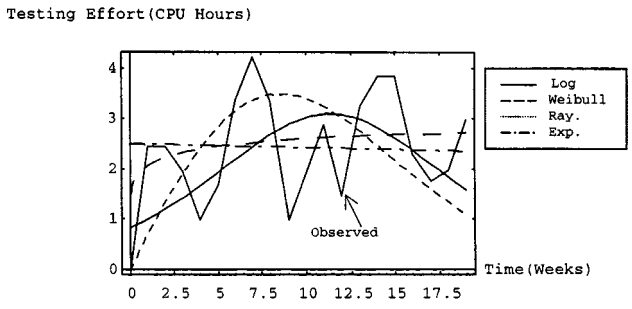

Fig. 2. Observed/estimated TE *vs* time, for DS #1.

required in some critical applications, such as large-scale commercial software or safety-critical flight software, the cost of extra computation for more accuracy is easily justified. From our simulations, the continuous curve of an estimated mean value function has an inflection point; i.e., it has S-shaped behavior due to $r_f \gg r_0$ in group B of Table III. The derived software reliability model under such an assumption, (22), was used [11], [12], [14], [20]. In summary, the combined model (group B) that incorporates the TE function and time-variable FDR, actually fits the DS satisfactorily in this experiment.

B. DS #2

This is the pattern of discovery of faults [27]. The cumulative number of discovered faults up to 22 days is 86 and the total consumed debugging time is 93 CPU hours. All debugging data are used in this experiment. Each parameter is estimated by MLE and LSE in the proposed SRGM, shown in Table IV. Fig. 3 fits the estimated TE by using (8)–(10). Table IV shows that the MSF for groups C and D are less than those of other groups and existing SRGM. The continuous curves for the estimated mean value function have an inflection point, showing S-shaped behavior due to $r_f > r_0$. In summary, the combined model (group C or D) of incorporating TE function and time-variable fault detection fits this DS better than others.

C. DS #3

This is the System T1 data of the US Rome Air Development Center (RADC) projects in [5], [6], [25]. The number of

TABLE IV
SUMMARY OF MODEL PARAMETERS AND COMPARISONS FOR DS #2

Model (Group A)	a	r		MSF
with Logistic function	88.8931	0.039059		25.2279
with Weibull function	87.0318	0.0345417		7.772
with Rayleigh function	86.1616	0.0359624		3.91643
Model (Group B)	a	r_0	r_f	MSF
with Logistic function	89.4528	0.188499	0.0543846	14.06603
with Weibull function	87.3126	0.017449	0.0522258	18.956772
with Rayleigh function	87.3472	0.0177506	0.0515699	20.4568
Model (Group C)	a	r_0	k	MSF
with Logistic function	97.5332	0.0472247	-0.0385523	7.354363
with Weibull function	97.6841	0.0360678	-0.0227224	6.5909
with Rayleigh function	112.182	0.0335812	-0.0335811	6.60318
Model (Group D)	a	r_0		MSF
with Logistic function	106.10	0.0437178		7.33727
with Weibull function	114.52	0.0314776		6.36531
with Rayleigh function	112.183	0.0335812		6.60318
Existing SRGM	a	r		MSF
G-O Model	137.072	0.0515445		25.33
Delayed S-Shaped Model	88.6533	0.228148		6.31268
HGDM	88.30	*		33.6812

object instructions in System T1 (a real-time command and control application) is 21 700. It took 21 weeks to complete the test by 9 testers. During this time, 136 faults were removed. Table V shows the estimated parameters of various SRGM. Our proposed models (Groups A–D) fit the failure data at a 5% level of s-significance through the K–S goodness-of-fit test; the other models do not have a good fit. Fig. 4 fits the estimated TE by using (7), (8), (10). For this DS, group B and C have a better goodness-of-fit measure than other groups and existing SRGM. Similarly, these continuous curves of estimated mean value function have inflection points. Hence, they are S-shaped.

V. FDR AND RELIABILITY TREND

Software reliability studies are usually based on the application of several SRGM to obtain various measures of interest. Reliability growth can be analyzed by trend tests. Blindly applying SRGM might not lead to meaningful results when the trend indicated by the data differs from that predicted by the model. If the model is applied to the data and shows a trend in accordance with its assumption, the results can be greatly improved [4], [5], [31]–[34]. Various statistical tests have been published (such as Kendall, Spearman or Laplace test) for identifying trends in grouped data or time-series. The Laplace test is very useful in determining the software reliability growth, and has excellent performance for NHPP. We calculate the Laplace trend factor $U(t)$ [31]–[33]. A negative $U(t)$ indicates a decreasing failure intensity, and thus, reliability growth. On the other hand, a positive $U(t)$ indicates an increasing failure intensity, and thus the reliability decreases. If $-2 \leq U(t) \leq 2$, then it indicates a stable reliability [4], [5], [31]–[34].

Using DS #1, Fig. 5 shows the Laplace trend test results. It shows that the trend test applied to this DS indicates a reliability growth. Thus, in this case our models can be applied. To illustrate, we only use Logistic TE function as the estimated value functions. Similar analyzes and discussions can be applied to the other TE functions or the other DS. The capability of the

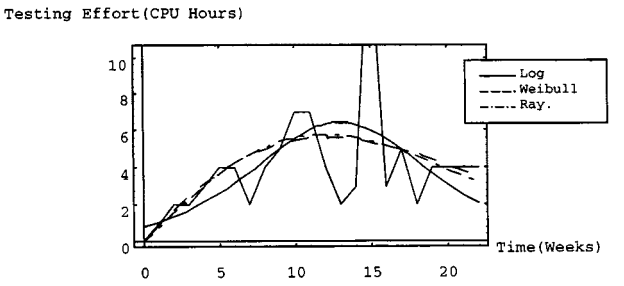

Fig. 3. Observed/estimated TE *vs* time, for DS #2.

TABLE V
SUMMARY OF MODEL PARAMETERS AND COMPARISONS FOR DS #3

Model (Group A)	a	r		MSF
with Logistic function	138.026	0.145098		62.41
with Rayleigh function	866.94	0.00962474		89.24095
Model (Group B)	a	r_0	r_f	MSF
with Logistic function	137.759	0.0502167	0.359256	14.6442
with Rayleigh function	150.047	0.013763	0.3222336	12.137
with Exponential function	187.537	0.000889141	0.166756	19.73719
Model (Group C)	a	r_0	k	MSF
with Logistic function	142.567	0.14881	-0.0450675	13.4266
with Rayleigh function	156.715	0.0183746	0.25801	10.9726
with Exponential function	173.064	0.000488055	0.194059	48.5971
Model (Group D)	a	r_0		MSF
with Logistic function	164.106	0.169151		38.121
with Rayleigh function	1543.47	0.00546049		89.7666
Existing SRGM	a	r		MSF
Exponential Model	137.2	0.156		3019.66
G-O Model	142.32	0.1246		2438.3
Delayed S-Shaped Model	237.196	0.0963446		245.246

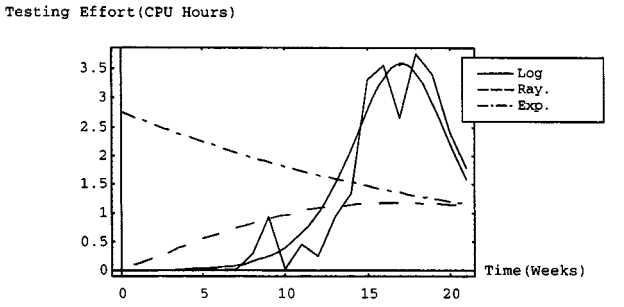

Fig. 4. Observed/estimated TE *vs* time, for DS #3.

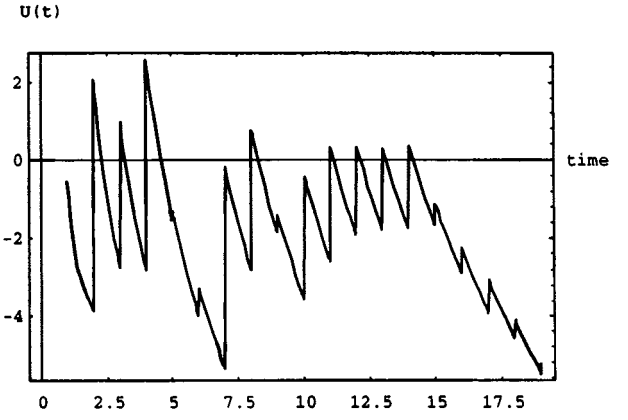

Fig. 5. Laplace trend test for DS #1.

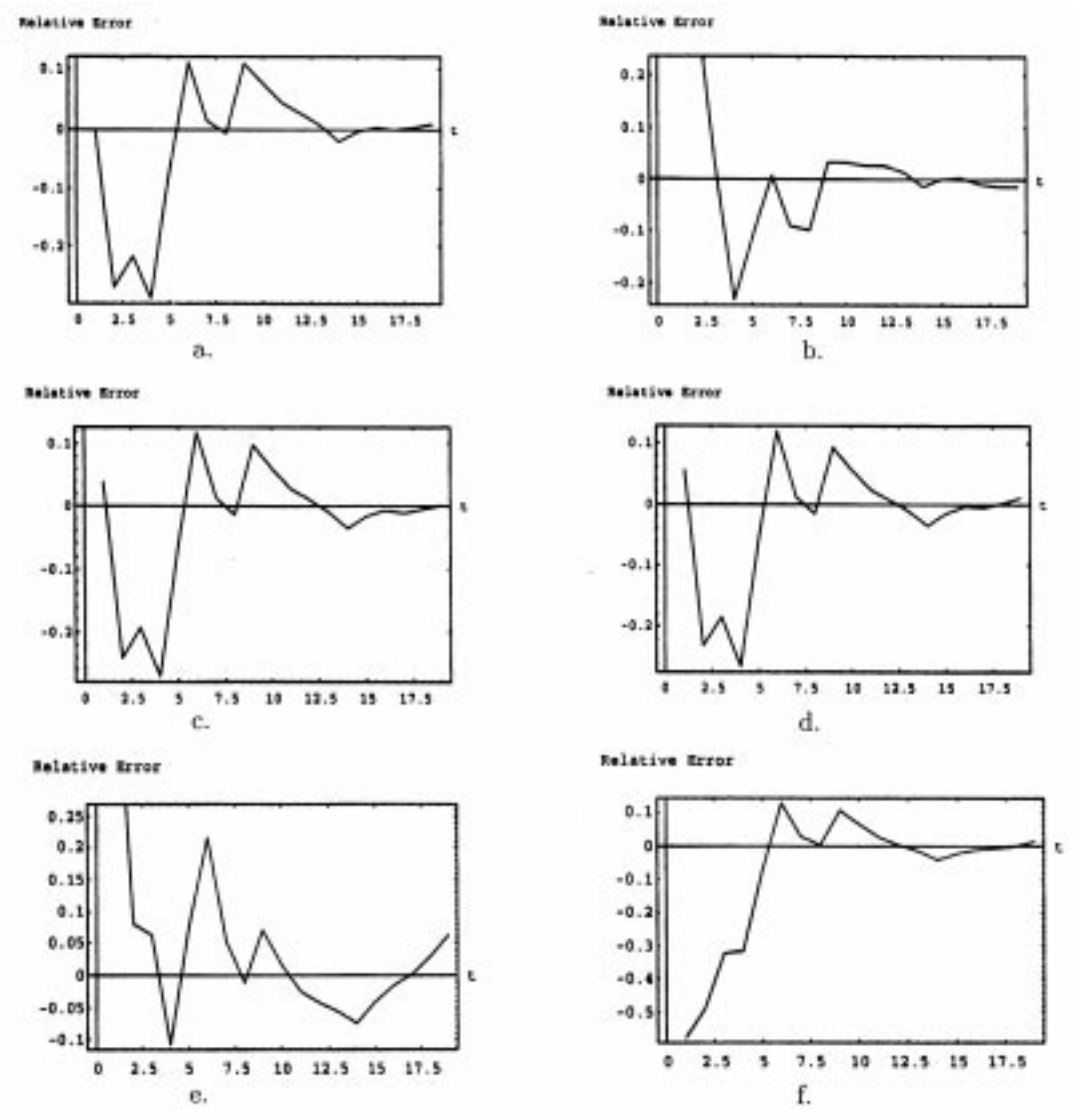

Fig. 6. Relative error curves for various SRGM, based on DS #1. (a), (b), (c), (d) depict the model with Logistic function in groups A, B, C, D, respectively. (e) is the G-O model; (f) is the Yamada S-shaped model.

model to predict failure behavior from present and past failure behavior is predictive validity [4]–[6], which can be represented by computing the RE for a DS:

$$\mathrm{RE} = \frac{m(t_q) - q}{q}. \tag{36}$$

q $\equiv$ number of failures
t_q $\equiv$ test time for q failures.

If q failures are observed by t_q, we use the failure data up to time t_e $(t_e \leq t_q)$ to estimate the parameters of $m(t)$. Substituting the estimates of these parameters in the mean value function yields the estimate of the number of failures $m(t_q)$ by t_q. The estimate is compared with q. The procedure is repeated for various values of t_e [5]. We can check the predictive validity by plotting the relative error *vs* t_q. Following [5] and using the real project failure data in Section IV-A, compute the relative error in prediction for the DS at the end of testing. Fig. 6 shows the results.

The comparisons in Fig. 4 show that the combined model (Group B) of incorporating the TE function and the time-variable fault detection has better predictive validity. Fig. 6 shows that the relative errors of the combined models (Groups A–D) approach zero faster than for the G-O model and the Yamada S-shaped model: these combined models can provide better estimation for this DS. Table VI shows the relative errors of various DS after the testing phase.

As stated in Section III, FDR is a metric which indicates a trend. It decreases when the software has been used and tested repeatedly, showing reliability growth. It can also increase if the testing techniques/requirements are changed, or new faults are introduced due to new software features or imperfect debugging. The experimental results showed that various SRGM incorporating various TE functions have different trends of FDR, even when they are simulated under the same DS. From the analysis results on 3 actual DS, the larger the FDR is as time progresses (FDR is increasing), the smaller the number of initial faults, and vice versa.

TABLE VI
COMPARISON OF RE (IN %) FOR DS1, DS2, DS3 AFTER THE TESTING PHASE

Model (Group A)	DS1	DS2	DS3
with Logistic function	0.75	-0.27	-1.05
with Weibull function	5.32	-2.68	-2.46
with Rayleigh function	-0.78	3.36	-2.34
with Exponential function	6.32	2.41	-1.98
Model (Group B)	**DS1**	**DS2**	**DS3**
with Logistic function	-0.215	9.09	1.22
with Weibull function	0.79	-0.48	2.57
with Rayleigh function	1.58	-9.09	2.44
with Exponential function	1.26	0.21	1.56
Model (Group C)	**DS1**	**DS2**	**DS3**
with Logistic function	0.23	-2.35	-0.53
with Weibull function	2.78	-1.23	-2.25
with Rayleigh function	-1.43	-1.16	3.4
with Exponential function	2.79	-1.48	1.25
Model (Group D)	**DS1**	**DS2**	**DS3**
with Logistic function	0.94	-2.17	-2.11
with Weibull function	5.85	-0.62	-0.14
with Rayleigh function	-0.74	-1.23	-0.07
with Exponential function	5.85	3.64	0.14

From our study [9], the first DS might have an imperfect debugging phenomenon, eflecting the characteristic of increasing FDR. In our models, sometimes we use 1 extra parameter to describe the fault detection/removal process, such as k, the slope parameter in (19), or r_f, the final FDR in (22). By adding a parameter in modeling the fault-detection process, the estimation procedure is more complicated because more numerical operations are involved. However, this approach does offer a better estimation result as shown in Tables III–V when the combination includes either a reasonable TE function or an adequate FDR function.

REFERENCES

[1] S. Yamada, H. Ohtera, and H. Narihisa, "Software reliability growth models with testing effort," *IEEE Trans. Reliability*, vol. R-35, pp. 19–23, Apr. 1986.

[2] 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.

[3] S. Yamada, H. Ohtera, and H. Narihisa, "A TE dependent software reliability model and its application," *Microelectronics and Reliability*, vol. 27, no. 3, pp. 507–522, 1987.

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

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

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

[7] C. Y. Huang, J. H. Lo, and S. Y. Kuo, "A pragmatic study of parametric decomposition models for estimating software reliability growth," in *Proc. 9th Int'l. Symp. Software Reliability Engineering (ISSRE'98)*, 1998, pp. 111–123.

[8] C. Y. Huang, S. Y. Kuo, and I. Y. Chen, "Analysis of a software reliability growth model with logistic TE function," in *Proc. 8th Int'l. Symp. Software Reliability Engineering (ISSRE'97)*, 1997, pp. 378–388.

[9] 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'l. Computer Software and Applications Conf. (COMPSAC 2000), 2000.

[10] 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'l. Symp. Software Reliability Engineering*, 1994, pp. 7–16.

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

[12] P. N. Misra, "Software reliability analysis," *IBM Systems J.*, vol. 22, no. 3, pp. 262–279, 1983.

[13] A. L. Goel and K. Okumoto, "Time-dependent fault detection rate model for software reliability and other performance measures," *IEEE Trans. Reliability*, vol. R-28, no. 3, pp. 206–211, 1979.

[14] 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.

[15] S. Yamada, S. Osaki, and H. Narihisa, "Software reliability growth modeling with number of test runs," *Trans. IECE Japan*, vol. E-67, no. 2, pp. 79–83, 1984.

[16] R. H. Huo, S. Y. Kuo, and Y. P. Chang, "Optimal release policy for hyper-geometric distribution software reliability growth model," *IEEE Trans. Reliability*, vol. 45, no. 4, pp. 646–651, Dec. 1996.

[17] Y. Tohma, H. Yamamoto, and R. Jacoby, "Parameter estimation of the hyper-geometric distribution for real/test data," in *Proc. 2nd Int'l. Symp. Software Reliability Engineering (ISSRE'91)*, 1991, pp. 28–34.

[18] K. Matsumoto, K. Inoue, T. Kikuno, and K. Torii, "Experimental evaluation of software reliability growth models," in *Proc. 18th Int'l. Symp. Fault-Tolerant Computing (FTCS)*, 1988, pp. 148–153.

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

[20] S. Bittanti, P. Bolzern, and E. P. EDRotti *et al.*, "A flexible modeling approach for software reliability growth," in *Software Reliability Modeling and Identification*: Springer-Verlag, 1988, pp. 101–140.

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

[22] 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. 1999.

[23] 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.

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

[25] J. D. Musa, "Software reliability data," report and database available from: Data and Analysis Center for Software; Rome Air Development Center (RADC), Rome, NY, USA.

[26] P. B. Moranda, "A comparison of software error-rate models," in *Texas Conf. on Computing*, 1975, pp. 2A-6.1–2A-6.9.

[27] Y. Tohma, R. Jacoby, Y. Murata, and M. Yamamoto, "Hyper-geometric distribution model to estimate the number of residual software faults," in *Proc. COMPSAC-89*, 1989, pp. 610–617.

[28] Y. K. Malaiya, A. V. Mayrhauser, and P. K. Srimani, "An examination of fault exposure ratio," *IEEE Trans. Software Engineering*, vol. 19, no. 11, pp. 1087–1094, 1993.

[29] D. Kahaner, *Numerical Methods and Software*: Prentice Hall, 1989.

[30] S. S. Gokhale, M. R. Lyu, and K. S. Trivedi, "Software reliability analysis incorporating fault detection and debugging activities," in *Proc. 9th Int'l. Symp. Software Reliability Engineering (ISSRE'98)*, 1998, pp. 202–211.

[31] K. Kanoun and J. C. Laprie, "Software reliability trend analyzes from theoretical to practical considerations," *IEEE Trans. Software Engineering*, vol. 20, no. 9, pp. 740–747, 1994.

[32] K. Kanoun, M. Martini, and J. Souza, "A method for software reliability analysis and prediction application to the TROPICO-R switching system," *IEEE Trans. Software Engineering*, vol. 17, no. 4, pp. 334–344, 1991.

[33] J. C. Laprie, K. Kanoun, C. Beounes, and M. Kaaniche, "The KAT (Knowledge–Action–Transformation) approach to the modeling and evaluation of reliability and availability growth," *IEEE Trans. Software Engineering*, vol. 17, no. 4, pp. 370–382, 1991.

[34] M. R. Martini and J. M. de Souza, "Reliability assessment of computer systems design," *Microelectronics Reliability*, vol. 31, no. 2/3, pp. 237–244, 1991.

[35] S. Yamada, S. Osaki, and H. Narihisa, "A fault detection rate theory for software reliability growth models," *Trans. IECE Japan*, vol. E-68, no. 5, pp. 292–296, May 1985.

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

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. He is a Professor and Chairman in the Department of Electrical Engineering, National Taiwan University. He was the 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 fault-tolerant parallel and distributed computing, software reliability, WDM and high speed networks, and low power VLSI systems. He received the distinguished research award in 1997 from the National Science Council, Taiwan. He received the Best-Paper Award in the 1996 Int'l. Symp. 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.

Chin-Yu Huang was awarded an honorary degree (1989) in electronic engineering from Hocking College, OH USA. 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) in electrical engineering from National Taiwan University, and is pursuing his Ph.D. in the Department of Electrical Engineering at National Taiwan University. He worked at the Bank of Taiwan and is now working at Taiwan Semiconductor Manufacturing Company. His research interests are software reliability, testing, and fault-tolerant computing.

Michael R. Lyu received the B.S. (1981) in electrical engineering from National Taiwan University, the M.S. (1985) in computer engineering from University of California, Santa Barbara, and the Ph.D. (1988) in computer science from University of California, Los Angeles. He is an Associate Professor at the Computer Science and Engineering Department of the Chinese University of Hong Kong. He worked at the Jet Propulsion Laboratory, Bellcore (now Telcordia) and Bell Labs, and taught at the University of Iowa. His research interests include software reliability engineering, distributed systems, fault-tolerant computing, web technologies, web-based multimedia systems, and wireless communications. He has published over 80 refereed journal and conference papers in these areas. Dr. Lyu initiated the first *Int'l. Symp. on Software Reliability Engineering* (ISSRE) in 1990. He was the program chairman for ISSRE'96 and PRDC'99, and has served on program committees for numerous international conferences. He is the editor for two books: *Software Fault Tolerance*, 1995 (Wiley) and the *Handbook of Software Reliability Engineering*, 1996 (IEEE and McGraw-Hill). He is on the editorial board for IEEE TRANSACTIONS KNOWLEDGE AND DATA ENGINEERING, IEEE TRANSACTIONS RELIABILITY, and *J. Information Science and Engineering*.