Analysis of Optimal Software Release Time Based on a Testing-Effort Dependent Reliability Model

基於測試心力相依可靠度模型之最佳軟體釋放時機分析

Chin-Yu Huang[*2], Jung-Hua Lo[**], Jenn-Wei Lin[1], and Bo-Ting Lin[*]

[*]*Department of Computer Science*
National Tsing Hua University
Hsinchu, Taiwan
cyhuang@cs.nthu.edu.tw

[**]*Department of Information Management*
Lan Yang Institute of Technology
I-Land, Taiwan
losir@mail.fit.edu.tw

Abstract

Software reliability is one of the most important aspects of software quality. Accurately modeling software reliability, and predicting its possible trends are essential to determining overall product′s reliability. Until now, many software reliability growth models (SRGMs) are proposed and they can help us to estimate time or resource needed to reach a reliability target. Actually, some important metrics can also be easily determined through SRGMs. One of the most important applications of SRGM is to determine the software release time. In this paper, we develop a useful method to compute the software release time considering cost, reliability and testing efficiency during the development phase. We first review a SRGM with generalized logistic testing-effort function and change-point. The proposed model can be precisely to illustrate the effectiveness of introducing new testing techniques. We then address the problem of how to decide when to stop testing and when to release software for use. In addressing the optimal release time, we consider cost and reliability factors. Moreover, we introduce the concept of testing efficiency. Several theorems and numerical illustrations are presented.

Keywords: Software Reliability, Software Testing, Testing-Effort, Non-homogeneous Poisson Process (NHPP), Software Cost.

[1]Jenn-Wei Lin is with Department of Computer Science & Information Engineering, Fun Jen Catholic University, Taipei, Taiwan.
Email: jwlin@csie.fju.edu.tw

[2]This research is supported by the National Science Council, Taiwan, R.O.C., under Grant NSC 92-2213-E-007-073.

摘　要

軟體可靠度是考量軟體品質的最重要的觀點之一。 正確的去模塑軟體可靠度，以及預測它可能的趨勢對於決定整個產品的可靠度是有其絕對必要的。截至目前為止，已有許多的軟體可靠度成長模型被提出且它們能幫助我們去評估為了要達成某種程度的可靠度目標所需要消耗的時間或資源。事實上，一些重要的度量也能經由軟體可靠度成長模型而來輕易地被決定。軟體可靠度成長模型的最重要幾個應用之一就是去決定軟體釋放時機。在本篇論文裡，我們提出了在軟體開發階段，一個用於計算軟體釋放時間的方法，而其考量了成本、可靠度以及測試效率等。我們首先回顧一個具有後勤測試心力函數與變動點之軟體可靠度成長模型，所提出的模型能精確地說明在軟體開發階段，倘引進新的測試技巧其所會造成的效果。接著我們也說明了何時才能停止測試及何時準備去釋放該軟體以供外界使用的問題。在說明最佳軟體釋放時機的同時，我們也考慮了成本及可靠度這兩個因素。此外，我們亦介紹了測試效率的概念，幾個定理及數值例子也將會提出且呈現出來。

關鍵詞：軟體可靠度、軟體測試、測試心力、非齊次卜以松過程、軟體成本。

1. Introduction

With the steadily growing power and reliability of hardware, software has been identified as a major stumbling block in achieving desired levels of system dependability. For example, US DOD spending for software intensive systems is significant and it continues to increase. Furthermore, software costs as a percentage of total computer system costs continue to increase; while associated hardware costs are continuing to decrease. To illustrate this point, in 1962 the ratio of computer hardware costs to software costs was 80:20. By 1985 the ratio had reversed to 20:80. Actually, it is very important to ensure the quality of the underlying software systems in the sense that they perform their functions correctly. Software reliability (SR) is defined as the probability of failure-free software operation for a specified period of time in a specified environment [1]. During the past 20 years, a number of *Software Reliability Growth Models* (SRGMs) were proposed [2-4]. From the study in [4-7], we can find that many authors considered an NHPP as a stochastic process to describe the fault process. Most SRGMs use calendar time as the unit of fault detection period. Very few SRGMs use the human power, number of test case runs, or CPU time as the unit [2]. Recently, we [8-11] proposed a new SRGM that incorporates the concept of *logistic* testing-effort function (TEF) into an NHPP model to get a better description on the software fault phenomenon.

On the other hand, during the development phase, software is subjected to several stages of testing to identify existing problems. At the end of each test stage, corrections and modifications are made to the software with the hope of increasing its reliability. Therefore, if we want to detect more additional faults in a short time, it is advisable to introduce new tools/techniques, which are fundamentally different from the methods currently in use. The benefit of these methods is that they can design/propose several testing programs/automated testing tools to test software for satisfying the client's technical requirements, schedule, and budget. Therefore, in this paper, we first review a SRGM with generalized *logistic* TEF & change-point. The proposed model has a fairly accurate prediction capability. In addition to modeling the software fault-detection process, we will also address the problem of how to decide when to stop testing and release software. We discuss the optimal software release time problem based on cost and reliability considering TE and efficiency.

In the remaining of this paper, there are four more sections. We give a brief review of the SRGM with a generalized logistic TEF and change-point in Section 2. Section 3 introduces the concept of testing efficiency obtained by new techniques and tools during testing. The optimal software release time problem based on minimizing cost subject to achieving a given level of reliability considering the extra cost of introducing new techniques/tools during testing is discussed in Section 4. Finally, Section 5 concludes this paper.

2. Testing-effort function and change-point problem

2.1. Review of SRGM with generalized logistic testing-effort function

Assumptions [8-11]:

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 the manifestation of remaining faults in the system.
3). The mean number of faults detected in the time interval $(t, t+\Delta t]$ by the current TE expenditures is proportional to the mean number of remaining faults in the system.
4). The proportionality is a constant over time.
5). The consumption curve of testing effort is modeled by a generalized *logistic* TEF.
6). Each time a failure occurs, the fault that caused it is immediately and perfectly removed and no new faults are introduced.

If we define the expected value number of faults, $N(t)$, whose mean value function is known as $m(t)$, then an SRGM based on NHPP can be formulated as a Poisson process:

$$P_r[N(t)=n]=\frac{[m(t)]^n\exp[-m(t)]}{n!} \qquad (1)$$

Furthermore, if the number of faults detected by the current TE expenditures is proportional to the number of remaining faults, then we obtain the following differential equation [9]:

$$\frac{dm(t)}{dt}\times\frac{1}{w(t)}=r\times[a-m(t)] \qquad (2)$$

where $m(t)$ is the expected mean number of faults detected in time $(0, t]$ ($m(0)=0$), $w(t)$ is current TE consumption at time t, a is the expected number of initial faults, and r is error detection rate per unit TE at testing time t that satisfies $r>0$.

Solving the above differential equation, we have

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

Eq. (3) is an NHPP model with mean value function considering the TE consumption. The consumed TE indicates how effective the faults are detected in the software and can be modeled by different distributions [12-15].

From the previous studies in [7-10], we know that the *logistic* TEF (i.e. the Parr model [16]) is based on a description of the actual software development process and can be used to describe the work profile of software development. If we relax some assumptions and take into account the structured development effort, we get a generalized *logistic* TEF [11]:

$$W(t) = N \times \left(\frac{(\kappa+1)/\beta}{1+Ae^{-\alpha\kappa t}} \right)^{1/\kappa} \qquad (4)$$

where κ is the structuring index and β is a constant. When κ=1, the above equation becomes

$$W(t) = \frac{N}{1+Ae^{-\alpha t}} \times \frac{2}{\beta} \qquad (5)$$

If β is viewed as a normalized constant and we have β=2, the above equation is equal to Eq. (4). Similarly, if κ=2, we have

$$W(t) = \frac{N}{\sqrt{1+Ae^{-2\alpha t}}} \sqrt{\frac{3}{\beta}} \qquad (6)$$

If β=κ+1, we get a more generalized solution:

$$W(t) = \frac{N}{\sqrt[\kappa]{1+Ae^{-\alpha\kappa t}}} \qquad (7)$$

Furthermore, the testing effort $w(t)$ reaches its maximum value at time

$$t_{\max} = \frac{\ln \frac{A}{\kappa}}{\alpha\kappa} \qquad (8)$$

2.2. SRGM with generalized logistic TEF & change-point

In general, among various SRGMs two most important parameters affect reliability: *the number of initial faults* and *the fault detection rate*. The number of initial faults is the number of faults in the software at the beginning of test. This number is usually a representative measure of SR. Knowing the number of residual faults can help us to determine whether the software is suitable for customers to use or not and how much more testing resources are required. It can also provide an estimate of the number of failures that customers will encounter when they use this software in practice [10].

The FDR, on the other hand is used to measure the effectiveness of fault detection by test techniques and test cases. In the vast literature, most researchers assume a *constant* detection rate per fault in deriving their SRGMs. That is, they assume that all faults have equal probability of being detected during the software testing process and the rate remains constant over the intervals between fault occurrences. In fact, the FDR strongly depends on the skill of test teams, program size, and software testability. Typically, 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 program modules.

At the beginning of the testing phase, many faults can be discovered by inspection and the FDR depends on the fault discovery efficiency, the fault density, the TE, and the inspection rate. On the other hand, in the middle stage of testing phase, the FDR 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 [9, 11]. Practically, during the SDP, we can detect/remove more additional faults through some new techniques. Therefore, the FDR may be not a constant or smooth, i.e., it may be changed at some time moment τ called change-point [17-21]. Actually, we can incorporate both generalized *logistic* TEF and change-point into software reliability growth modeling. Therefore, for the assumption 4 in Section 2, it may be modified as: *the proportionality is not just a constant or in some case may be changed at some time moment τ called change-point.*

Therefore, we can describe an SRGM based on TEF and change-point as follow:

$$\frac{dm_1(t)}{dt} \times \frac{1}{w(t)} = r(t) \times [a - m_1(t)] \qquad (9)$$

and
$$r(t) = \begin{cases} r, & 0 \le t \le \tau \\ r_2, & t > \tau \end{cases} \qquad (10)$$

where a>0, $m_1(t)$ is the expected mean number of faults detected in time (0, t] and $m_1(0)$=0.

Note that Eq. (9) have two components which influence the number of faults detected (NFD): the TE function $w(t)$, and the FDRs $r(t)$. Since the *s*-expected current detected fault content is finite at any time, $m_0(t)$ is an increasing function of ; $m_0(0)$=0. Solving above two equations, we have

$$m_1(t) = a \times (1 - e^{-r(W(t)-W(0))}) = a \times (1 - e^{-rW^*(t)}),$$
$$\text{when } 0 \le t \le \tau . \quad (11)$$

$$m_1(t) = a \times (1 - e^{-\{rW^*(\tau)+r_2(W(t)-W(\tau))\}}),$$
$$\text{when } t > \tau . \quad (12)$$

3. New techniques for increasing software testing efficiency

It is well known that when the software coding is completed, the testing phase comes next and it is a necessary but expensive process. Once all the detectable faults are removed from a new computer software package, the computer company will need to determine when to stop testing and make a software risk evaluation. If the results meet their requirement specifications and the related criteria are also satisfied, the company will adorn and announce that this software product is ready for releasing. Therefore, adequately adjusting some specific parameters of a SRGM and adopting the corresponding actions in the proper time interval can greatly help us to speedup getting the desired solution. For example, we have discussed the applications of TE control and management problem in our previous studies [8-11]. Alternative to controlling the TE expenditures, we believe that new testing schemes will help achieve a given operational quality at a specified time. That is, through some new techniques, we can detect more additional faults, although these new methods will increase the extra cost [11].

On the other hand, the change-point problems have been studied by many authors. Zhao [17] think that the change-point can be occurred when the testing strategy and testing-resource allocation are changed. Besides, the increasing knowledge of the program, the testing facilities and other random factors can be the causes of the change-points. Here we will to modify the concept of change-point in software reliability modeling and use it to describe the behavior or characteristics of introducing automated testing techniques/tools during the SDP. Here we will use Ohba's real data set as illustration [22]. Let us consider the following scenario:

1) Due to economic considerations, software testing and debugging will eventually be terminated at a specified time point, T_2 (here we assume T_2=30).
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 and $0<T_1<T_2$ (here we assume T_1=19).
3) By applying the estimated parameters into the SRGM, the test teams can predict the cumulative number of faults at time T_2. The estimated value may have already satisfied the developers' desired goal. If not, in order to meet the requirements, the developers must detect more extra faults during the time interval T_2–T_1.

In our past studies [11], we ever introduced a gain parameter to describe the behavior or characteristics of new testing techniques/tools and incorporate it into the mean value function. That is, the modified mean value function is depicted in the following:

$$m_e(t) = a(1-e^{-\sigma r(W(t)-W(0))}) = a(1-e^{-\sigma r W^*(t)}) \quad (13)$$

where $t>\tau$ and σ is the gain parameter (GP). Therefore, from Eq. (12) & (13), we have

$$\sigma = \frac{r_2W(t) - (rW(0) + r_2W(\tau) - rW(\tau))}{r(W(t) - W(0))}$$

On the other hand, from Eq. (3), (11), and (12), we can also re-define the gain-effect of employing new automated techniques/tools and depicted it as follow:

$$\frac{a(1-e^{-\{rW^*(\tau)+r_2(W(t)-W(\tau))\}})}{a(1-e^{-rW^*(t)})} = (1+P), \text{ where } t>\tau.$$

Hence, we can conclude that

$$\begin{aligned} m_1(t) &= a(1-e^{-\{rW^*(\tau)+r_2(W(t)-W(\tau))\}}) \\ &= a(1-e^{-\sigma r(W(t)-W(0))}) \\ &= (1+P)\times a(1-e^{-r(W(t)-W(0))}) \\ &= (1+P)\times m(t) \qquad (14) \end{aligned}$$

where P is the additional fraction of faults detected by using new automated tools or techniques during testing, and $\tau<t<T_{LC}$ (T_{LC} is software life-cycle length and generally $\to\infty$).

We can treat Eq. (14) as the modified MVF of adopting new techniques into SDP and it is effective when $t>\tau$. Rather, we can use $(1+P)m(t)$ to represent the MVF when new techniques/methods are introduced. Altogether, introducing new automated tools/methods may help us in detecting & removing more additional faults which are hard to detect without these new methods. But the most important thing is how to provide enough information about these approaches to the test team. Before adopting these automated techniques/tools, we should get the quantitative information from the industrial data relative to the methods' past performance applied in other instances, or qualitative information from the subjective valuation of methods' attributes. Certainly, the methods' past performance in aiding the reliability growth should be considered in determining whether they will be successful again or not [11].

4. Optimal software release policy

Optimization models for software release time are many and varied. A majority of models view the software system as a whole and study the growth of reliability as testing time increases. One of the most important applications of SRGM is to determine the software release time. When the software testing is completed, software product is ready to release to users. The software release time problem is posed as an unconstrained optimization problem where the cost associated with testing is minimized or a constrained optimization problem where constraints on minimum reliability requirements are imposed. However, proper timing is very important. If the reliability of the software does not meet the manager's goal, the developers or testers may introduce external help to aid in testing [23-25]. In this section, based on the proposed SRGM, useful rules are developed for determining optimal software release time subject to various constraints.

4.1 Software release time based on cost criterion

Okumoto and Goel [4, 23] firstly discussed the software optimal release policy from the cost-benefit viewpoint. Using the total software cost evaluated by cost criterion, the cost of TE expenditures during software development phase and the cost of correcting errors before and after release is:

$$C1(T) = C_1 m(T) + C_2[m(T_{LC}) - m(T)] + C_3\int_0^T w(x)dx \tag{15}$$

where C_1 is the cost of correcting an error during testing, C_2 is the cost of correcting an error during operation, and C_3 is the cost of testing per unit TE expenditures [4, 9, 11, 24, 26].

Generally, in order to detect additional faults during testing, the test teams/debuggers may use new automated tools or techniques if they are available. Hence the cost trade-off of tools should be considered in software cost model. But they thereby save some of the greater expense of correcting errors during operation. By summing up above stated cost factors, the modified software cost model can be shown as follow:

$$C2(T) = C_0(T) + C_1\times(1+P)\times m(T) + C_2\times[m(T_{LC}) - (1+P)\times m(T)] + C_3\times\int_0^T w(x)dx \tag{16}$$

where $C_0(T)$ is the cost function of including automated tools/techniques to detect an additional fraction P of faults during testing.

In fact, $C_0(T)$ may not be a constant during the testing phase of software development process. Moreover, in order to determine the testing cost $C_0(T)$, the most general cost estimating technique is to use the parametric methods if there are some meaningful data available. Under the cost-benefit considerations, the automated tools or techniques will pay for themselves. By differentiating Eq. (16) with respect to T and let C_1 $(1+P)$= C_1^* and C_2 $(1+P)$= C_2^*, we have

$$\frac{d}{dT}C2(T) = \frac{d}{dT}C_0(T) + C_1^*\frac{d}{dT}m(T) - C_2^*\times\frac{d}{dT}m(T) + C_3\times w(T) \tag{17}$$

$$= \frac{d}{dT}C_0(T) + C_1^* arw(T)e^{-rW^*(T)} - C_2^* arw(T)e^{-rW^*(T)} + C_3\times w(T)$$

4.1.1. $C_0(T)$=C_0, $T \geq \tau$; $C_0(T)$=0, $T<\tau$, where τ is the start time of adopting new techniques/methods.

$$\frac{d}{dT}C2(T) = w(T)\times[-ar\times(C_2^* - C_1^*)e^{-r((W(T)-W(0))} + C_3] \tag{18}$$

Since $w(t)$>0 for $0<T<\infty$, $\frac{d}{dT}C2(T)$=0 if

$$ar\times(C_2^* - C_1^*)e^{-r(W(T)-W(0))} = C_3 \tag{19}$$

The left-side in Eq. (19) is monotonically decreasing function of T.

If $ar\times(C_2^* - C_1^*)e^{-r(W(\tau)-W(0))} \leq C_3$, then $ar\times(C_2^* - C_1^*)e^{-r(W(T_{LC})-W(0))} < C_3$ for τ<T<T_{LC}. Therefore, the optimal software release time $T^*=\tau$ since $\frac{d}{dT}C2(T)$>0 for τ<T<T_{LC}. On the other hand, if $ar\times(C_2^* - C_1^*)\times e^{-r(W(\tau)-W(0))} > C_3$ and $ar\times(C_2^* - C_1^*)e^{-r(W(T_{LC})-W(0))} < C_3$, there exists a finite and unique solution T_0 satisfying Eq. (19).

$$T_0 = \frac{1}{\alpha}\times\ln(\frac{A\Theta^{\kappa}}{N^{\kappa} - \Theta^{\kappa}}) \text{ minimizes } C2(\mathrm{T}) \tag{20}$$

where $\Theta = \frac{1}{r}(\ln[ar\frac{C_2^* - C_1^*}{C_3}]) + \frac{N}{\sqrt[\kappa]{1+A}}$

since $\frac{d}{dT}C2(T)$<0 for $\tau<T<T_0$ and $\frac{d}{dT}C2(T)$>0 for $T_0<T<T_{LC}$.

If $ar\times(C_2^* - C_1^*)e^{-r(W(T_{LC})-W(0))} \geq C_3$, then $ar\times(C_2^* - C_1^*)e^{-r(W(T)-W(0))} > C_3$ for $\tau<T<T_{LC}$. Therefore, the optimal software release time $T^*=T_{LC}$ since $\frac{d}{dT}C2(T)$<0 for $\tau<T<T_{LC}$.

4.1.2. $C_0(T) = C_{01} + C_0\int_\tau^T w(t)dt$, $T \geq \tau$; $C_0(T)$=0, $T<\tau$, where C_{01} is the nonnegative real number that indicates the basic cost of adopting new techniques.

$$\frac{d}{dT}C2(T) = C_0 w(T) + C_1^* arw(T)e^{-rW^*(T)} - C_2^*\times arw(T)e^{-rW^*(T)} + C_3\times w(T)$$

$$= w(T)\times[ar\times(C_1^* - C_2^*)e^{-r((W(T)-W(0))} + C_3 + C_0] \tag{21}$$

Since $w(t)$>0 for $0\leq T<\infty$, $\frac{d}{dT}C2(T) = 0$ if

$$ar\times(C_2^* - C_1^*)e^{-r(W(T)-W(0))} = C_3 + C_0 \tag{22}$$

Because the left-side in Eq. (22) is monotonically decreasing function of T, if $ar\times(C_2^* - C_1^*)e^{-r(W(\tau)-W(0))} > (C_3 + C_0)$ and $ar\times(C_2^* - C_1^*)e^{-r(W(T_{LC})-W(0))} < (C_3 + C_0)$, there exists a finite and unique solution T_0 satisfying Eq. (22).

$$T_0 = \frac{1}{\alpha} \times \ln\left(\frac{A\Theta^{\kappa}}{N^{\kappa} - \Theta^{\kappa}}\right) \text{ minimizes } C2(T) \quad (23)$$

where $\Theta = \frac{1}{r}\left(\ln\left[ar\frac{C_2^* - C_1^*}{C_3 + C_0}\right]\right) + \frac{N}{\sqrt[\kappa]{1+A}}$.

4.1.3. $C_0(T) = C_{01} + C_0 \times (\int_{\tau}^{T} w(t)dt)^m, T \geq \tau$; $C_0(T)=0$, $T<\tau$.

$$\frac{d}{dT}C2(T) = C_0 m w(T) \times (\int_{\tau}^{T} w(t)dt)^{m-1} + C_1^* arw(T) \times e^{-rW^*(T)} - C_2^* arw(T)e^{-rW^*(T)} + C_3 w(T)$$

$$= w(t) \times [ar \times (C_1^* - C_2^*)e^{-rW^*(t)} + C_3 + C_0 m \times (\int_{\tau}^{T} w(t)dt)^{m-1}]$$

because $w(t)>0$ for $0 \leq T < \infty$, $\frac{d}{dT}C2(T) = 0$ if

$$P(T) \equiv [ar(C_2^* - C_1^*)e^{-rW^*(t)} - C_0 m(\int_{\tau}^{T} w(t)dt)^{m-1}] = C_3 \quad (24)$$

The left-side in Eq. (24) is monotonically decreasing function of T. Therefore, if

$ar \times (C_2^* - C_1^*)e^{-r(W(\tau)-W(0))} > C_3$ and $P(T_{LC})<C_3$,

it means that there exists a finite and unique solution T_0 satisfying Eq. (24) which can be solved by numerical methods [26]. It is noted that $\frac{d}{dT}C2(T)$ <0 for $0 \leq \tau \leq T < T_0$ and $\frac{d}{dT}C2(T)$ >0 for $T>T_0$. Thus, $T=T_0$ minimizes $C2(T)$ for $T_0 <T_{LC}$.

4.2 Numerical examples.

Here we illustrate how to minimize the software cost in which the new automated tools/techniques are introduced during testing. For the estimated parameters of our proposed model, we have N=48.7768, A=429.673, α=0.158042, κ=2.63326, a=369.029, r=0.0509553. Besides, C_{01}=\$1000, C_1=\$10 per error, C_2=\$50 per error, C_0=\$10, C_3=\$100 per unit TE expenditures, τ=19, and T_{LC}=100 weeks [11]. The numerical example on the relationship between the cost optimal release time and P is given in Table 1. From Table 1, we find that the bigger the P, the larger are the optimal release time and the smaller the total expected software cost. The reason is that if we have better testing performance, we can detect more latent or undetected faults through additional techniques/tools. Therefore, we can really shorten the testing time and release this software earlier. Similarly, the relationship between the optimal release time and P based on the other cost function is shown in Table. 2.

Table 1: Relationship between T_0*, $C(T_0$*), and P based on the cost function

$$C_0(T) = 1000 + 10 \times \int_{19}^{100} w(t)dt$$

P	Cost Optimal Release Time T_0*	Total Expected Cost $C(T_0$*)
0.01	19.7381	5574.05
0.02	20.0016	5414.5
0.03	20.2887	5254.74
0.04	20.6072	5094.77
0.05	20.965	4934.6
0.06	21.9747	4774.24
0.07	21.8541	4613.69
0.08	22.4464	4452.94
0.09	23.2027	4292.02
0.10	24.2839	4130.91

Table 2: Relationship between T_0*, $C(T_0$*), and P based on the cost function

$$C_0(T) = 1000 + 10 \times (\int_{19}^{100} w(t)dt)^{1.2}$$

P	Cost Optimal Release Time T_0*	Total Expected Cost $C(T_0$*)
0.01	19.6006	5573.46
0.02	19.7501	5414.07
0.03	19.9157	5254.54
0.04	20.0983	5094.88
0.05	20.2998	4935.07
0.06	20.5221	4775.13
0.07	20.7684	4615.04
0.08	20.0436	4454.81
0.09	21.3539	4294.43
0.10	21.7086	4133.91

4.3 Software release time based on reliability Criterion

In general, the software release time problem is also associated with the reliability of software system. If we know that the SR has reached an acceptable reliability level, then we can determine the right time to release this software. Here we define the measure of SR for the proposed model, i.e., the ratio of the cumulative number of detected faults at the time T to the expected number of initial faults.

$$R(T) = \begin{cases} \frac{m(T)}{a}, 0 \leq t \leq \tau \\ \frac{m_1(t)}{a} = \frac{(1+p)m(T)}{a}, t > \tau \end{cases} \quad (25)$$

We can solve this equation and obtain a unique T_1 satisfying $R(T_1)=R_0$. It is noted that $R(T)$ is increasing in T. Using the above equation, we can easily get the required testing time needed to reach the reliability objective R_0 or decide whether the reliability objective R_0 is reached or not at a specified time interval [25-26]. If $R(\tau)<R_0$, there exists a unique $T_1>\tau$ satisfying $R(T_1)=R_0$. Therefore, by solving Eq. (25), we can determine the testing time needed to reach a desired reliability. Tables 3 and 4 show the relationship between the cost optimal release time T_0*, and P based on two different cost functions and the corresponding SR $R(T)$. From Tables 3 and 4, we find that as P increases, the optimal release time T_1* increases.

Table 3: Relationship between the T_0*, $R(T)$, and P based on the cost function

$$C_0(T) = 1000 + 10 \times \int_{19}^{100} w(t)dt$$

P	Cost Optimal Release Time T_0*	$R(T)$
0.01	19.7381	0.890825
0.02	20.0016	0.900818
0.03	20.2887	0.910616
0.04	20.6072	0.920616
0.05	20.965	0.930616
0.06	21.9747	0.941974
0.07	21.8541	0.0950601
0.08	22.4464	0.096064
0.09	23.2027	0.970618
0.10	24.2839	0.980617

Table 4: Relationship between T_0*, $R(T)$, and P based on cost function

$$C_0(T) = 1000 + 10 \times (\int_{19}^{100} w(t)dt)^{1.2}$$

P	Cost Optimal Release Time T_0*	$R(T)$
0.01	19.6006	0.889954
0.02	19.7501	0.89949
0.03	19.9157	0.909071
0.04	20.0983	0.91869
0.05	20.2998	0.928341
0.06	20.5221	0.938019
0.07	20.7684	0.94772
0.08	20.0436	0.953784
0.09	21.3539	0.967182
0.10	21.7086	0.976937

4.4 Software release time based on cost-reliability criterion considering efficiency

From Section 4.3, 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 to achieve the desired SR and then the optimal software release time is obtained. Therefore, the optimal release policy problem can be formulated as minimize C(T) and subject to $R(T) \geq R_0$ where $0<R_0<1$.

T* = optimal software release time = ***max***(T_0, T_1) where T_0= finite and unique solution T satisfying Eq. (20), Eq. (23), or Eq. (24), and T_1= finite and unique T satisfying Eq. (25).

Combining the cost and reliability requirements and considering the efficiency, we have the following theorems.

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

(1) if $ar \times (C_2^* - C_1^*)e^{-r(W(\tau)-W(0))} > C_3$ and $ar \times (C_2^* - C_1^*)e^{-r(W(T_{LC})-W(0))} < C_3$, T^*=***max*** (T_0, T_1) for $R(\tau)<R_0<1$ or $T^*=T_0$ for $0<R_0 \leq R(\tau)$.

(2) if $ar \times (C_2^* - C_1^*)e^{-r(W(\tau)-W(0))} < C_3$, $T^* = T_1$ for $R(\tau)<R_0<1$ or $T^*=\tau$ for $0<R_0 \leq R(\tau)$.

(3) if $ar \times (C_2^* - C_1^*)e^{-r(W(T_{LC})-W(0))} > C_3$, $T^* \geq T_1$ for $R(\tau)<R_0<1$ or $T^* \geq \tau$ for $0<R_0 \leq R(\tau)$.

Theorem 2:
Assume $C_0(T) = C_{01} + C_0 \int_{\tau}^{T} w(t)dt$, C_{01}, $C_0>0$, $C_1>0$, $C_2>0$, $C_3>0$, and $C_2>C_1$, we have

(1) if $ar \times (C_2^* - C_1^*)e^{-r(W(\tau)-W(0))} > (C_3 + C_0)$ and $ar \times (C_2^* - C_1^*)e^{-r(W(T_{LC})-W(0))} < (C_3 + C_0)$, T^*=***max***(T_0, T_1) for $R(\tau)<R_0<1$ or $T^*=T_0$ for $0<R_0 \leq R(\tau)$.

(2) if $ar \times (C_2^* - C_1^*)e^{-r(W(\tau)-W(0))} < (C_3 + C_0)$, $T^*=T_1$ for $R(\tau)<R_0<1$ or $T^*=\tau$ for $0<R_0 \leq R(\tau)$.

(3) if $ar \times (C_2^* - C_1^*)e^{-r(W(T_{LC})-W(0))} > (C_3 + C_0)$, $T^* \geq T_1$ for $R(\tau)<R_0<1$ or $T^* \geq \tau$ for $0<R_0 \leq R(\tau)$.

Theorem 3:
Assume $C_0(T) = C_{01} + C_0 \times (\int_{\tau}^{T} w(t)dt)^m$, C_{01}, $C_0>0$, $C_1>0$, $C_2>0$, $C_3>0$, and $C_2>C_1$, we have

(1) if $ar \times (C_2^* - C_1^*)e^{-r(W(\tau)-W(0))} > C_3$ and $P(T_{LC})<C_3$, T^*=***max***(T_0, T_1) for $R(\tau)<R_0<1$ or $T^*=T_0$ for $0<R_0 \leq R(\tau)$.

(2) if $ar \times (C_2^* - C_1^*)e^{-r(W(\tau)-W(0))} < C_3$, $T^*=T_1$ for $R(\tau)<R_0<1$ or $T^*=\tau$ for $0<R_0 \leq R(\tau)$.

(3) if $P(T_{LC})>C_3$, $T^* \geq T_1$ for $R(\tau)<R_0<1$ or $T^* \geq \tau$ for $0<R_0 \leq R(\tau)$.

From the above theorems, we can easily determine the optimal software release time based on the cost and reliability requirements considering efficiency.

5. Conclusions

In this paper we present an SRGM with generalized *logistic* TEF & change-point. It is a more realistic model and very suitable for describing the software fault detection/removal process. Furthermore, we also discussed the effects of introducing new tools/techniques for increased efficiency of software testing, and studied the related optimal software release time problem from the cost-reliability viewpoint. The procedure for determining the optimal release time has been developed and the optimal release time has been shown to be finite. In practice, sometimes it is difficult for us to locate the faults that have caused the failure based on the test data reported in the test log and test anomaly documents. Therefore, it is advisable to introduce new tools/techniques, which are fundamentally different from the methods in use.

References

[1] American Institute of Aeronautics and Astronautics, *Recommended Practice for Software Reliability* ANSI/AIAA R-013-1992, February 23, 1993.

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

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

[4] Xie M (1991). Software Reliability Modeling. World Scientific Publishing Company.

[5] H. Pham (2000), *Software Reliability*, Springer-Verlag.

[6] Y. K. Malaiya and P. K. Srimani (1990), *Software Reliability Models*: *Theoretical Developments, Evaluation and Applications*, IEEE Computer Society Press.

[7] P. Rook (1990). *Software Reliability Handbook*. Elsevier Applied Science.

[8] C. Y. Huang, M. R. Lyu, and S. Y. Kuo, "A Unified Scheme of Some Non- Homogenous Poisson Process Models for Software Reliability Estimation," *IEEE Trans. on Software Engineering*, Vol. 29, No. 3, pp. 261-269, March 2003.

[9] C. Y. Huang and S. Y. Kuo, "Analysis and Assessment of Incorporating Logistic Testing Effort Function into Software Reliability Modeling," *IEEE Trans. on Reliability,* Vol. 51, No. 3, pp. 261-270, Sept. 2002.

[10] S. Y. Kuo, C. Y. Huang, and M. R. Lyu, "Framework for Modeling Software Reliability, Using Various Testing-Efforts and Fault-Detection Rates," *IEEE Trans. on Reliability*, Vol. 50, No. 3, pp. 310-320, Sept. 2001.

[11] C. Y. Huang, S. Y. Kuo, and M. R. Lyu, "Optimal Software Release Policy Based on Cost, Reliability and Testing Efficiency," *Proceedings of the 23rd IEEE Annual International Computer Software and Applications Conference* (COMPSAC'99), pp. 468-473, Oct. 1999, Phoenix, Arizona.

[12] 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.

[13] 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.

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

[15] 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.

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

[17] M. Zhao, "Change-Point Problems in Software and Hardware Reliability," *Communications in Statistics-Theory and Methods*, Vol. 22(3), pp. 757-768, 1993.

[18] H. J. Shyur, "A Stochastic Software Reliability Model with Imperfect-Debugging and Change-Point," *Journal of Systems and Software*, Vol. 66, pp. 135-141, 2003.

[19] K. M. Jeong, "An Adaptive Failure Rate Change-Point Model for Software Reliability," *International Journal of Reliability and Applications*, Vol.2, No.3, pp. 99-207, 2001.

[20] Y. P. Chang, "Estimation of Parameters for Non-homogeneous Poisson Process Software Reliability with Chang-Point Model," *Communications in Statistics: Simulation and Computation*, Vol. 30, pp. 623-635, 2001.

[21] F. Z. Zou, "A Change-Point Perspective on the Software Failure Process," *Software Testing, Verification and Reliability*, Vol. 13, pp.85-93, 2003.

[22] M. Ohba, "Software Reliability Analysis Models," *IBM J. Res. Develop.*, Vol. 28, pp. 428-443, 1984.

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

[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, 1996.

[25] S. Zheng, "Dynamic Release Policies for Software Systems with a Reliability Constraint," *IIE Transactions*, Vol. 34 (3), pp. 253-262, March 2002.

[26] B. Boehm (1981). *Software Engineering Economics*. Prentice-Hall.