Analysis of a Software Reliability Growth Model with Logistic Testing-Effort Function

Chin-Yu Huang and Sy-Yen Kuo
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
sykuo@cc.ee.ntu.edu.tw

Ing-Yi Chen
Department of Electronic Engineering
Chung Yuan Christian University
ChungLi, Taiwan

Abstract

In this paper, we investigate a Software Reliability Growth Model (SRGM) based on the Non Homogeneous Poisson Process (NHPP) which incorporates a logistic testing-effort function. Software reliability growth models proposed in the literature incorporate the amount of testing-effort spent on software testing which can be described by an Exponential curve, a Rayleigh curve, or a Weibull curve. However, it may not be reasonable to represent the consumption curve for testing-effort only by an Exponential, a Rayleigh or a Weibull curve in various software development environments. Therefore, we will show that a logistic testing-effort function can be expressed as a software development/test effort curve and give a reasonable predictive capability for the real failure data. Parameters are estimated and experiments on three actual test/debug data sets are illustrated. The results show that the software reliability growth model with logistic testing-effort function can estimate the number of initial faults better than the model with Weibull-type consumption curve. In addition, the optimal release policy of this model based on cost-reliability criterion is discussed.

1. Introduction

A computer system consists of two major components : hardware and software. Although extensive research has been done in the area of hardware reliability, research has also been conducted to study the software reliability of computer systems since 1970. Software reliability is the probability that a given software will be functioning without failure in a given environment during a specified period of time. Hence, software reliability is a key factor in software development process and software quality. The testing phase is an important and expensive part during the software development process which includes the following four phases: specification, design, programming and test-and-debug. Many resources are consumed by a software development project. Most papers assumed that the consumption rate of testing resource expenditures during the testing phase is a constant or even do not consider such testing effort. In reality, software reliability models should be developed by incorporating different testing-effort functions. Yamada et al. [1-4] and Musa et al. [5] proposed a new and simple software reliability growth model which describes the relationship among the calendar testing, the amount of testing-effort, and the number of software errors detected. The test-effort is measured by the number of CPU hours, the number of executed test cases, and so on.

SRGMs proposed by most papers incorporate the effect of testing effort in the software reliability growth and the software development effort can be described by the traditional Rayleigh, Weibull or Exponential curve. However, in many software testing environments it is difficult to describe the testing-effort function by the above three consumption curves. In this paper, we thus will show that a logistic testing-effort function can be expressed as a software development/test effort curve. Experiments have been performed based on three real test/debug data sets. The results show that the SRGMs with a logistic testing-effort function can estimate the number of initial faults better than previous approaches.

1071-9458/97 $10.00 © 1997 IEEE

Assumptions [1-4]:

1. The error removal process follows the *Non Homogeneous Poisson Process* (NHPP).
2. The software system is subject to failures at random times caused by errors remaining in the system.
3. The mean number of errors detected in the time interval $(t, t+\Delta t]$ by the current test-effort is proportional to the mean number of remaining errors in the system.
4. The proportionality is a constant over time.
5. The consumption curve of testing effort is modeled by a logistic testing-effort function.
6. Each time a failure occurs, the error which caused it is immediately removed, and no new errors are introduced.

This paper is divided into six sections. Section 2 gives a brief description of some testing-effort functions already published in the literature and the Logistic testing-effort function. Section 3 investigates the software reliability growth model with the logistic testing-effort function. We estimate the parameters of this SRGM with the logistic testing-effort function by using the least square method, apply this model to actual software failure data, and show numerical illustrations in Section 4. Section 5 is concerned with applications of this model on the optimal release policy based on the cost-reliability criterion. Finally, Section 6 concludes this paper.

2. Testing-effort functions

2.1 Review of traditional testing-effort functions

In this section, we will briefly review some testing -effort functions. During software testing phase, it consumes much test-effort, such as man power, number of test cases, and CPU time. Traditionally, the test-effort during the testing phase and the time-dependent behavior of development effort in the software development process can be described by an Exponential, a Rayleigh or a Weibull curve, which were proposed by Yamada et al. [1, 2, 3, 4], Musa et al. [5], Putnam [6] and Kapur et al. [7, 8]. Let $W(t)$ be the cumulative amount of testing-effort expenditures in the testing time interval $(0, t]$ and $g(t)$ be the consumption rate of the testing effort expenditures. Thus, the testing-effort consumed per unit time is assumed to the proportional to the remaining amount of the testing-effort expenditures $\alpha - W(t)$. Following the general assumptions [3, 4], we can get the differential equation [1, 3]:

$$\frac{dW(t)}{dt} = g[t] \times [\alpha - W(t)],\ \alpha > 0 \qquad (1)$$

Solving the above equation, we get

$$W(t) = \alpha \times [1 - e^{\left(-\int_0^t g(x)dx\right)}] \qquad (2)$$

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

$$W(t) \equiv \int_0^t w(x)dx \qquad (3)$$

where α is the total amount of testing effort to be eventually consumed, $g(t)$ is the consumption rate of testing-effort expenditures and $w(t)$ is the current testing effort consumption at time t [3].

1. If $g(t) = \beta$, then $w(t) = \alpha\beta \exp[-\beta t]$, we have an Exponential curve and the integral forms of $w(t)$ (i.e. the cumulative testing-effort consumed in time $(0, t]$) is $W(t)=\alpha(1- \exp[-\beta t])$. (4)
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 integral forms of $w(t)$ is $W(t)=\alpha(1- \exp[-\frac{\beta}{2}t^2])$. (5)
3. And if $w(t)=\alpha\beta m\ t^{m-1} \exp[-\beta t^m]$, we have a Weibull Curve and the integral forms of $w(t)$ is $W(t)=\alpha(1-\exp[-\beta t^m])$, (6) where β is the scale parameter and m is the shape parameter.

In the Weibull-type curves (i.e. Eq. (6)), when m=1 or m=2, we obtain the exponential or the Rayleigh curve respectively and they are the special cases of the Weibull testing-effort function. In the Weibull-type curves, when m=3, 4, and 5, we can find that these testing-effort curves almost have an apparent peak Phenomenon (i.e. non-smoothly increasing and degrading consumption curve) during the software development process. That is, an extreme peak work rate will occur. This phenomenon seems not so realistic because it is not commonly used to interpret the actual software development/test process. It sometimes may not be suitable for modeling the test effort consumption curve although the Weibull function can be made to fit or approximate many distributions and represents flexible testing effort by controlling the shape parameter m. General speaking, from our study [1-4, 7, 8], the estimated value of m usually will not be larger than 2.5 in many real world applications .

2.2 Logistic testing-effort function

Since actual testing-effort data express various expenditure patterns, sometimes the testing-effort expenditures are difficult to be described by only a Exponential or Rayleigh curve. Although the Weibull -type curve can fit the data well under the general software development environment, it will have an apparent peak phenomenon when the shape parameter $m>3$. Therefore, we try to use a logistic testing-effort function which was first presented by Parr [9] instead of the Weibull type testing-effort consumption function as the testing-effort function to describe the test effort patterns during the software development process. This obtained function differs from the Weibull-type function described in the above subsection and was used to derive the form of the resource consumption curve of a project over its life cycle [9]. The logistic testing-effort function has the following form:

The cumulative testing effort consumption in time $(0, t]$ is

$$W(t) = \frac{N}{1+Ae^{-\alpha t}} \tag{7}$$

and the current testing effort consumption

$$w(t) = \frac{dW(t)}{dt} = \frac{\alpha AN e^{-\alpha t}}{(1+Ae^{-\alpha t})^2} \tag{8}$$

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. Therefore, we can see that $w(t)$ $(=\frac{\alpha AN e^{-\alpha t}}{(1+A e^{-\alpha t})^2})$ is a smooth bell-shaped function. The testing effort $w(t)$ reaches its maximum value at time

$$t_{\max} = \frac{1}{\alpha} \ln A \tag{9}$$

Compared with the Weibull-type testing-effort function in the starting point, the value of logistic testing-effort function $W(0)$ is non-zero. The divergence between the Weibull-type curve and $W(t)$ is concentrated in the earlier stages of software development where progress is often least visible and formal accounting procedures for recording the amount of testing effort applied may not have been instituted. It is possible for us to judge between these models using some statistical test of their relative ability to fit actual failure data, such as adjusting the origin and scales linearly [9].

3. Software reliability growth model

Based on the assumptions, if the number of detected errors due to the current testing-effort expenditures is proportional to the number of remaining errors, then we obtain the following differential equation [3]:

$$\frac{dm(t)}{dt} \times \frac{1}{w(t)} = r[a - m(t)], \quad a>0,\ 0<r<1 \tag{10}$$

where $m(t)$=the expected mean number of errors detected in time $(0, t]$

$w(t)$= current testing effort consumption at time t
a = the expected number of initial faults
r = error detection rate per unit testing-effort at testing time t that satisfies $r>0$.

Solving the above differential equation under the boundary conditions $m(0)=0$ (i.e., the mean value function $m(t)$ must be equal to zero at time 0), we have

$$m(t)=a(\ 1-\ exp[-r\ (W(t)-W(0))\ ]\ ) \tag{11}$$
$$=a(\ 1-\ exp[-r\ W^*(t)\ ]\ )$$

where $W^*(t)=\frac{N}{1+A\exp(-\alpha t)} - \frac{N}{1+A}$ (12)

Substituting Eq. (12) into Eq. (11), we obtain $m(t)=a(1-exp[-r\ (\frac{N}{1+A\exp(-\alpha t)} - \frac{N}{1+A})\ ]\ )$ and the failure intensity function (or instantaneous error detection rate) $\lambda(t)=\frac{dm(t)}{dt}$ $=arw(t)\ exp[-rW^*(t)\ ]$.

The expected number of errors to be detected eventually is $m(\infty) = a(1-\ exp[-r\ \frac{NA}{1+A}\ ]\ )$. If $A >> 1$, then $m(\infty) \cong a(1-\ exp[-r\ N]\ )$. It represents the expected number of undetected errors after an infinite test time $a-m(\infty) = = a \times \exp[-r\frac{NA}{1+A}] \cong a \times \exp[-rN]$. Hence, all the original errors in a software system can't be fully detected after a long time because the total amount of testing effort to be eventually consumed during the testing phase is limited to N. It requires an infinitely large amount of testing effort if all errors are to be tested/removed which is almost impossible because the test team can not devote all their efforts and resources on a software product forever. The test team has to release a software to market at the right/best time taking into the economic considerations.

4. Numerical examples

4.1 Estimation of model parameters by least square method

To validate the proposed Software Reliability Growth Model with a logistic testing-effort function in Eq.(11), experiments on three real test/debug data sets were

performed. Two most popular estimation techniques are *Maximum Likelihood Estimation (MLE)* and *Least Squares Estimation (LSE)* [5, 9, 10, 19]. The method of least squares minimizes the sum of squares of the deviations between what we actually observe/get and what we expect and the least squares estimation provides the best point estimates [10]. Therefore, we decided to fit the logistic curve and the proposed model directly on the above three data sets, using the least sum of squares criterion to give a "best fit". That is, the parameters N, A, and α of the logistic testing function in Eq. (7) and the parameters a, r given in Eq. (11) can be estimated by the method of least squares. In the method of least square sum, the evaluation formula $S1(N, A, \alpha)$ and $S2(a, r)$ are as follow:

$$\text{Minimize} \quad S1(N,A,\alpha) = \sum_{k=1}^{n} [W_k - W(t_k)]^2 \qquad (13)$$

$$\text{Minimize} \quad S2(a,r) = \sum_{k=1}^{n} [m_k - m(t_k)]^2 \qquad (14)$$

where W_k is the cumulative testing effort really consumed in time $(0, t_k]$ and $W(t_k)$is the cumulative testing effort estimated by the logistic testing function in (7). The m_k is the cumulative number of detected errors in a given time interval $(0, t_k]$ and $m(t_k)$ is the estimated cumulative number of detected errors in (11). Differentiating $S1$ ($S2$) with respect to N, A, and α (a, r) , setting the partial derivatives equal to zero and rearranging these terms, we can solve such kind of nonlinear least square problems. For example, consider the Logistic testing effort function and take the partial derivatives of $S1$ with respect to N, A and α. First we get

$$\frac{\partial S1}{\partial N} = \sum_{k=1}^{n} 2\left(W_k - \frac{N}{1+Ae^{-\alpha t}}\right)\frac{1}{1+Ae^{-\alpha t}} = 0 \qquad (15)$$

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

$$N = \frac{\sum_{k=1}^{n} \frac{W_k}{1+Ae^{-\alpha t}}}{\sum_{k=1}^{n} \frac{1}{(1+Ae^{-\alpha t})^2}} \qquad (16)$$

$$\frac{\partial S1}{\partial A} = \sum_{k=1}^{n} 2\left(W_k - \frac{N}{1+Ae^{-\alpha t}}\right)\frac{Ne^{-\alpha t}}{(1+Ae^{-\alpha t})^2} = 0 \ ; \qquad (17)$$

$$\frac{\partial S1}{\partial \alpha} = \sum_{k=1}^{n} 2\left(W_k - \frac{N}{1+Ae^{-\alpha t}}\right)\frac{NAte^{-\alpha t}}{(1+Ae^{-\alpha t})^2} = 0 \qquad (18)$$

The other parameters A and a can also be solved by substituting the least squares estimator N into Eq. (17) and (18). Besides, we choose two comparison criteria of estimation described as follow:

(1) The Accuracy of Estimation [5, 11, 18]

$$(AE) = \left|\frac{M_a - m}{M_a}\right| \qquad (19)$$

where M_a is the actual cumulative number of detected errors during the test and after the test, and m is the estimated parameter a in Eq. (11).

(2) The Mean of Square fitting Errors

$$(MSE) = \frac{\sum_{i=1}^{k} [m(t_k) - m_k]^2}{k} \qquad (20)$$

The lower MSE indicates less fitting errors and better performance [8].

4.2 Fitting model to real software data and data analysis

First Data Set

The first set of real data is from a study by Ohba [12]. The system is a PL/I database application software consisting of approximately 1,317,000 lines of code. During nineteen weeks, 47.65 CPU times were consumed and about 328 software errors were removed. Besides, the total cumulative number of detected faults after a long time of testing is 358 [11, 12]. In order to estimate the parameters N, A, and α of the logistic testing function, we fit the actual testing-effort data into Eq. (13) and solve it by using the method of least squares. These estimated parameters are:

N=54.8364, A=13.0334, α=0.226337.

The above parameters of testing effort function and the actual software error data are applied to Eq. (14). The other parameters a, r in (11) can also be solved numerically by the method of least squares for these failure data. Table 1 shows the estimated parameters of Eq.(11) and compares with the estimated initial faults m and MSE of other general models. Fig. 1(a), Fig. 1(b) and Fig. 1(c) graphically illustrate the fitting of the estimated current testing effort by using Eq. (8), Weibull function and Rayleigh function respectively. The cumulative numbers of estimated failures by Eq. (11) are:

a=394.076, r=0.0427223

In addition, substituting the estimated parameters A and α in Eq.(9), the testing effort function reaches the maximum at time t=11.3438 weeks which corresponds to $w(t)$=3.10288 CPU hours and $W(t)$=23.5107 CPU hours. Besides, the number of errors removed up to this time t_{max}

is 245.421. Fig. 1(d) shows the actual (observed) and fitted software failures versus test time. From the fit in Fig. 1(a) and the comparison criteria in Table 1, we see that the fit in Fig. 1(a) is better than that in Fig. 1(b) and Fib. (c). Hence, we can conclude that a SRGM with Logistic testing-effort function gives a better fit in this experiment .

Table 1. Comparison results for the first data set.

Model	*a*	*r*	*AE* (%)	*MSE*
Eq. (11)	394.076	0.0427223	10.06	118.29
G-O Model [11]	562.8	*	56.98	157.75
Inflection S-Shaped Model [12]	389.1	0.0935493	8.69	133.53
Delayed S-Shaped Model [11]	353.2	*	1.4	290.69
Exponential Model [12]	455.37	0.0267368	27.09	206.93
Eq. (10) with Weibull function	565.35	0.0196597	57.91	122.09
Eq. (10) with Rayleigh function	459.08	0.0273367	28.23	268.42
Logarithmic Poisson Model [11]	not exist	*	*	171.23

a = expected initial faults

r = error detection rate per unit testing effort

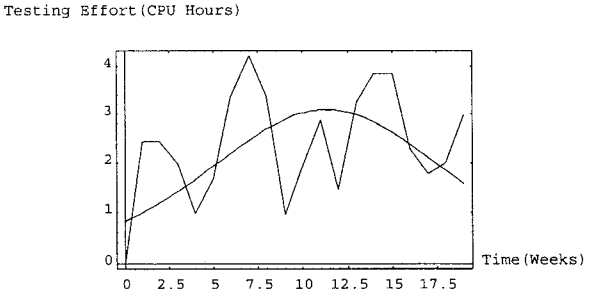

Figure 1(a). Observed/estimated current testing -effort by using logistic function vs. time.

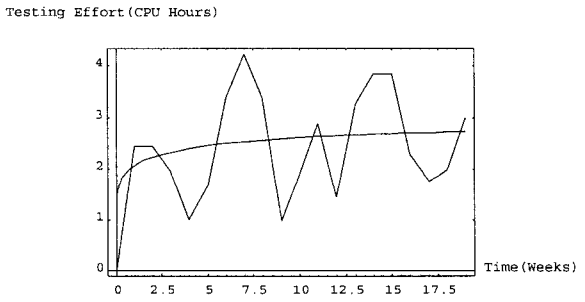

Figure 1(b). Observed/estimated current testing -effort by using weibull function vs. time.

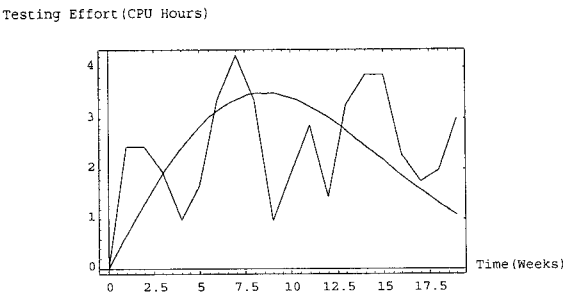

Figure 1(c). Observed/estimated current testing -effort by using rayleigh function vs. time.

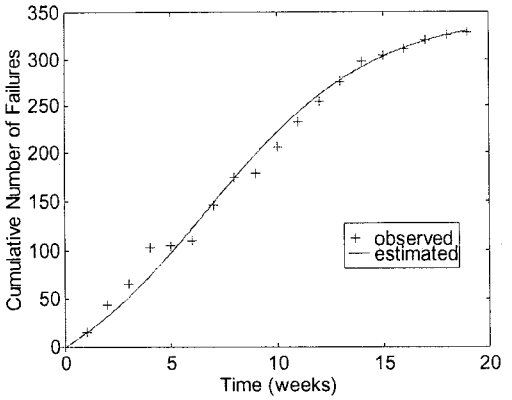

Figure 1(d). Cumulative number of observed /estimated failures vs. time.

Second Data Set

The second set of real data is the pattern of discovery of errors in the software that supported Space Shuttle flights STS2, STS3, STS4 at the Johnson Space Center. The system is also a real-time command and control application [13]. A weekly summary of software test hours and the errors of various severity discovered is given in [13]. The cumulative number of discovered faults up to thirty-eight weeks is 227. Following the same procedure described in First Data Set, the testing-effort data are applied to estimate the parameters N, A, and α of the logistic testing function described in Eq. (13) by using the method of lease squares. These estimated parameters are :

N=2828.88, A=10.5057, α=0.0988842

The above parameters of testing effort function and the actual software error data are applied to Eq. (14). The other parameters a, r in (11) can be solved numerically by the method of least squares for these failure data:

a=241.325, r=0.000907329

In addition, substituting the estimated parameters A and α in Eq. (9), the testing effort function reaches the maximum at time t=23.7846 weeks which corresponds to $w(t)$=69.9329 CPU hours and $W(t)$=1168.57 CPU hours. Besides, the number of errors removed up to this time t_{max} is 154.672. Table 2 shows the estimated parameters of Eq. (11) and compares with the estimated initial faults m and MSE of other general models. Fig. 2(a) and Fig. 2(b) plot the fitting of the estimated current testing effort by using Eq. (8) and the Rayleigh function respectively. Fig. 2(c) shows the actual (observed) and fitted software failures versus test time. From Fig. 2(a), Fig. 2(b) and Table 2, we know that the SRGM with Logistic testing effort function fits the second data set better than others in this experiment.

Table 2. Comparison results for the second data set.

Model	*a*	*r*	*MSE*
Eq. (11)	241.325	0.000907329	75.50
G-O Model [13]	597.887	0.00020988	78.87
Eq. (10) with Rayleigh function	230	0.84	15041.21

a = expected initial faults

r = error detection rate per unit testing effort

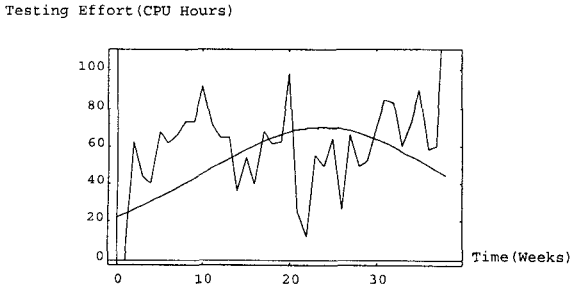

Figure 2(a). Observed/estimated current testing -effort by using logistic function vs. time.

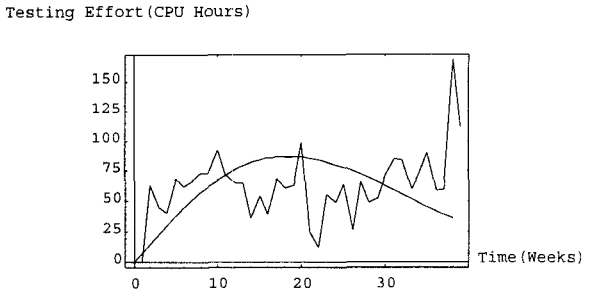

Figure 2(b). Observed/estimated current testing -effort by using rayleigh function vs. time.

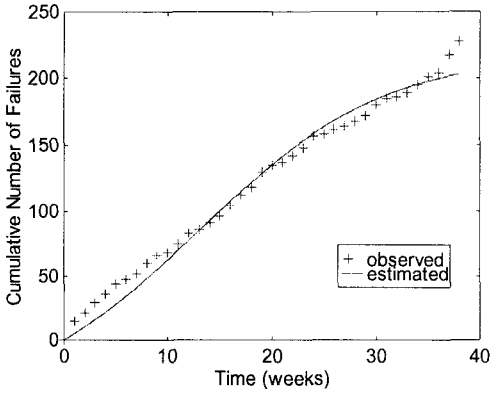

Figure 2(c). Cumulative number of observed /estimated failures vs. time.

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 [14] and the failure data is generally of the best quality. The system T1 is used for a real-time command and control application. In this case, the size of the software is approximately 21,700 object instructions. It took twenty-one weeks and nine programmers to complete the test. During the test phase, about 25.3 CPU hours were consumed and 136 software errors were removed [14].

First, we still will estimate the parameters N, A, and a of the logistic testing function in (13) by using the method of least squares. These estimated parameters are :
N=29.1095, A=4624.89, a=0.493515.
The above parameters of testing effort function and the actual software error data are applied to Eq. (14). The other parameters a, r in Eq. (11) can be solved numerically by the method of least squares for these failure data:

a=138.165, r=0.145098

In addition, substituting the estimated parameters A and a in Eq.(9), the testing effort function reaches the maximum at time t=17.1002 weeks which corresponds to $w(t)$=3.59149 CPU hours and $W(t)$=14.5484 CPU hours. Besides, the number of errors removed up to this time t_{max} is 121.343. The fit in Fig. 3(a) appears to be good. Table 3 and Fig. 3(b) show the data analysis, and the actual(observed) and fitted software failures versus test time respectively.

Table 3. Comparison results for the third data set.

Model	*a*	*r*	*MSE*
Eq. (11)	138.026	0.145098	62.41
Exponential Model [5]	137.2	0.156	3019.66

a = expected initial faults

r = error detection rate per unit testing effort

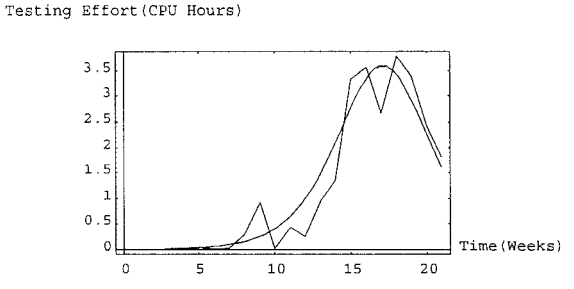

Figure 3(a). Observed/estimated current testing -effort by using logistic function vs. time.

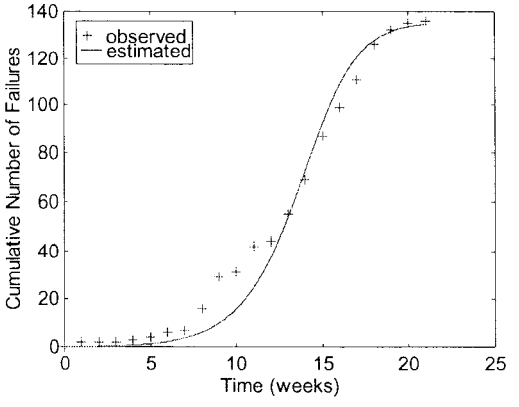

Figure 3(b). Cumulative number of observed /estimated failures vs. time.

5. Optimal release policy

5.1 Software Release Time based on Reliability Criterion

In general, the software release time problem is associated with the reliability of software system. First we discuss the release policy based on the reliability criterion. If we know that the software reliability of this computer system has reached an acceptable reliability level, then we can determine when to release this software at the right time. Okumoto and Goel [15] first dealed with the release problem considering the software cost-benefit. The conditional reliability function after the last failure occurs at time t is obtained by

$$R(t+\Delta t|t) = \exp(-[m(t+\Delta t) - m(t)])$$
$$= \exp(-m(\Delta t) \times \exp(-rW^*(t))) \quad (21)$$

Taking the logarithm on both sides of the above equation and rearranging the above equivalent, we obtain

$$\ln R = -m(\Delta t) \times \exp(-rW^*(t)) \quad (22)$$

$$\text{Thus, } W^*(t) = \frac{1}{r}\left[\ln m(\Delta t) - \ln \ln \frac{1}{R}\right] \quad (23)$$

Solving Eq. (22) and Eq. (12) we can calculate that the testing time needed to reach a desired reliability R. For example, we discuss the first real data set described in subsection 4-2. In First Data Set, N=54.8364, A=13.0334, a=0.226337, a=394.076 and r=0.0427223. Suppose this software system is desired that this testing would be continued till the operational reliability is equal to 0.8 (at Δt=0.1), From Eq. (23) and Eq. (12), we get t=19.1221 weeks. If the desired reliability is 0.85, then t=31.0593 weeks (refer to Fig. 4). Similarly, in Second Data Set, N=2828.88, A=10.5057, a=0.0988842, a=241.325 and r=0.000907329. Suppose this software system is desired that this testing would be continued till the operational reliability is equal to 0.9 (at Δt=0.1), From Eq. (23) and Eq. (12), we get t=31.5848 weeks. If the desired reliability is 0.92 (0.95), then t=36.2893 (57.1049) weeks.

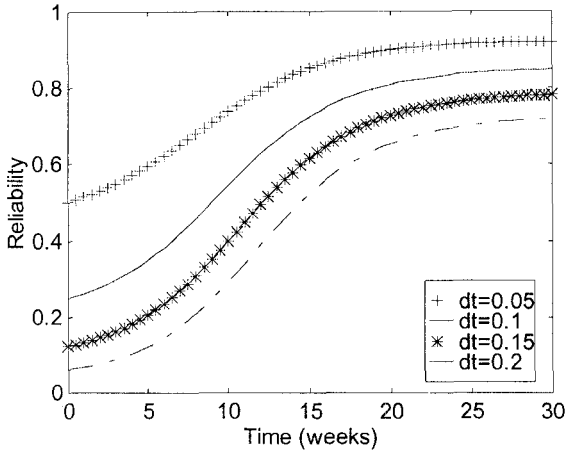

Figure 4. Plots of reliability of first data set versus time for Δt=0.05, 0.1, 0.15 and 0.2.

5.2 Optimal release time based on cost-reliability criterion

In this section, we will discuss the cost model and release policy based on the cost-reliability criterion. Using the total software cost evaluated by cost criterion, the cost of testing-effort expenditures during software testing/development phase and the cost of fixing errors before and after release are [2, 16, 17, 20]:

$$C(T) = C1 \times m(T) + C2 \times \{m(T_{LC}) - m(T)\} + C3 \times \int_0^T w(x)d \quad (24)$$

where T_{LC}=software life-cycle length
C1=cost of fixing an error during testing
C2=cost of fixing an error during operation (C2>C1)
C3=cost of testing per unit time

Differentiating the above equation with respect to T and setting it to zero, we obtain

$$\frac{dC(T)}{dT} = C1m'(T) - C2m'(T) + C3w(T) = 0 \quad (25)$$

$$\frac{m'(T)}{w(T)} = \frac{\lambda(T)}{w(T)} = are^{-rW'(T)} = r(a - m(T)) \quad (26)$$

Case 1: if T=0, then $m(0)$=0, $\frac{\lambda(T)}{w(T)} = ar$,

Case 2: if $T \to \infty$, then $w(\infty)$=N, $m(\infty)$ =$a(1 - e^{-\frac{rN\Delta t}{1+\Delta t}})$, $\frac{\lambda(T)}{w(T)} = are^{-\frac{rN\Delta t}{1+\Delta t}}$

Therefor, we can know that $\frac{\lambda(T)}{w(T)}$ is monotonically decreasing in T. If $\frac{\lambda(0)}{w(0)} = ar \le \frac{C3}{C2-C1}$, then $\frac{\lambda(T)}{w(T)} \le \frac{C3}{C2-C1}$ for $0 < T < T_{LC}$. Hence, for this case, the optimal software release time T^*=0. If $\frac{\lambda(0)}{w(0)}(= ar) > \frac{C3}{C2-C1} > \frac{\lambda(T)}{w(T)}(= are^{-\frac{rN\Delta t}{1+\Delta t}})$, there exists a finite and unique solution T_0 satisfying

$$\frac{\lambda(T)}{w(T)} = \frac{C3}{C2-C1} = r(a - m(T)) = are^{-r\left(\frac{N}{1+Ae^{-\alpha t}} - \frac{N}{1+\Delta t}\right)}$$

Rearranging the above equation, we obtain

$$T_0 = \frac{1}{\alpha} \times \ln\left(\frac{A\Theta}{N-\Theta}\right) \text{ minimizes C(T)} \quad (27)$$

where $\Theta = \frac{1}{r}(\ln(ar\frac{C2-C1}{C3})) + \frac{N}{1+A}$. Because $\frac{dC(T)}{dT} < 0$ for $0 < T < T_0$ and $\frac{dC(T)}{dT} > 0$ for $T > T_0$, the minimum of $C(T)$ is at $T = T_0$ for $T_0 \le T$.

From subsection 5-1, we can easily get the required testing time needed to reach the reliability objective R_0. Here our goal is to minimize the total software cost under the consideration of desired software reliability and then the optimal software release time is obtained. That is, we can mathematically minimize C(T) subject to $R(t+\Delta t|t) \geqq R_0$ where $0<R_0<1$ [2, 16, 17, 20].

T* = optimal software release time or total testing time
= max{T_0, T_1}
where T_0 = finite and unique solution T satisfying Eq. (24)
T_1 = finite and unique T satisfying $R(t + \Delta t|t) = R_0$ $(0 < R_0 < 1)$

Theorem 1 :
Assume C1>0, C2>0, C3>0, C2>C1, Dt>0, $0<R_0<1$, we have

1. if $\frac{\lambda(0)}{w(0)} > \frac{C3}{C2-C1}$ and $\frac{\lambda(T)}{w(T)} = are^{-\frac{rN\Delta t}{1+\Delta t}} < \frac{C3}{C2-C1}$ then T*= max{T_0, T_1} for $R(\Delta t|0)<R_0<1$ or T*=T_0 for $0<R_0 \le R(\Delta t|0)$
2. if $\frac{\lambda(0)}{w(0)} \le \frac{C3}{C2-C1}$ then T*=T_1 for $R(\Delta t|0)<R_0<1$ or T*=0 for $0<R_0 \le R(\Delta t|0)$
3. if $\frac{\lambda(0)}{w(0)} \ge \frac{C3}{C2-C1}$ then T*$\ge T_1$ for $R(\Delta t|0)<R_0<1$ or T* $\ge$ 0 for $0<R_0 \le R(\Delta t|0)$

In order to illustrate the above derivation, we consider the three real data set described in the previous section and use them to work as numerical examples on the optimal software release problem.

Discussions:

1) First Data Set: From the previous estimated parameters: we know N=54.8364, A=13.0334, a=0.226337, a=394.076, r=0.0427223 and assume $C1$=10, $C2$=50, $C3$=100, T_{LC}=100, R_0 =0.85, Δt=0.1. Then we get the optimal release time T_0 estimated as 20.3753 based on minimizing C(T) of Eq. (24) and T_1 estimated as 31.0593 based on satisfying the reliability criterion of $R(t+\Delta t|t)=R_0$. Moreover, since $\frac{\lambda(0)}{w(0)} > \frac{C3}{C2-C1}$, $\frac{\lambda(T)}{w(T)} = are^{-\frac{rNA}{1+A}} < \frac{C3}{C2-C1}$ and $R(\Delta t|0)=0.2481<R_0$, T^*= max {20.3753, 31.0593}=31.0593 weeks. The optimal total software cost C(T^*)=8572.21 (plot is given in Fig. 5) and the achieved software reliability R(31.0593+Δt(=0.1) |31.0593) is 0.85.

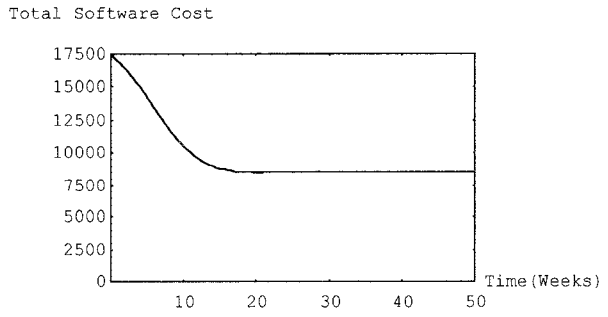

Figure 5. Total software cost of the first data set vs. time.

2) Second Data Set: From the previous estimated parameters: we know N=2828.88, A=10.5057, a=0.0988842, a=241.325, r=0.000907329 and assume $C1$=1, $C2$=50, $C3$=2, T_{LC}=200, R_0 =0.95, Δt=0.1. Then we get the optimal release time T_0 estimated as 34.4345 based on minimizing $C(T)$ of Equation (24) and T_1 estimated as 57.01049 based on satisfying the reliability criterion of R(t+Δt|t)=R_0. Since $\frac{\lambda(0)}{w(0)} > \frac{C3}{C2-C1}$, $\frac{\lambda(T)}{w(T)} = are^{-\frac{rNA}{1+A}} < \frac{C3}{C2-C1}$ and $R(\Delta t|0)<R_0$, T^*=max{34.4345, 57.01049 }=57.01049. The optimal total software cost C(T*)=5290.2 (see Fig. 6) and the achieved software reliability is R(57.01049+Δt(=0.1)|57.01049)=0.95 .

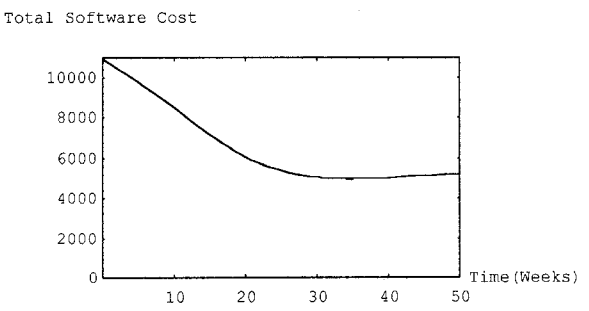

Figure 6. Total software cost of the second data set vs. time.

3) Third Data Set: From the previous estimated parameters: we know *N=29.1095, A=4624.89, a=0.493515, a=138.165, r=0.145098 and assume C1=1, C2=100, C3=50,* T_{LC}*=100,* R_0 *=0.95,* Δt=1. Then we get the optimal release time T_0 estimated as 20.9839 based on minimizing $C(T)$ of Equation (24) and T_1 estimated as 12.7957 based on satisfying the reliability criterion of $R(t+\Delta t|t)=R_0$. Because $\frac{\lambda(0)}{w(0)} > \frac{C3}{C2-C1}$, $\frac{\lambda(T)}{w(T)} = are^{-\frac{rNA}{1+A}} < \frac{C3}{C2-C1}$ and $R(\Delta t|0)<R_0$, T^* is estimated as max{20.9839, 12.7957}=20.9839. The optimal total software cost $C(T^*)$=1329.44 (plot is shown in Fig. 7).

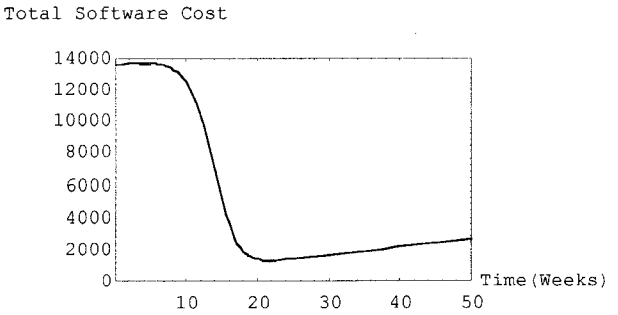

Figure 7. Total software cost of the third data set vs. time.

6. Conclusions

In this paper, we have proposed a Software Reliability Growth Model which incorporates a logistic testing-effort function and the testing effort consumption curve is different from the weibull-type curve used in fitting real date set. Due to incorporating the logistic testing-effort function and comparing it with the model proposed in [1-4], the proposed Software Reliability Growth Model

fits the real project data fairly well and it could give us a reasonable description of resource consumption behavior. Experimental results obtained are in close accord with the real data. In fact, the derivation of logistic testing-effort function in the original basic concepts already incorporates the human factors into consideration. Besides, the optimal release times of this model based on cost-reliability criterion are also illustrated by numerical examples.

Here, some consequences and future works are as follow:

1. The capability of the model to predict failure behavior from present and past failure behavior is called *predictive validity*. Following the work in [5] and using the real project failure data described in subsection 4-2, we can compute the relative error in prediction for the three data set at the end of testing as 0.0075, -0.106, and -0.00958 respectively (see Fig. 8). From the computation results, we see that the relative errors of the first and the third data set approach zero which indicate that our model provides an accurate estimation for these data set and the relative error of the second data set is negative which indicate that the model tends to underestimate the failure phenomenon under this testing. However, the above statements imply that our model still have better predictive validity based on real failures experienced and give a more accurate prediction. That is, the software reliability growth model with logistic testing-effort function will yield the better predictions for other reliability metrics [5, 19, 21-23].

2. Although we have demonstrated that the Logistic testing-effort function is applicable to the GO model and the combined model estimate the number of initial faults better than the model with Weibull-type consumption curve, only the GO model is not sufficient to prove the applicability of the proposed Logistic testing -effort function. Presently, we are investigating how to integrate the new testing-effort function into conventional software reliability models, such as Logarithmic Poisson Model, Inflection S-Shaped Model or Delayed S-Shaped Model, and so on. If the research results fit the real failure data well, we can conclude that the Logistic testing-effort function is superior and give a more accurate description/estimation for the resource consumption during software development process.

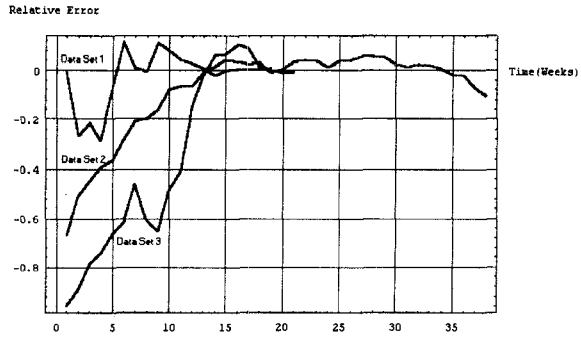

Figure 8. Relative error curve for different data set.

Acknowledgment

We would like to express our gratitude for the support of the National Science Council, Taiwan, R.O.C., under Grant NSC 86-2213-E259-002. Referees' helpful comments and suggestions are also highly appreciated.

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] 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.

[7] P. K. Kapur. and S. Younes, " Modeling an Imperfect Debugging Phenomenon with Testing Effort," *Proceedings of the 5th International Symposium on Software Reliability Engineering*, pp. 178-183, November 1994, Monterey, California.

[8] P. K. Kapur and R. B. Garg, " Modeling an Imperfect Debugging Phenomenon in Software Reliability ," *Microelectronics and Reliability*, Vol. 36, pp. 645-650, 1996.

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

[10] A. Wood, "Predicting Software Reliability, "*IEEE Software*, Nov. 1996, pp. 69-77.

[11] 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, November 1994, Monterey, California.

[12] M. Ohba, "Software reliability analysis models, " *IBM J. Res. Develop.*, Vol. 28, No. 4, pp. 428-443, July 1984.

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

[14] J. D. Musa, *Software Reliability Data*, report and data base available from Data and Analysis Center for Software, Rome Air Development Center, Rome, NY.

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

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

[17] P. K. Kapur and R. B. Garg, "Cost reliability Optimum Release Policies for a Software System under Penalty Cost", *Int. J. of Systems Science*, Vol. 20, pp. 2547-2562, 1989.

[18] A. L. Goel and K. Okumoto, "Time dependent error detection rate model for software reliability and other performance measures," *IEEE Trans. on Reliability*, Vol. R-28, No. 3, pp. 206-211, 1979.

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

[20] P. K. Kapur and R. B. Garg, "Cost-Reliability Optimum Release Policies for a Software System with Testing Effort," *OPSEARCH*, Vol. 27, No. 2, pp. 109-116, 1990.

[21] S. Yamada, J. Hishitani, and S. Osaki, "Software Reliability Growth Modeling: Models and Applications", *IEEE Trans. on Software Engineering*, Vol. SE-11, No. 12, pp.1431 -1437, 1985.

[22] P. K. Kapur and R. B. Garg, "A software reliability growth model for an error removal phenomenon, " *Software Engineering Journal*, pp. 291-294, 1992.

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