Pragmatic Study of Parametric Decomposition Models for Estimating Software Reliability Growth

Chin-Yu Huang, Jung-Hua Lo and Sy-Yen Kuo
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
sykuo@cc.ee.ntu.edu.tw
http://lion.ee.ntu.edu.tw

Abstract

Numerous stochastic models for the software failure phenomenon based on Nonhomogeneous Poisson Process (NHPP) have been proposed in the recent three decades. Although these models are quite helpful for software developers and have been widely applied in the industrial organizations or research centers, we still need to do more works on examining/estimating the parameters of existing software reliability growth models (SRGMs). In this paper, we investigate and account for three possible trends of software fault detection phenomenon during the testing phase: increasing, decreasing and steady state. We present empirical results from the quantitative studies on evaluating the fault detection process and develop a valid time-variable fault detection rate model which has the inherent flexibility of capturing a wide range of possible fault detection trends. The applicability of the proposed model and the related methods of parametric decomposition are illustrated through several real data sets from different software projects. Our evaluation results show that the analytic parametric decomposition approach for SRGM have a fairly accurate predicting capability. In addition, the testing-effort control problem based on the proposed model is also demonstrated.

1. Introduction

Due to the rapid development of computer and information technology, modern society has become increasingly dependent on software-intensive systems in the recent three decades. Software is embedded in everything and plays an important role from expensive scientific computing systems, financial banking systems or university computer centers to general industrial applications or home personal computers. Since these demands for complex and large-scale software systems have increased more rapidly than before, the possibility of programmers' design errors and incompleted debugs will grow relatively. Consequently, the possibility of crises due to computer failures increases significantly. These failures may generate enormous losses of revenues for many enterprises. Therefore, in order to determine the overall system's reliability, the software reliability must be considered and we should pay more attention to it.

Software reliability is similar to hardware reliability since both can be described by probability distributions. But software faults are harder to visualize, detect, and correct because they are not like the hardware's physical faults. According to the ANSI definition [17]: " *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 to determining the overall system reliability. In order to achieve a highly reliable software system, a number of software fault detection/removal techniques are widely used by the program developers or testing teams to detect and remove software faults. In fact, no matter how the fault detection/removal techniques are applied, the *Software Reliability Growth Models* (SRGMs) always play a rather important role and can provide quite useful information for developers during the testing/debugging phase.

There are numerous fault-prediction models published in the literature and many efforts have been made to estimate the software reliability from real data sets. Most of them are based on calendar-time (such as Jelinski-Moranda Model [17]), manpower-time (such as Shooman Model [17]) or computer-time (such as Musa

Model [5, 17]). In the field of software reliability modeling, Musa [5] first discussed the validity of execution time theory by taking data sets from real software systems. In fact, from the literature, we can find that most existing software reliability models do not incorporate the execution time concept into their derived models. Recently, Yamada et al. [1-5] and Huang et al. [8, 18] proposed two simple and new software reliability growth models with Weibull-type and Logistic testing-effort function respectively, which attempt to account for 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 measured by the man power, the number of test cases, the number of CPU hours, ...,etc. By applying these models extensively on real software development projects, we know that the testing-effort dependent software reliability model provides a reasonable fit to the observed data and gives a very good interpretation for the resource consumption process during software development [8, 18].

In general, among the various software reliability growth models, there are two most important parameters which will impact the reliability: *the total number of initial faults* and *the fault detection rate*. The total number of initial faults are the number of faults in the software at the beginning of test. The fault detection rate is used to measure the effectiveness of fault detection for test techniques and test cases. Based on the vast literature [1-5, 9-11, 16-17, 19, 21, 24, 28-29], we know that most researchers assume a *constant* fault detection rate in deriving their original software reliability growth models. That is, they assume that all faults are equally exposed during the software testing process and the rate remains constant over the intervals between fault occurrences. In fact, the fault detection rate strongly depends on the skill of test teams, program size, or testability. From our studies [20], through the real data experimrnts/analysis on several software development projects, we can find that the fault detection rate has three possible trends as time progresses: *increasing*, *decreasing* or *steady state*. We thus analyze and view the fault detection rate as a function of time to interpret these possible trends, that is, a time-variable fault detection function. Furthermore, we will consolidate the above two concepts, *testing-effort function* and *time-variable fault detection rate*, in the following analysis of software reliability modeling. In this paper, the SRGM's parameters are estimated by the *Maximum Likelihood Estimation* (MLE) and the *Least Squares Estimation* (LSE). We will take the estimated parameters into the proposed software-fault prediction model and compare the predicted results with other existing software reliability models. From the comparison results, analysis to determine the reasons why the predicted results do or don't agree with the actual results is performed. Experimental results show that the combined model gives a more accurate prediction. Further, the testing-effort control problem based on the derived model is also discussed.

There are six sections in this paper. Section 2 provides a detailed description of characteristics of model's parameters in the literature. Section 3 derives a software reliability growth model which combines the testing-effort function and the time-variable fault detection rate. We estimate these parameters of the proposed SRGM based on the actual observed software failure data, plot the mean value functions, and give a fair comparison with other existing models in section 4. Section 5 is concerned with the applications of testing-effort control and management problem. Finally, section 6 presents the concluding remarks and possible future extensions of this work.

2. Basics for Software Reliability Modeling

A. Assumptions [1-4, 8, 10]

1. The fault removal process follows the *Non Homogeneous Poisson Process* (NHPP).
2. The software system is subject to failures at random times caused by faults remaining in the system.
3. The mean number of faults detected in the time interval $(t, t+\Delta t]$ by the current testing-effort expenditures is proportional to the mean number of remaining faults in the system.
4. The proportionality is a function of time.
5. The time-dependent behavior of testing effort can be modeled by a Logistic or a Weibull-type distribution.
6. Each time a failure occurs, the fault which caused it is immediately and perfectly removed (i.e. no new faults are introduced) by the system programmers.

Based on the above assumptions, the mathematical development of software reliability growth model is as following:

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

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

Rearranging the above equation and if $r(t)$ is a constant over time, we get

$$\frac{dm(t)}{dt} = w(t) \times r \times [a - m(t)] \quad (2)$$

where $m(t)$=expected mean number of faults detected in time $(0, t]$.
a=expected number of initial faults.
r=fault detection rate (FDR) per unit testing-effor.
w=current testing-effort consumption.

Solving Eq. (2) under the boundary condition $m(0)=0$, we have

$$m(t) = a \times (1 - e^{-r \times (W(t) - W(0))}) \quad (3)$$

B. Testing-Effort Modeling

During the software testing/debugging phase, it consumes significant test-effort, such as volume of test cases, man power, and CPU time, ...etc. The consumed testing-effort can indicate how effective the faults are detected in the software. Hence, resource consumption or allocation of man power can be modeled by different distributions. From the studies in [1-4, 6-8, 18], several testing-effort pattern expressions exist as shown in the following.

B.1. Constant Testing-Effort Consumption

In the derivation of most classical software reliability growth models [10-11, 16-17, 19, 24-25, 28, 30-31], the researchers assumed that the testing-effort (workload) of a software system is constant.

$$w(t)=w_0,\ t=1, 2, ..., n. \tag{4}$$

where w_0 is the initial testing effort.

B.2. Weibull-Type Testing-Effort Function

According to Yamada et al. [1-4], Musa et al. [5] and Putnam [27], we know that the cumulative consumption of testing-effort during the testing phase may not be a constant and grows from zero to some finite value. Hence, the testing-effort can be described by a Weibull-type distribution:

$$W(t) = \alpha \times \left[1 - e^{\left(-\int_0^t g(x)dx\right)} \right] \tag{5}$$

where α is the total amount of testing effort to be eventually consumed.
$g(t)$ is the consumption rate of the testing effort expenditures at instant t.

And $W(t)$ is defined as follow:

$$W(t) \equiv \int_0^t w(x)dx \tag{6}$$

where $w(t)$ is the current testing-effort consumption at time t.

1. If $g(t)=\beta$, then $w(t)=\alpha\beta\exp[-\beta t]$, we have an *Exponential* curve and the cumulative testing-effort consumed in time $(0, t]$ is $W(t)=\alpha(1-\exp[-\beta t])$. (7)
2. If $g(t) = \beta t$, then $w(t)=\alpha\beta t\exp[-\frac{\beta}{2}t^2]$ and we have a *Rayleigh* curve and the cumulative testing-effort consumed is $W(t)=\alpha(1-\exp[-\frac{\beta}{2}t^2])$. (8)
3. And if $w(t)=\alpha\beta m t^{m-1}\exp[-\beta t^m]$, we have a *Weibull* curve and the cumulative testing-effort consumed is $W(t)=\alpha(1-\exp[-\beta t^m])$. (9)

   where β is the scale parameter and m is the shape parameter.

B.3. Logistic Testing-Effort Function

In the Weibull-type curves, when $m>3$, we find that Weibull-type testing-effort curves have an apparent peak phenomenon [6, 8, 18]. Therefore, we try to use a Logistic testing-effort function to describe the test effort patterns. Besides, Demarco also reported that this function was fairly accurate in the Yourdon 1978-1980 project survey [8, 18]. The cumulative testing effort consumption of Logistic testing-effort function in time $(0, t]$ is

$$W(t) = \frac{N}{1+Ae^{-\alpha t}} \tag{10}$$

and the current testing-effort consumption is

$$w(t) = \frac{dW(t)}{dt} = \frac{\alpha AN\, e^{-\alpha t}}{(1+Ae^{-\alpha t})^2} = \frac{\alpha NA}{\left(e^{\frac{\alpha t}{2}}+Ae^{\frac{-\alpha t}{2}}\right)^2} \tag{11}$$

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

C. Fault Detection Rate

The second parameter of Eq. (2) is the *fault detection rate*. It is the rate of discovering new faults in software during the testing phase. First, we should distinguish *error*, *fault* and *failure*. In considering a computer software system, an error occurs when some parts of the software produce an undesired state or it is the programmer action or omission that results in a fault. A fault is created in the written software (faulty instructions or data patterns) when a programmer makes an error. Consequently, a fault is a defective, missing, or extra instruction which is the cause of one or more actual failures. A fault causes failures and is uncovered when a failure occurs within the program [5]. Hence, we can clearly know that software reliability is the probability of failure-free operation of a software component or system in a specified environment for a specified period time.

Secondary, if a computer software program is designed by software designers or programmers, then the tasks of testing/debugging software may be performed by these people or other test teams after coding. Therefore, they should understand the characteristics of programming (such as number of lines of source code (LOC), language type, program size, modularity or complexity), inspection, testing, and operational environments. That is, whether the software faults can be detected or not depends on the abilities of programmers/debuggers, the software structure, the maturity of software development procedure, and the correlation among modules. At the beginning of the testing phase, most faults can be easily discovered by inspection and the fault detection rate depends on the discovery-to-fault relationship, the fault density, the testing-effort, and the inspection rate. On the other hand, in the middle stage of testing phase, the fault detection rate normally 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 hours per calendar day [17]. Consequently, we know that the fault detection rate can be calculated and is used to track the progress of checking activities and evaluate the effectiveness of planning how to test and the checking methods we adopted.

C.1. Constant fault detection rate

From our studies in [10-11, 16-17, 19, 24-25, 28], most existing SRGMs assumed that the fault detection rate remains constant over the intervals between fault occurrences. That is,

$$r(t)=r_0,\ t=1, 2, 3, 4, ..., n. \qquad (12)$$

where r_0 is the initial fault detection rate.

C.2. Time-variable fault detection rate

In our experiments, we know that the fault detection rate can be measured by the average number of faults detected per testing-effort expenditure or the number of faults detected by special checking activities. It is very helpful for the system developers to plan the checking activities, diagnose problems and assess the effects of changes. Besides, it provides enough information which we want to know about the cost-effectiveness of various checking activities during the long-term running. Therefore, in order to interpret the possible variation in fault detection rate (FDR) with time, we survey some real test/debug data sets given in [5, 10-12, 14-15, 25-28]. From those different software systems (from USA, Japan and France) in Table 1, we obtain adequate knowledge about the fault detection processes and observe the various fault detection behaviors. Most of the grouped data sets in Table 1 have the following form:

$(t_0 , m_0), (t_1 , m_1), (t_2 , m_2), (t_3 , m_3), (t_4 , m_4),....., (t_n , m_n)$

where m_j is the total number of faults detected by time t_j.

Generally, the obtained data based on calendar time tends to be noisy (short-term randomness) and might not comply with most existing assumptions for SRGMs [5, 18, 22]. If possible, it must be filtered by applying some data-smoothing techniques. One way of interpreting FDR at different times is to use *computational approach* [22]. From Eq. (3) and using $m(t_i)$ and $m(t_{i+1})$, the FDR during the time t_i and t_{i+1} can be estimated as following:

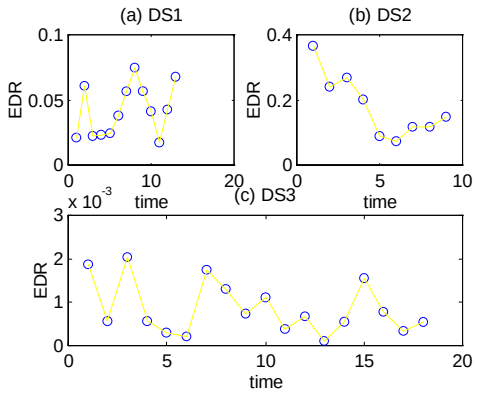
$$\frac{m(t_i)}{m(t_{i+1})} = \frac{a\times(1-e^{-r(W(t_i)-W(0))})}{a\times(1-e^{-r(W(t_{i+1})-W(0))})} \qquad (13)$$

Rearranging the above equation, we obtain

$$m(t_i)\times(1-e^{-r(W(t_{i+1})-W(0))}) - m(t_{i+1})\times(1-e^{-r(W(t_i)-W(0))}) = 0 \qquad (14)$$

We can solve the above equation by numerical methods.

After some numerical calculations by computer, Fig. 1 shows that the fault detection rate varies with time for different real data sets. Fig. 1 (a), (e), (f), (j) and (n) show that FDR has a rise trend as time increases. Fig. 1 (b), (c), (d), (g), (k), and (m) indicate that FDR is non-increasing in time *t*. Besides, Fig. 1 (h), (i), (l) show that FDR seems to be in a steady state. Here, we must point out: there are some peaks and valleys in describing the possible FDR states because these test cases may probably be switched to other test teams or make some modifications during the software testing phase. Additionally, in these experiments, we also eliminate some oscillation phenomena in the beginning. In fact, during the software testing process, there are several testing stages which includes *unit testing*, *integration testing*, *system testing* and *installation test.* If the whole software system is very complex and large, such as the space shuttle project, the weather prediction or airplane reservation systems, the programmers should remove all easy-to-detect errors in their own programs at the early stage of software testing phase. As time passes, the testing phase proceeds to the integration testing and system testing phase, such that it is relatively more difficult for programmers to detect other embedded errors. That is, initially FDR is increasing and then the FDR is decreasing in this case (see Fig. 1(b), (k) or (m)). On the other hand, if the software projects are designed for median to small scale business/company, they are usually not large in scale and does not own many program modules. As time progresses, the testing skill of programmers also improves or they have modified their testing techniques and tools when new technologies are discovered and become available. They can assimilate some new methodologies in fault detection, fault correction, or fault avoidance which are described in the professional journals, proceedings or trade publications. These modifications may help the programmers or testers in creating tests and easily eliminating some redundant tests. Accordingly, the FDR may have an increasing trend (see Fig. 1(f), (j) or (n)). Sometimes, if the requirements are changed or new faults are introduced during corrective activities, the FDR will increase. Hence, through real project data analysis, we clearly observe that the fault detection rate (FDR) have three possible trends as time progresses: *increasing*, *decreasing* and *steady-state*.



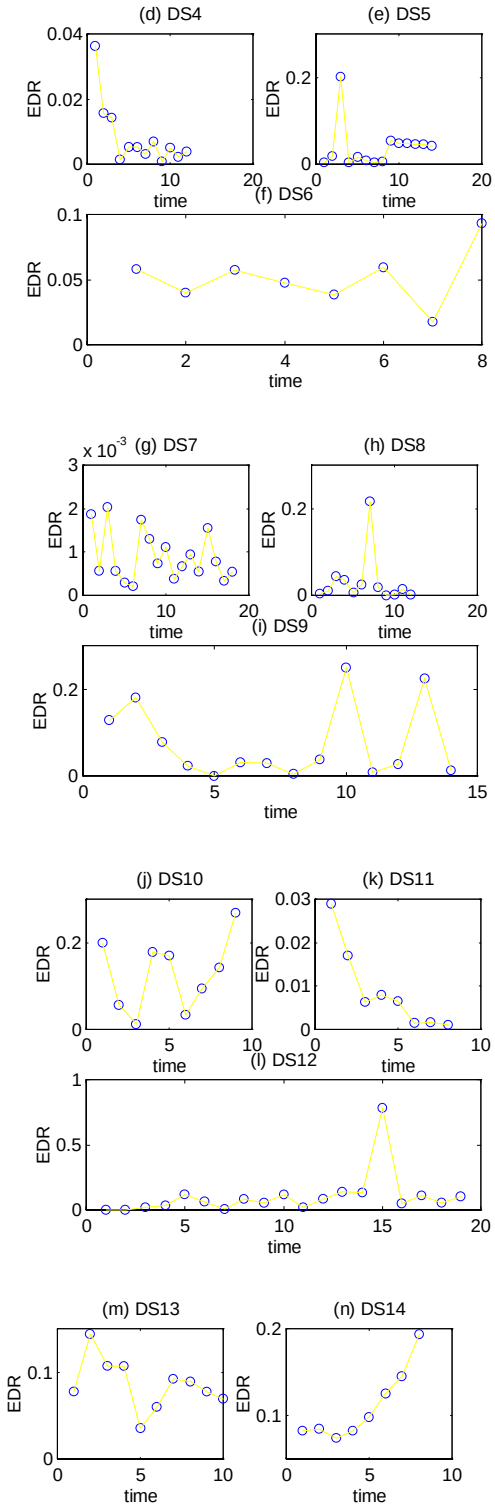


Fig 1. Variation of FDR with time.

Table 1: Summary of real data sets studied.

Data Set	*Reference*	*No. of Faults Detected*	*Observation Period*	*Software Project/Program Descriptions and Characteristics*
DS 1	[10]	328	19 weeks	PL/I Application Program, Execution Time: 47.65 CPU hours, Software Size: 1,317KLOC
DS 2	[12]	136	21 weeks	Real-Time Command and Control Application (System T1), Execution Time: 25.3 CPU hours, 9 Programmers, Software Size: 21,700 LOC
DS 3	[11]	227	38 weeks	Execution Time: 2456 CPU hours
DS 4	[21]	73	18 days	PL/I Assembler Language,418 Executions, Software Size: 50,000 LOC
DS 5	[28]	137	25 days	418 Executions
DS 6	[15]	107	15 days	F11-D Fortran Program, Execution Time: 226.11 Seconds of CPU Time, Software Size: 3~4KLOC
DS 7	[5]	32	11 weeks	System T38
DS 8	[5]	77	19 weeks	System T39
DS 9	[25]	86	22 days	Execution Time: 93 CPU hours
DS10	[17]	3,207	13months	240,000 LOC
DS11	[29]	27	27 days	Compiler Project, Software Size: 1,000 LOC
DS12	[14]	211	32 weeks	Switching System TROPICO-4096, Software Size: 300,000 LOC
DS13	[10]	46	21 days	Software Size: 40,000 LOC
DS14	[26]	198	16 days	Real-Time Control System Consisting of About 90 Modules. Software Size: 1 KLOC of PL/1, Fortran, and Assembly languages.

3. Software Reliability Growth Modeling

Since the fault detection task is performed by programmers or computers after coding, they will analyze the source code or the results of object code executions. From the detailed discussions in subsection 2.B.2, it suggests that we can use a time-dependent coefficient to replace the *constant* FDR [20]. In order to interpret the observed results, we assume that the FDR is a function of $m(t)$. That is, there exist some relationships between the number of initial faults, the number of detected faults and the fault detection rate. From Eq. (1), making some rearrangements, the fault detection rate per remaining error at testing time t is described as following and it represents the *detectability* of an error for the current error content [3]: $d(t) = \frac{dm(t)}{dt}/(a - m(t)) = r(t) \times w(t)$ (15)

that is, $r(t) = \frac{d(t)}{w(t)}$. Eq. (15) implies that $d(t) \propto r(t)$, i.e. $r(t) \uparrow$, $d(t) \uparrow$and $r(t) \downarrow$, $d(t) \downarrow$. It also indicates that the fault detection rate per remaining error is a function of the current testing- effort expenditures $w(t)$. Here we can view the $d(t)$ as a software reliability growth index and efficiency of testing. Besides, since most software reliability models assume that $w(t)$ is a constant or even do not consider the testing-effort, and equally they also set the value of r as a constant rate [5, 10-11, 16-17]. Therefore, with the above assumptions, we can get $d(t)$=constant which indicates that this model has a homogeneous fault

detection rate. Otherwise, Eq. (15) should give us a more precise description about the behavior of $d(t)$.

Discussion 1: $r(t)$ is at a steady state with time t.

***Case* :** $r(t) = r$ (16)

The above equation describes that all faults are equally exposed during testing. Under this assumption, it means that the fault detection rate per unit testing-effort is in a steady-state. Hence, substitute Eq. (16) into Eq. (1) and solve the differential equation under the boundary condition $m(0)=0$, we have [8, 18]:

$$m(t)=a\times(1- exp[-r\ (W(t)-W(0)\ )\ ]\ ) \quad (17)$$

And from Eq. (15), the fault detection rate per remaining error at testing time t is

$$d(t) = r \times w(t) \quad (18)$$

The above equation indicates that whether $d(t)$ is a homogeneous or nonhomogeneous fault detection rate is totally dominated by the current testing-effort pattern, $w(t)$.

Discussion 2: $r(t)$ is non-decreasing with time t.

***Case* I:** $r(t) = r_0 + k \times \frac{m(t)}{a}$, $k>0$ (19)

Under this assumption, we use a linear regression model to estimate the FDR. In Eq. (19), r_0 is the initial FDR and k is the *slope* (model parameter) which can be estimated from least squares. It is used to track and predict the possible increasing FDR trends. Substituting Eq. (19) into Eq. (1) and solving this differential equation, we obtain

$$m(t)=a\times(1-\frac{(r_0+k)}{r_0\times\exp[(r_0+k)\times(W(t)-W(0))]+k}) \quad (20)$$

From Eq. (15), the fault detection rate per remaining error is

$$d(t) =w(t)\times(1-\frac{k}{\exp[(r_0+k)\times(W(t)-W(0))]+k}) \quad (21)$$

and it is monotonically increasing in testing time t. That is, Eq. (21) means that Eq. (20) describes a fault detection process in which the detecability of an error increases with the progress of software testing.

***Case* II:** $r(t) = r_0 + (r_f - r_0)\frac{m(t)}{a}$, $0<r_0<r_f$ (22)

Under this assumption, in Eq. (22), r_0 is the initial FDR and r_f is the final FDR. Substituting Eq. (22) into Eq. (1), we obtain a Riccati differential equation and solve it:

$$m(t)=a\times\left(1-\frac{r_f}{r_0\times\exp[r_f\times(W(t)-W(0))]+r_f-r_0}\right),\ 0<r_0<r_f \quad (23)$$

Similarly, from Eq. (15), the fault detection rate per remaining error at testing time t is

$$d(t) =w(t)\times r_f\times\left(1-\frac{r_f-r_0}{r_0\times\exp[r_f\times(W(t)-W(0))]+r_f-r_0}\right) \quad (24)$$

and it is monotonically increasing. Similarly, Eq. (24) means that Eq. (23) describes a fault detection process in which the detecability of an error increases with the progress of software testing.

Discussion 3: $r(t)$ is non-increasing with time t.

This case describes a fact: a large number of trivial faults are easily detected in the beginning and the last few faults are difficult to detect.

***Case* I:** $r(t) = r_0 \times (1 - \frac{m(t)}{a})$ (25)

Substituting Eq. (25) into Eq. (1), the solution of this equation is given by

$$m(t)=a\times\left(1-\frac{1}{r_0\times(W(t)-W(0)+1}\right) \quad (26)$$

From Eq. (15), the fault detection rate per remaining error at testing time t is

$$d(t) = \frac{r_0\times w(t)}{r_0+(W(t)-W(0))+1} \quad (27)$$

***Case* II:** $r(t) = r_0 + k \times \frac{m(t)}{a}$, $k<0$ (28)

Substituting Eq. (28) into Eq. (1), the solution of this equation is given by

$$m(t) = a\times(1-\frac{(r_0+k)}{r_0\times\exp[(r_0+k)\times(W(t)-W(0))]+k}) \quad (29)$$

And from Eq. (15), the fault detection rate per remaining error at testing time t is

$$d(t) =w(t)\times(1-\frac{k}{\exp[(r_0+k)\times(W(t)-W(0))]+k}) \quad (30)$$

***Case* III:** $r(t) = r_0 + (r_f - r_0)\frac{m(t)}{a}$, $r_0>r_f$ (31)

Substituting Eq. (31) into Eq. (1), the solution of this equation is given by

$$m(t)= a\times\left(1-\frac{r_f}{r_0\times\exp[r_f\times(W(t)-W(0))]+r_f-r_0}\right) \quad (32)$$

And from Eq. (15), the fault detection rate per remaining error at testing time t is

$$d(t) =w(t)\times r_f\times\left(1-\frac{r_f-r_0}{r_0\times\exp[r_f\times(W(t)-W(0))]+r_f-r_0}\right) \quad (33)$$

In order to check the validity of the proposed model and make a fair comparison with other existing SRGMs, we divide the above equations (i.e. Eq. (16), (19), (22), (25) , (28), and (31)) into four groups:

1. *GROUP A*: $r(t) = r$.
2. *GROUP B*: $r(t) = r_0 + (r_f - r_0)\frac{m(t)}{a}$, $0<r_0<r_f$ or $0<r_f<r_0$.
3. *GROUP C*: $r(t) = r_0 + k \times \frac{m(t)}{a}$, $k>0$ or $k<0$.
4. *GROUP D*: $r(t) = r_0 \times (1 - \frac{m(t)}{a})$, $r_0>0$.

Furthermore, if we want to accept/reject some SRGMs under a specific software development experiment, we must have some clear criteria for evaluation and comparison among the acceptable models.The comparison criteria for estimation are described as follows:

(1) The Accuracy of Estimation [5, 8-9] (AE)= $\left|\frac{M_a-m}{M_a}\right|$ (34)

where M_a is the actual cumulative number of detected faults during the test and after the test, and m is the estimated number of initial faults.

(2) The Mean of Square fitting Faults (MSF)= $\frac{\sum_{i=1}^{k}[m(t_i)-m_i]^2}{k}$ (35)

A smaller MSF indicates fewer number of fitting faults and better performance.

When completing the derivation of software reliability growth models, we also can get some useful quantitative measures in order to assist in determining the number of residual faults and the probability of software system survivability for software developers/testers. They are (1) *Maximum faults* (MF), i.e. the total number of initial faults, $m(\infty)$; (2) *Remaining faults* (RF) in the system at testing time t, i.e., $m(\infty) - m(t)$ which is an important indicator of the software reliability and very useful for planning maintenance activities and discussions; (3) *Time-Interval Between Software Failures* (TBSF), and (4) *Software Reliability* (SR) [8, 18].

4. Experimental Results

First Data Set

The first set of real data to be analyzed came from Ohba [10]. The system is a PL/I database application software and the size of software is approximately 1,317,000 lines of code (LOC). During the testing period of nineteen weeks, 47.65 CPU hours were consumed and about 328 software faults were removed. The original data report gives that the total cumulative number of detected faults after a long period of testing is 358 faults. The parameters α, β, and m of the Weibull-type testing effort function in Eq. (7), (8), and (9), and N, A, and α of the Logistic testing effort function in Eq. (10) can be derived by using the method of *Maximum Likelihood Estimation* and *Least Squares Estimation*. Similarly, the other parameters a, r_0 , r_f and k of the mean value function can also be solved numerically. Fig. 2 plots the fitting of the estimated testing effort by using Eq. (7), (8), (9), and (10). Table 2 summarizes the estimated parameters for different testing-effort functions, mean value function, and the comparison criteria. We can find that our proposed software reliability growth function fits pretty well at the 5% level of significance through the *Kolmogorov-Smirnov* goodness-of-fit. Fig. 3(a)-(d) graphically shows the actual (observed) and the fitted number of software faults, according to different groups in Table 2. From Table 2, both *MSF* and *AE* in *Group B* are less than those in other groups/existing SRGMs and it is conceivable that *Group B* has a better goodness-of-fit. In fact, for *Group B*, it uses two parameters to interpret the various fault detection patterns instead of the traditional assumption of constant fault detection rate, and indeed has a very good performance. But we must point out that by adding an extra parameter in modeling the fault detection phenomenon, the estimation becomes more difficult because more numerical calculations are involved. However, if very high reliability is needed in some critical applications, such as very large scale commercial software or space shuttle software, we may not avoid the extra complex numerical operations. Besides, from Fig. 3(b), we see that these continuous curves of estimated mean value function have an inflection point. That is, they show S-shaped behavior due to $r_f >> r_0$ in *Group B* of Table 2. The derived software reliability model under such assumption (i.e. Eq. (22)) has been used by Yamada [10-11, 20]. Finally, we can conclude that the combined model (*Group B*) of incorporating testing-effort function and time-variable fault detection actually fits the data set satisfactorily in this experiment.

Table 2: Summary of model parameters and comparisons for the first data set.

Model *(Group A)*	***a***	***r***		***MSF***	***AE***
Eq. (17) with Logistic function	394.076	0.0427223		118.29	10.06
Eq. (17) with Weibull function	565.35	0.0196597		122.09	57.91
Eq. (17) with Rayleigh function	459.08	0.0273367		268.42	28.23
Eq. (17) with Exponential	828.252	0.0117836		140.66	131.35
Model *(Group B)*	***a***	r_0	r_f	***MSF***	***AE***
Eq. (23) with Logistic function	337.41	0.018962	0.113343	163.095	5.75
Eq. (23) with Weibull function	345.686	0.0125642	0.106949	91.0226	3.43
Eq. (23) with Rayleigh function	371.438	0.0137198	0.08050	158.918	3.75
Eq. (23) with Exponential	352.521	0.0108348	0.10819	83.998	1.53
Model *(Group C)*	***a***	r_0	***k***	***MSF***	***AE***
Eq. (29) with Logistic function	430.662	0.0409427	-0.014653	103.03	20.11
Eq. (20) with Weibull function	385.39	0.0229036	0.0393828	87.5831	7.65
Eq. (20) with Rayleigh function	379.947	0.0239006	0.0385439	406.71	6.13
Eq. (20) with Exponential	385.179	0.0180857	0.0547021	83.3452	7.69
Model *(Group D)*	***a***	r_0		***MSF***	***AE***
Eq. (26) with Logistic function	582.538	0.0308452		96.9321	62.72
Eq. (26) with Weibull function	958.718	0.0118215		124.399	167.79
Eq. (26) with Rayleigh function	702.693	0.0191208		247.84	96.09
Eq. (26) with Exponential	1225.66	0.0082272		169.72	242.36
Existing SRGMs	***a***	***r***		***MSF***	***AE***
G-O Model [10]	562.8	*		157.75	56.98
Inflection S-Shaped Model [10]	389.1	0.0935493		133.53	8.69
Delayed S-Shaped Model [8]	374.05	0.197651		168.67	4.48
Exponential Model [10]	455.371	0.0267368		206.93	27.09
HGDM [9]	387.71	*		138.12	8.30
Logarithmic Poisson Model[9]	NA	*		171.23	*

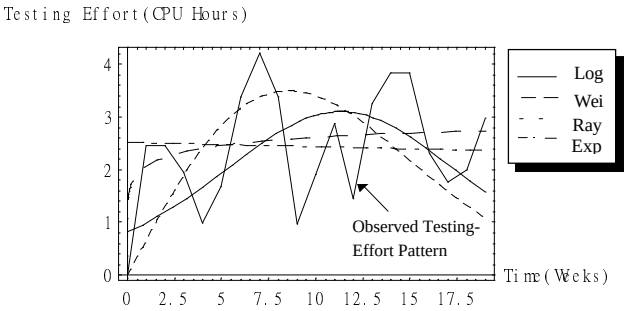
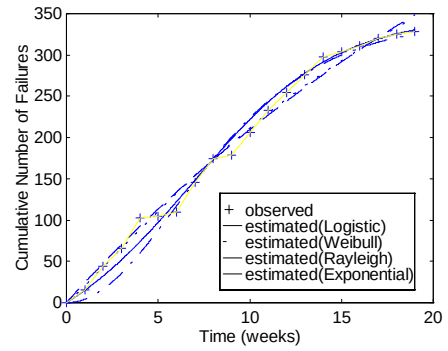
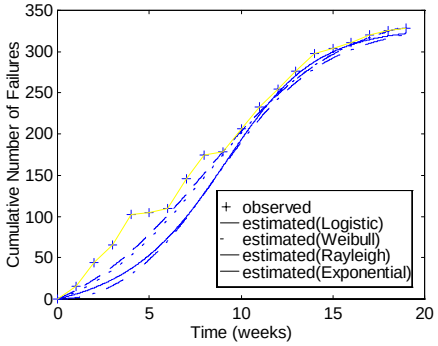


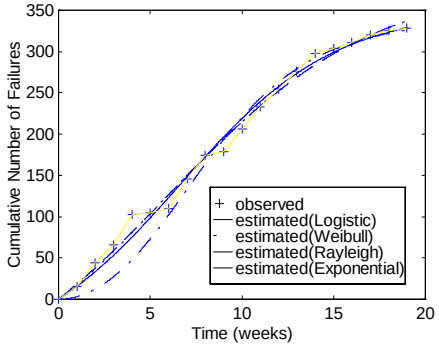
Fig. 2: Plot of observed/estimated testing-effort vs. time.



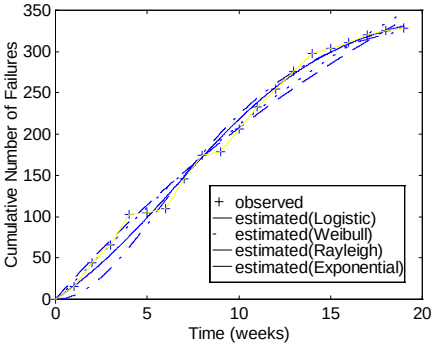
GROUP A



GROUP B



GROUP C



GROUP D

Fig. 3: Cumulative number of observed/ estimated failures vs. time.

Second Data Set

The second set of real data is the pattern of discovery of errors by Thoma in [26]. The debugging time and the number of detected faults per day are reported. The cumulative number of discovered faults up to twenty-two days is 86 and the total consumed debugging times is 93 CPU hours. All debugging data are used in this experiment. Similarly, we can estimate each parameter by *MLE* and *LSE* in the proposed SRGM and they are shown in Table 3. Fig. 4 plots the fitting of the estimated testing effort by using Eq. (8), (9), and (10). Fig. 5(a)-(d) graphically shows the fitted number of software failures as compared with the observed error data, according to the different groups in Table 3. We observed that the fitness of mean value function to actual failure data is still good. From Table 3, we can clearly see that the *MSFs* for *Group C* and *Group D* are less than those of other groups/existing SRGMs and the results of measures indicate that they perform better. Similarly, from Fig. 5(b), we see that these continuous curves of estimated mean value function have an inflection point. They also show S-shaped behavior due to $r_f > r_0$ in *Group B* of Table 3. From the above discussion, we know that the combined model (*Group C & D*) of incorporating testing-effort function and time-variable fault detection fits this data set better than others.

Table 3: Summary of model parameters and comparisons for the second data set.

Model *(Group A)*	***a***	***r***		***MSF***
Eq. (17) with Logistic function	88.8931	0.0390591		25.2279
Eq. (17) with Weibull function	87.0318	0.0345417		7.772
Eq. (17) with Rayleigh function	86.1616	0.0359624		3.91643
Model *(Group B)*	***a***	r_0	r_f	***MSF***
Eq. (23) with Logistic function	89.4528	0.0188499	0.0543846	14.06603
Eq. (23) with Weibull function	87.3126	0.017449	0.0522258	18.956772
Eq. (23) with Rayleigh function	87.3472	0.0177506	0.0515699	20.4568
Model *(Group C)*	***a***	r_0	***k***	***MSF***
Eq. (29) with Logistic function	97.5332	0.0472247	-0.0385523	7.354363
Eq. (29) with Weibull function	97.6841	0.0360678	-0.0227224	6.5909
Eq. (29) with Rayleigh function	112.182	0.0335812	-0.0335811	6.60318
Model *(Group D)*	***a***	r_0		***MSF***
Eq. (26) with Logistic function	106.1	0.0437178		7.33727
Eq. (26) with Weibull function	114.52	0.0314776		6.36531
Eq. (26) with Rayleigh function	112.183	0.0335812		6.60318
Existing SRGMs	***a***	***r***		***MSF***
G-O Model	137.072	0.0515445		25.33
Delay S-Shaped Model [8]	88.6533	0.228148		6.31268
HGDM [26]	88.3	*		33.6812

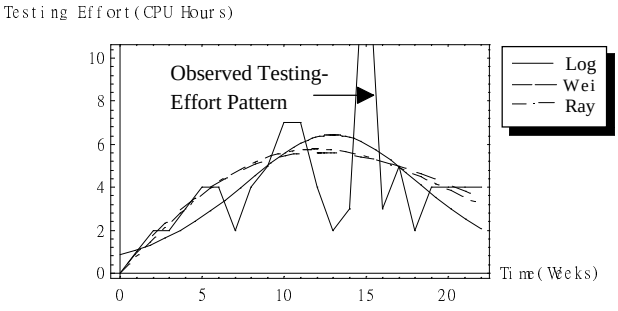
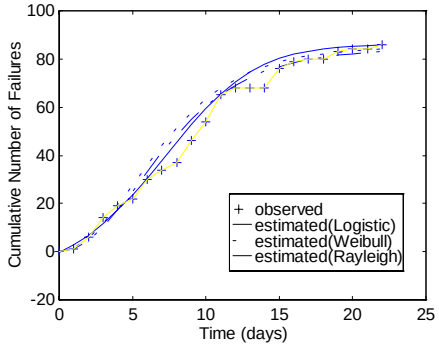
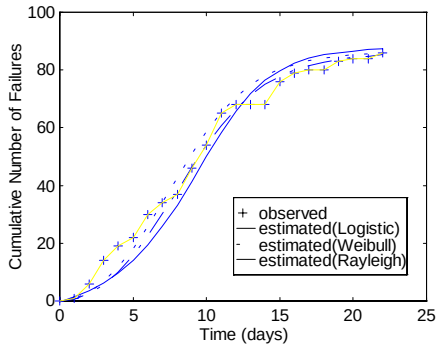


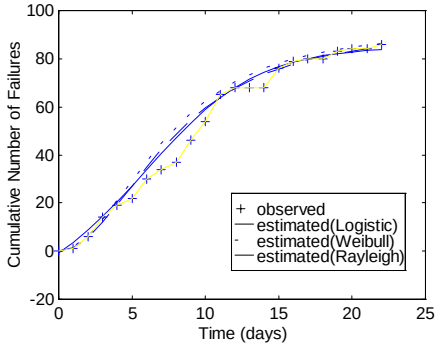
Fig. 4: Plot of observed/estimated testing-effort vs. time.



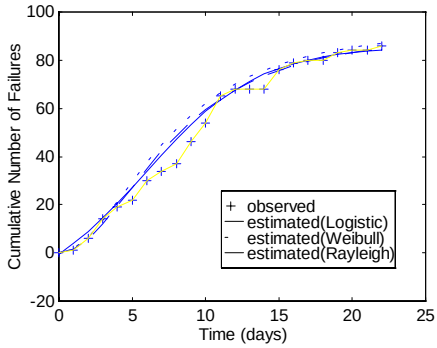
GROUP A



GROUP B



GROUP C



GROUP D

Fig. 5: Cumulative number of observed/ estimated failures vs. time.

Third Data Set

The third set of real data in this paper is the System T1 data of the Rome Air Development Center (RADC) projects in [12] and the failure data is generally of the best quality. The number of object instructions for the System T1 which is used for a real-time command and control application was 21,700 and it was developed by Bell Laboratories. It took twenty-one weeks and nine programmers to complete the test. The intervals are measured by the wall clock time, which is proportional to execution time. Similarly, the parameters in the proposed SRGM under the assumptions of different possible trends are shown in Table 4. Furthermore, we know that the proposed mean value function based on NHPP adequately fits the actual error data at a 5% level of significance through the *Kolmogorov-Smirnov* goodness-of-fit test. Fig. 6 plots the fitting of the estimated testing-effort by using Eq. (7), (8), and (10). Fig. 7(a)-(d) plots the estimated/observed number of failures vs. time. It can be seen that for this data set, *Group B* has a better goodness-of-fit than other groups and the existing SRGMs. From Fig. 7(b), we also see that these continuous curves of estimated mean value function still have inflection points. Hence, they are S-shaped. The reason is $r_f >> r_0$ in *Group B* of Table 4. Note, however, that *Group C* or *D* is the next advisable option if we want concurrently a good estimation and a simpler model. From the above discussion, we know that the combined model (*Group B*) of incorporating testing-effort function and time-variable fault detection fit this data set better than others. In fact, in this software project, we can see that if we use a constant fault detection rate model instead of a time-variable model in order to obviate complex numerical operations, we still can get a reasonable prediction in estimating the number of software faults.

Table 4: Summary of model parameters and comparisons for the third data set.

Model *(Group A)*	***a***	***r***		***MSF***
Eq. (17) with Logistic function	138.026	0.145098		62.41
Eq. (17) with Rayleigh function	866.94	0.00962		89.2409
Model *(Group B)*	***a***	r_0	r_f	***MSF***
Eq. (23) with Logistic function	137.759	0.0502167	0.359256	144.6442
Eq. (23) with Rayleigh function	150.047	0.013763	0.322236	12.137
Eq. (23) with Exponential	187.537	0.00088	0.166756	19.73719
Model *(Group C)*	***a***	r_0	***k***	***MSF***
Eq. (29) with Logistic function	142.567	0.14881	-0.0450675	53.4266
Eq. (20) with Rayleigh function	156.715	0.0183746	0.25801	10.9726
Eq. (20) with Exponential	173.064	0.000048	0.194059	48.5971
Model *(Group D)*	***a***	r_0		***MSF***
Eq. (26) with Logistic function	164.106	0.169151		38.121
Eq. (26) with Rayleigh function	1543.47	0.00546049		89.7666
Existing SRGMs	***a***	***r***		***MSF***
Exponential Model [5]	137.2	0.156		3019.66
G-O Model	142.32	0.1246		2438.3
Delayed S-Shaped Model [8]	237.196	0.0963446		245.246

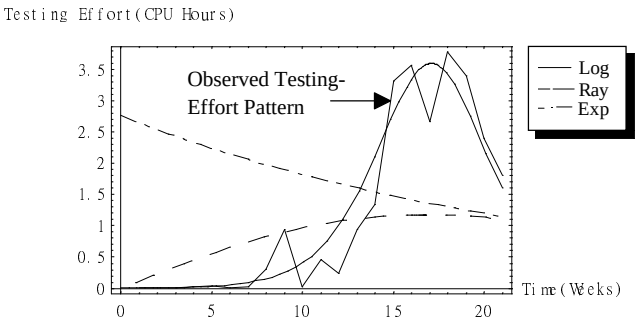
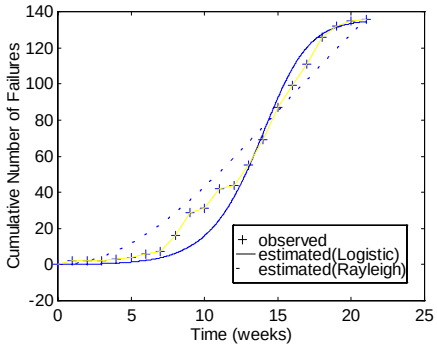
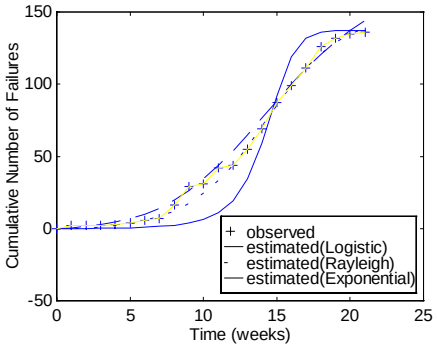


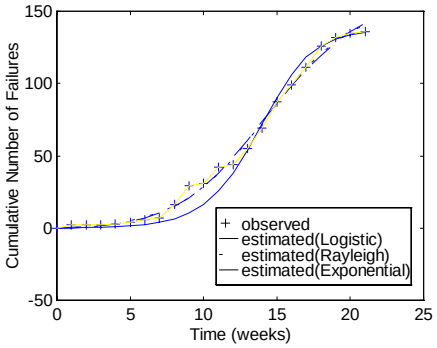
Fig. 6: Plot of observed/estimated testing-effort vs. time.



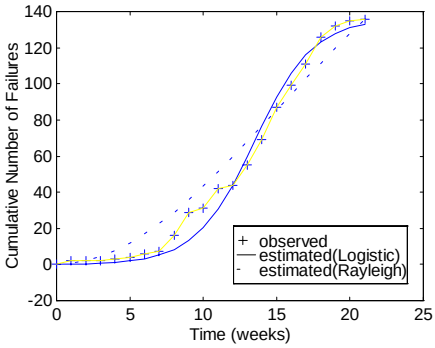
GROUP A



GROUP B



GROUP C



GROUP D

Fig. 7: Cumulative number of observed/ estimated failures vs. time.

5. Applications of Testing-Effort Control and Management

After coding is completed, software testing is a necessary but expensive process. Once the obvious and easy-to-detect faults are removed from a new computer software package, the computer company will need to determine when to stop the testing and make a software risk evaluation. If the results meet their requirements, the company will decorate and declare that this software package is ready for releasing. Hence, adequately

adjusting some specific parameters of a SRGM in the proper time interval can help us to speedup getting the desired solution.

A. Software Risk Assessment

Some people view the risk definition in Webster's New Universal Unabridged Dictionary as *risk exposure* which is equal to [*Probability of an Unsatisfactory Outcome*]× [*Loss if the Outcome is Unsatisfactory*] [23-24, 30]. Based on this definition, we can see that the risk exposure is mainly dominated by the probability of an unsatisfactory outcome. Therefore, we can map such idea into the software reliability analysis. We define:

$$R_{isk}(t) \equiv 1 - \frac{m(t)}{m(\infty)} \tag{36}$$

$R_{isk}(t)$ can be used to evaluate the risk caused by the remaining failure patterns which still exist after the testing phase. Hence, if we can accurately evaluate the risk of the tested software, then another other useful index, the operational quality, is available. The *Operational Quality Index* is defined as: the degree to which a software is free of remaining faults and this quality index is very important for a widely distributed commercial software [23-24, 30] .

$$Q(t)=(1\text{-}R_{isk}(t))\times 100\% \tag{37}$$

B. Testing-Effort Control Problem

In order to possess the lowest risk and achieve a given operational quality at a specified time, we can use the software reliability growth model to estimate/control the extra testing effort. The major problem is how to estimate the number of extra faults $\delta m'(t')$ which have to be found [3-4]. Let us consider the following scenario:

1. Due to economic/beneficial considerations, software testing/debugging will be eventually terminated at a specified time point, T_2.
2. Based on the software reliability growth model selected by software developers or test teams, the expected number of initial faults, a, in this software system is estimated at time T_1 ($0<T_1<T_2$).
3. By applying the estimated parameters into the software reliability model, the test teams can predict the cumulative number of faults at time T_2 and the $R_{isk}(T_2)$. Fortunately the estimated value of $R_{isk}(T_2)$ may sometimes already satisfy the company's desired goal or has reached the acceptance level. But if not, in order to meet the requirements (i.e. $R'_{isk}(T_2)$, where $R'_{isk}(T_2)<R_{isk}(T_2)$), the decision-maker must ask these test personnel to detect extra faults $\delta m'(T_2\text{-}T_1)$ during the time interval $T_2\text{-}T_1$.

Considering the above statements, if we know that $R'_{isk}(T_2) = 1 - \frac{m'(t_2)}{m(\infty)} = 1 - \frac{a^*}{m(t)}$ and a^* $(a^* > m(T_2))$ is the goal number of detected faults at time T_2, then

$$a^* = m(T_1) + \delta m'(T_2 - T_1),\ a^* \geq m(T_2) \tag{38}$$

$$= a \times (1 - \exp[-r \times (W(T_1) + W'(T_2 - T_1) - W(0))])$$
$$= a \times (1 - \exp[-r \times W'(T_2 - T_1)] \times \exp[-r \times (W(T_1) - W(0))])$$

where $m(T_1)$ is the cumulative number of faults detected at time T_1 and $\delta m'(T_2\text{-}T_1)$ is the extra faults needed to be detected in order to reach the desired goal at time T_2 ($\delta m'(T_2\text{-}T_1)>\delta m(T_2\text{-}T_1)$).

Rearranging the above equation, we have

$$a^* - m(T_1) = \delta m'(T_2 - T_1) = (a - m(T_1)) \times (1 - \exp[-r \times W'(T_2 - T_1)])$$

$$a - a^* = (a - m(T_1)) \times \exp[-r \times (W'(T_2 - T_1))] \tag{39}$$

where $W'(T_2\text{-}T_1)=\int_{T_1}^{T_2} w'(t)=W'(T_2)\text{-}W'(T_1)$

$$= \frac{N}{1+Ae^{-\alpha^* T_2}} - \frac{N}{1+Ae^{-\alpha^* T_1}} \tag{40}$$

Substituting Eq. (40) into Eq. (39), we get

$$\frac{N}{1+Ae^{-\alpha^* T_2}} - \frac{N}{1+Ae^{-\alpha^* T_1}} = -\frac{1}{r}\ln\left(1 - \frac{a^* - m(T_1)}{a - m(T_1)}\right) \tag{41}$$

The modified testing-effort function $W'(T_2\text{-}T_1)$ during the time interval $(T_1, T_2]$ can be controlled by using α^*, the modified consumption rate of testing-effort expenditures which satisfy Eq. (41) and it can be solved numerically. For example, because of the limitations of space, we only use Eq. (17) with Logistic testing-effort function as the estimated value function for a software development project. In fact, it is compact and easy enough to apply. The other models we proposed in this paper also can be applied similarly based on the same procedure. In the first data set, we set T_1=19 and T_2=30, then m(19)=330.472 and m(30)=347.801, respectively. If the desired operational quality index is larger than Q(30)=88.25 %, then we have the following four cases:

- Case [1]:α^*=347.801, Q(30)=88.25% (Originally).
- Case [2]:α^*=350, the desired operational quality Q(30)=88.81%.
- Case [3]:α^*=355, the desired operational quality Q(30)=90.08%.
- Case [4]:α^*=360, the desired operational quality Q(30)=91.35%.

Hence, under the requirements of Case [1]-[4] and using Eq. (38)-(41), the modified expenditure rate α^* for the first data set is estimated as 0.226337, 0.0689997, 0.0824989 and 0.0996481 respectively. It means that α^* can be used to satisfy Eq. (38) and Eq. (40) during time interval $(T_1, T_2]$ and to achieve the desired operational quality. Fig. 8 shows the modified testing-effort function for this data set.

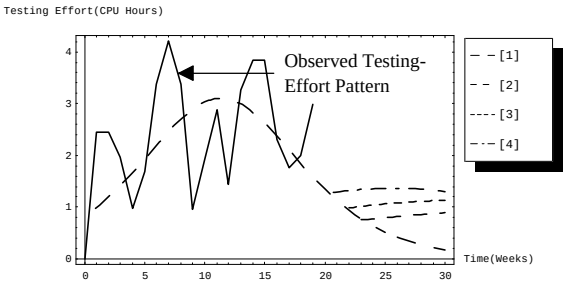


Fig. 8: Modified testing-effort function vs. time.

6. Conclusions and Future Works

A good software reliability growth model should be compact as well as simple enough and the required failure data is easy to collect. In addition, the model should be able to estimate/compute useful numerical information . Hence, it is essential that researchers continue to develope or improve existing software reliability models so that they are easy to use, not failure-data resource intensive, and can be comprehended by programmers or software engineers readily. This paper has focused primarily on offering a reasonable parametric decomposition method to modeling software reliability and describing each important factor of a SRGM in further detail, especially in the fault detection rate which is usually used to measure the expected rates from the historical records of other similar software projects and plan the related checking activities. We use three real case studies to illustrate the analytic approaches on how the models can be applied and provide enough evidences on supporting our arguments. Consequently, from the analysis of the data sets and our experiences, we deduce some conclusions as follow:

1. Fault detection rate is a metric which is used to indicate a trend. It decreases when the software has been used and tested repeatedly. We also find that the rate may increase if the testing techniques/ requirements are changed, or new faults are introduced when performing corrective actions. From our study in [18], it indicates that the first data set may have an imperfect debugging phenomenon. It really reflects the characteristic of increasing FDR.
2. In section 3, we ever used an extra parameter to describe the fault detection/removal process, such as *k*, the *slope parameter* in Eq. (19) or r_f, the final *fault detection rate* in Eq. (22). Just as we stated before: although adding a parameter in modeling the fault detection phenomenon, the estimation will become more difficult since more numerical operations are involved. But this decomposition approach does offer a better goodness-of-fit and give results with acceptable accuracy from the experiments provided in Table 2-4 when the combination includes either a well fitting testing-effort function or an adequate function of fault detection rate. The relative errors of different data sets after testing phase are given in Table 5. We find that the sample size offered by original data sets is sufficient to predict the future failure phenomenon using our methods. In fact, in order to assess the fault-prediction capability of the proposed model, we should not only compute the relative error for these data sets, but also need to use more criteria such as *u*-plot, prequential likelihood ration, ..etc. in the future.
3. Current ly we know that no one single model has been shown to be sufficiently trustworthy in all applications. If a model is used in practice, it means that this model usually obtain relatively accurate measurements of software reliability in most cases. In this case, we definitely should own more different categories of failure data sets to verify such a software reliability model and to support the conclusions we made. Due to confidential or proprietary reasons, the real data sets from industrial organizations or research institutions are hard to obtain in recent years. Presently we are collecting a new failure data set of software failure from a local banking information management system which will include the OS version, wall-clock time of incidence of each failure, failure identification number and type, testing-effort expenditures, failure impact, failure location, failure severity, ...etc. The source code is an AP control program written in about 450KLOC of COBOL language. Issues of how we collect the valuable data failure set, how we adopt *data-smoothing* procedures, how we use the techniques described in section 3, how we evaluate the data stability of software reliability growth models we proposed, and how we assess the accuracy/performance of fault prediction, will be addressed. We plan to study the *Failure Physics* and present the censored data set in the future.

Table 5: Comparisons of predictive errors

Model *(Group B)*	*RE* (DS1)	*RE* (DS2)	*RE* (DS3)
Eq. (22) with Logistic function	-0.00258142	0.0909459	0.0122385
Eq. (22) with Weibull function	0.00798354	-0.00487272	0.0257411
Eq. (22) with Rayleigh function	0.015897	-0.0909459	0.0244519
Eq. (22) with Exponential	0.0126933	0.00213582	0.01569224
Model *(Group C)*	*RE* (DS1)	*RE* (DS2)	*RE* (DS3)
Eq. (29) with Logistic function	0.00236482	-0.0235028	-0.00530852
Eq. (20) with Weibull function	0.0278819	-0.0123287	-0.0225587
Eq. (20) with Rayleigh function	-0.0143296	-0.0116224	0.0340336
Eq. (20) with Exponential	0.0279026	-0.0148521	0.01256871
Model *(Group D)*	*RE* (DS1)	*RE* (DS2)	*RE* (DS3)
Eq. (26) with Logistic function	0.00945084	-0.0217074	-0.0211915
Eq. (26) with Weibull function	0.0585257	-0.00625291	-0.00148956
Eq. (26) with Rayleigh function	-0.00744914	-0.0123181	-0.0007975
Eq. (26) with Exponential	0.0585257	0.0364782	0.00142456

Acknowledgment

We would like to express our gratitude for the support of the National Science Council, Taiwan, R.O.C. and the Taiwan Power Company, under Grant NSC 87-TPC-002-017. The authors thank the referees for their suggestions and comments in revising the original draft.

References

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

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

[3] S. Yamada, and H. Ohtera, "Software Reliability Growth Models for Testing Effort Control, " *European Journal of Operational Research*, pp. 343-349, 1990.

[4] S. Yamada, H. Ohtera, and H. Narihisa, "A Testing-Effort Dependent Software Reliability Model and Its application, " *Microelectronics and Reliability*, Vol. 27, No. 3, pp. 507-522, 1987.

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

[6] K. Pillai and V. S. Sukumaran Nair, "A Model for Software Development Effort and Cost Estimation, " *IEEE Trans. on Software Engineering*, Vol. 23, No. 8, August 1997.

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

[8] C. Y. Huang and S. Y. Kuo, "Analysis of a Software Reliability Growth Model with Logistic Testing-Effort Function," *Proc. of the 8th International Symposium on Software Reliability Engineering*, pp. 378-388, Nov. 1997, Albuquerque, New Mexico.

[9] R. H. Hou, S. Y. Kuo, and Y. P. Chang, "Applying Various Learning Curves to Hyper-Geometric Distribution Software Reliability Growth Model, "*Proceedings of the 5th International Symposium on Software Reliability Engineering*, pp. 7-16, Nov. 1994, Monterey, California.

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

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

[12] J. D. Musa, *Software Reliability Data*, Report and Data Base Available from Data and Analysis Center for Software, Rome Air Development Center (RADC), Rome, NY.

[13] S. Yamada, S. Osaki, and H. Narihisa, "A Fault Detection Rate Theory for Software Reliability Growth Models, " *Trans. IECE Japan*, E-68, No. 5, pp. 292-296, May 1985.

[14] Martini, M. R., and de Souza, J.M., "Reliability Assessment of Computer Systems Design," *Microelectronics Reliability*, vol. 31, no. 2/3, 1991, pp. 237-244.

[15] Moranda, Paul B., "A Comparison of Software Error-Rate Models, " *Texas Conference on Computing*, 1975.

[16] A. L. Goel and K. Okumoto, "Time-Dependent Fault Detection Rate Model for Software Reliability and Other Performance Measures, " *IEEE Trans. on Reliability*, Vol. R-28, No. 3, pp. 206-211, 1979.

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

[18] C. Y. Huang and S. Y. Kuo, " Effort-Index Based Software Reliability Growth Models and Their Comparisons," *Technical Report*, Dept. Electrical Engineering, National Taiwan University, Taipei, Taiwan, 1997.

[19] S. Yamada, J. Hishitani, and S. Osaki, "Software Reliability Growth Modeling: Models and Applications, " *IEEE Trans. on Software Engineering*, Vol. SE-11, pp.1431- 1437, 1985.

[20] S. Bittanti, P. Bolzern, E. Pedrotti, M. Pozzi and R. Scattolini, "A Flexible Modeling Approach for Software Reliability Growth, " *Software Reliability Modeling and Identification*, Springer-Verlag, Berlin, pp. 101-140, 1988.

[21] 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.

[22] Y. K. Malaiya, A. V. Mayrhauser and P. K. Srimani, "An Examination of Fault Exposure Ratio, " *IEEE Trans. on Software Engineering*, vol. 19, pp. 1087-1094, 1993.

[23] T. Keller and , N. F. Schneidewind, " Successful Application of Software Reliability Engineering for the NASA Space Shuttle, " *Proc. of the 8th International Symposium on Software Reliability Engineering, Case Studies*, pp. 71-82, November 1997, Albuquerque, New Mexico, U.S.A.

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

[25] 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. on Software Engineering*, vol. 15, No. 3, pp. 345-355, March 1989.

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

[27] L. H. Putnam, " A General Empirical Solution to the Macro Software Sizing and Estimating Problem, " *IEEE Trans. on Software Engineering*, Vol. 4, pp. 345-367, 1978.

[28] Y. Tohma, H. Yamamoto and R. Jacoby, " Parameter Estimation of the Hyper-Geometric Distribution for Real/ Test Data, " *Proc. of the 2nd International Symposium on Software Reliability Engineering (ISSRE'91)*, May 1991, pp. 28-34, Austin, Texas.

[29] Matsumoto, K., Inoue, K., Kikuno, T., and Torii, K., "Experimental Evaluation of Software Reliability Growth Models, " *Proc. of the 18th International Symposium on Fault Tolerant Computing*, pp. 148-153, 1988, Japan.

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

[31] R. H. Huo, S. Y. Kuo, and Y. P. Chang, " Efficient Allocation of Testing Resources for Software Module Testing Based on the Hyper-Geometric Distribution Software Reliability Growth Model, " *IEEE Trans. on Reliability*, Vol. 45, No.4, pp.541-549, Dec. 1996.