The Journal of Systems & Software 188 (2022) 111267

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A software reliability growth model for imperfect debugging☆

Yeu-Shiang Huang [a,*], Kuei-Chen Chiu [b], Wan-Ming Chen [c]

[a] Department of Industrial and Information Management, Center for Innovative FinTech Business Models, National Cheng Kung University, Taiwan
[b] Department of Finance, Shih Chien University (Kaohsiung Campus), Taiwan
[c] Department of Industrial and Information Management, National Cheng Kung University, Taiwan

ARTICLE INFO

Article history:
Received 19 April 2021
Received in revised form 23 December 2021
Accepted 9 February 2022
Available online 16 February 2022

Keywords:
Software reliability
Non-homogeneous Poisson process (NHPP)
Imperfect debugging
Learning effect
Change point

ABSTRACT

In general, software reliability growth models (SRGMs) are often developed based on the assumptions of perfect debugging, single error type, and consistent testing environment. However, such the assumptions may be unrealistic for testing projects because new errors are always introduced during the debugging process, software errors are not alike, and the testing environment may change as the workforce escalates. Furthermore, the learning effect of the debugging process is taken under advisement, and assumes that it is unstable since the process of the error removal is also imperfect, which may cause a fluctuation of errors in the system. Therefore, the study is based on the Non-Homogeneous Poisson Process with considerations of the phenomenon of imperfect debugging, varieties of errors and change points during the testing period to extend the practicability of SRGMs. Furthermore, the error fluctuation rate is described by a sine cyclical function with a decreasing fluctuation breadth during the detection period, which indicates that the influence of increasing new errors during the testing period will be gradually unremarkable as the testing time elapses. Besides, the expected time of removing simple or complex errors is assumed to be different truncated exponential distributions. Finally, the optimal software release policies are proposed with considerations of the costs which occur in the testing and warranty period under an acceptable threshold of software reliability.

© 2022 Elsevier Inc. All rights reserved.

1. Introduction

Software reliability is an important basis for related decision-making activities for software developing teams. Research on software reliability has often assumed that perfect debugging is available in which errors can be immediately removed or corrected when they are detected. In practice, new errors may emerge due to the removal of an error, and debugging teams may not be capable of eliminating the errors thoroughly. The issue of imperfect debugging has therefore been raised in recent years. In addition, debugging staffs who repeatedly perform the removal of software errors may have learning effects that influence the number of errors removed in a system. Software errors can be simply distinguished into two categories: the simple and the complex ones. The former indicates that minor errors can be removed with lower cost and little time while the latter indicates major errors take more time and higher cost to be corrected. In general, the performance of error debugging is unstable at the beginning of the testing period due to the fact that the testing staff has not been so familiar with the characteristics of the software system yet. The learning effect will be beneficial in enhancing the efficiency of debugging process. As the testing elapsed, the learning effect will become critical, and the performance of error debugging will eventually be substantially improved.

Besides, testing environments may vary in practice due to factors such as hiring new testing personnel, the replacement of hardware, and the changes in testing tools or strategies, which result in changes in the efficiency of debugging. These changes cause conditions in which software reliability no longer grows steadily but rather grows with a different trend after every change event. The reliability curves which are formed before and after change events will result in a discrete software growth model with these change points. Thus, this study also includes a change point in the construction of the proposed software reliability growth model as a practical consideration.

Accordingly, this study considers both human factors and the nature of errors during the debugging process, such as imperfect debugging, the learning effect, error classification, and a change point, to construct a software reliability growth model that offers practical considerations. The objective of the study is to offer a more accurate software reliability growth model that can be a reference to decision-making for software developers and testing personnel. This study uses historical failure data to estimate the

☆ Editor: W. Eric Wong.

* Correspondence to: Department of Industrial and Information Management, National Cheng Kung University, 1 University Rd., Tainan 701, Taiwan.
E-mail address: yshuang@mail.ncku.edu.tw (Y.-S. Huang).

https://doi.org/10.1016/j.jss.2022.111267
0164-1212/© 2022 Elsevier Inc. All rights reserved.

parameters of the proposed model and compares the results with other software reliability models to ensure that the proposed model has a better fit than that of other existing models. By using the proposed model, the optimal timing at which software is released to the market can be obtained that is subject to the software reliability threshold and the testing cost.

2. Literature review

In the last five decades, software reliability is a crucial indicator for measuring software quality and can be a groundwork to scheme software testing works while developing software. Various software reliability growth models (SRGMs) have been proposed, and the statistic assumption of the Non-Homogeneous Poisson Process is the most widely adopted in the studies of classic SRGMs (Goel and Okumoto, 1979); Yamada, 1983; (Pham and Zhang, 2003; Huang, 2005; Chiu et al., 2008). Okamura et al. (2003) developed an EM algorithm for Goel and Okumoto's model to estimate the parameters. In their experiments, the developed algorithm is more efficient than Newton's method and Fisher's scoring method due to the global convergence property. Lai and Gard (2012) evaluated and discussed that NHPP is suitable to describe the failure counting process and applicable to the detection of errors in both calendar and execution times due to the time-dependent mean value function characteristic and thus are widely used in the area of software reliability. Accordingly, in recent years, the SRGMs based on NHPP have therefore been developed and adapted to more situations in practice. For example, Kapur et al. (2009) constructed an NHPP model to apply in a distributed development environment. Rafi and Rao (2010) incorporated logistic-exponential testing efforts in their proposed SRGM based on NHPP. Yadavalli et al. (2010) developed a software reliability model which can be described by NHPP with consideration of testing efforts. Huang and Lin (2010) take the concepts of testing compression factor into account and the quantified ratio of faults to failures in the proposed SRGM based on NHPP. Wang et al. (2016) proposed the theory of a general optimized NHPP model. Fang and Yeh (2016) discussed that the validity of confidence interval estimation between NHPP-based models and stochastic-based models. Besides, in order to improve prediction accuracy, an NHPP model with covariates was developed, which utilize extra variables to adjust the original prediction. Shibata et al. (2006) developed a framework to assess the quantitative software reliability with time-dependent covariate as well as software-fault data. Nagaraju et al. (2020) developed software reliability and security models with covariates to solve the allocation of testing activities. The model considers the relevant covariates to effectiveness of prediction. However, the studies based on NHPP above not only receive good performance in prediction but also easily adapt to different software testing environments. Therefore, the assumption of NHPP is taken to develop the proposed model in the study.

The aforementioned studies only considered situations for perfect debugging scenarios but most situations involve imperfect debugging scenarios in reality. It means that new errors might be still generated as the programmers were making corrections to defective codes. However, most of the related studies assume that once errors are detected, they can be removed immediately and will not result in any new errors, which is perfect debugging. However, some studies suggest that such an assumption is unrealistic and that debugging will tend to be imperfect since debugging staff may make mistakes while removing or correcting the software defects. Therefore, calibrating errors may result in new errors that are also possible causes of imperfect debugging. Shyur (2003) considered that new errors may introduce from the process of error removing, and thus it is necessary to take the assumptions of imperfect debugging into consideration in order to enhance the validity of SRGMs. In recent years, several studies have taken imperfect debugging assumptions into consideration to develop their SRGMs. For example, Pham (2007) considered the cause of new errors due to imperfect debugging and constructed a software reliability growth model examining the potential number of errors. Ahmad et al. (2010) and Raju (2011) constructed inflection S-shaped software reliability models based on an NHPP in which imperfect debugging had the probability of introducing new errors during the debugging process. Their proposed model is applicable to exponential and S-shaped types of data. Chatterjee et al. (2012) considered imperfect debugging by incorporating the time-dependent functions of error detection and error introduction develops a software reliability growth model based on an NHPP. Aktekin and Caglar (2013) modeled a multiplicative failure rate whose components evolved stochastically over testing stages. Their proposed model can account for imperfect debugging phenomena under certain conditions. Wang et al. (2015) proposed an imperfect debugging model with consideration of a log–logistic distribution fault content function. Wang and Wu (2016) proposed an NHPP-based imperfect software debugging model that considers the nonlinear fault introduction process. Li and Pham (2017, 2019) proposed an SRGM that considers the uncertainty of operating environments with imperfect debugging and testing coverage. They utilized a random-distributed variable to simulate the uncertainty of operating environments. Pavlov et al. (2018a,b) proposed a Hausdorff approximation of an impulse function based on Pham's SRGM. The proposed approaches can illustrate the possibility of equitable use of the supersaturation in combination with the confidence intervals of the mean value function. Zhao et al. (2018) adopted a Bayesian approach to the treatment of uncertainty of software reliability with consideration of perfect and imperfect factors. Saraf and Iqbal (2019) developed an imperfect SRGM for dealing with the issue of multiple software releases and change points. The above studies had successfully taken imperfect debugging into different issues to extend applications of SRGMs with efficiency. However, most of them did not consider that the learning effect and the error fluctuation rate may be related to the potential errors produced, and therefore we take these factors into consideration for measuring the new produced errors due to imperfect debugging.

Moreover, the testing staff can acquire learning effects on the account of the accumulated experience of testing and debugging works. More and more studies discussed how the learning effects can elevate the efficiency of the detection and removal of errors. However, each study has its distinctive perspectives on the learning effect of testing works. Xia et al. (1992) is an early researcher for studying a learning factor in software reliability growth, and they consider the learning effect is simply due to testing staffs' experience. Hou et al. (1994) utilized a logistic learning factor to propose a model that can describe why the mean value function can present an exponential or S-shaped learning curve. Chatterjee et al. (1997) proposed a model which can incorporate testing effort with the learning effect, and they consider that such the joint effect can truly reflect the phenomenon of software reliability growth. Kapur et al. (2004) proposed a related model with a similar concept, and the effect of testing effort is described as a logistic function. Moreover, they assume that the software system consists of a finite number of reused and newly developed sub-systems. Chiu et al. (2008) stated that both the capability of debugging personnel and learning effects may influence error removal efficiency and constructed an NHPP-based SRGM with a learning effect which shows a good fit with software failure data compared with other previous S-shaped and exponential models. Ho et al. (2008) reconstruct the SRGM by stochastic differential equations with the same ideas. Their model can determine the

optimal software release time at different confidence levels for software developers with consideration of the learning effect. Kapur et al. (2013) adopted the control-theoretic approach to investigate the dynamic optimal allocation of the testing effort. They also considered the experience of learning curve effect to control for the testing effort for managers with an aim to determine the maximum number of errors at the minimum cost. Duffey and Fiondella (2014) investigated the evidence that the learning trends exist in software testing works or not, and they concluded that the learning trends may not be presented in all the testing data. However, if the learning effect exists in testing projects, the proposed model can measure the learning performance from the test staffs' experience and skill acquisition. Zhu and Pham (2017) consider that testing environment is the most significant factor to influence the reliability on the software development project. The staff's domain knowledge and education can influence the effectiveness of software testing projects. Lemos et al. (2018) conducted experiments to survey the impact of testing knowledge and experience in terms of software reliability. Chiu et al. (2019) further assumed that the learning effect is time-dependent and may linearly or exponentially increase over time and constructed an NHPP SRGM to simultaneously describe S-shaped and exponential software testing data. In addition, software reliability growth models with learning effects have been applied to decision-making. Since the above studies did not consider the unstable learning effect at the beginning of testing work, a cyclic function will be proposed for describing the phenomenon that the debugging efficiency will be gradually stable as the testing time elapses.

Most of the related studies simply consider that each error-removal task will consume the same testing effort or resource but it may be unrealistic in practice. Generally, each error-removal task requires a different processing time for correction from different degrees of error complexity. Accordingly, a part of the related studies had begun to revise the assumption of a single error type for improving the practicality of SRGMs. Kapur et al. (2007) classified errors as simple, difficult, and complex and assumed that the error detection rates for different types of errors may be different under multiple change points. Kapur et al. (2010) used the stochastic differential equation to construct a software reliability growth model taking into consideration the severity of errors and testing effort. Jain et al. (2012) constructed a software reliability model with consideration of two-type occurring during the debugging process. Garmabaki et al. (2011) considered upgrading software and error conditions with different degrees of severity to construct software reliability models for different software versions, in which the number of error removals in each version included the detected errors in the current version and those left from previous versions. Khatri and Chhillar (2012) also classified errors as simple, difficult, and complex according to the degree of detection and removal difficulty and then constructed software reliability growth models with perfect and imperfect debugging for object-oriented software systems, respectively. Kaushal and Khullar (2012) stated that understanding the causes of errors or estimating levels of severity before removing errors is beneficial for planning and managing the software development process and saving time and costs. Since the calibration of a complex error requires more time than that of a simple one does, a unified standard of the estimated cost and time for different types of errors may lead to the miscalculation of testing projects' budget. Furthermore, software errors are always diverse in practice, and therefore they should be classified into different types according to their complexity. Generally, the time for correcting different errors varies, and therefore it will be considered that the expected time of correcting different error types is assumed to follow a specified probability distribution with different parameters in the study.

Most SRGMs simply assume that software developers will not change or adjust the original testing resource or environment throughout the testing phase. This assumption can keep the efficiency of testing work remain unchanged to estimate the variation of software reliability at different time points more easily. However, it may be unrealistic in practice because software developers may change their testing policy to accelerate testing works in consideration of business opportunities. The changes in the policies related to software testing, debugging personnel, and allocation of testing resources and efforts influence the error detection rate, and therefore it may result in the issue of change point. Some studies had perceived the issue and tried to model it to adapt to the situation of change point. Kapur et al. (2009) constructed a generalized software reliability growth model with consideration of change points for applying in a more practical debugging condition. Huang and Lyu (2011) further considered multiple change points and used the weighted arithmetic mean, the weighted geometric mean, and the weighted harmonic mean to derive the software reliability growth models proposed in previous studies. Their model is applicable to both the stages of software testing and the real implementation, and is also able to describe exponential and S-shaped data. Jain et al. (2014) proposed an SRGM with Weibull type testing effort function. Their model can deal with situations of multiple change points and imperfect debugging. Inoue et al. (2015) proposed an SGRM to deal with an all-stage truncated multiple change-point environment. Inoue et al. (2016) proposed a bivariate SRGM with a change point depending on testing effort with consideration of the uncertainty of a testing environment. Inoue and Yamada (2018) utilized a Markovian modeling approach to evaluate software reliability with the impacts of change-point and imperfect debugging environment. Wang et al. (2019) also proposed an entropy Markov model for real-time reliability prediction in software systems, and a Bayesian method was used to model the conditional probabilities. Dadkhah et al. (2020) studied semantic technology can support software testing in both industry and academia, and it indicated that software testing is a knowledge-intensive process. Although the above studies received good performance in the prediction from the application of change point, the integration of multiple issues with change point may be insufficient in the related studies. Accordingly, the study will integrate the change point with other critical issues for enhancing the application.

Based on the above discussions of all the issues, this study devises a software reliability growth model based on an NHPP with a more practical consideration of imperfect debugging and learning effects with respect to different error classifications. In addition, we also consider the possibility that the replacement of testing teams or changes in the testing environment may result in different reliability growth curves with respect to the changing events, i.e., change points. Previous studies have rarely considered all of the aforementioned factors, and learning effects and error classification have not been extensively discussed. This study considers both human factors and the nature of errors to propose a practical model that can estimate software reliability more accurately. Software failures may result in disasters for large-scale software systems, such as air traffic control systems, financial information systems, battle command systems, and medical diagnostic systems. Thus, the removal of potential software errors is important with regard to increasing software reliability. The proposed model can assist software developers, such as large-scale software development companies, software project managers, and software developers with making more effective decisions when developing software.

3. Model development

Assuming that a software development company has developed a software program and has tested it to detect all the potential errors, a. Suppose that the number of detected errors can be described by an NHPP model, i.e., the number of detected errors will change over time; $\lambda(t)$ denotes the expected number of detected errors in the software at time t, and $m(t)$ denotes the expected cumulative number of detected errors from time 0 to t. However, due to the fact that the complexity of the detected errors is divergent in reality, the effort of removing a complex error will certainly take more time than that of removing a simple one. Accordingly, the traditional assumption of a single type of errors is unrealistic, and therefore leads to the misevaluation of debugging cost. Thus, the errors are classified as either complex or simple ones in the study, and the proportion of different types of errors can be reasonably estimated from historical data.

Generally, the experiences of the repeatable detecting and debugging errors, which leads to learning effects, can be accumulated for the testing staff. Boh et al. (2007) stated that software development is an experience-oriented activity in which engineers focus on the development of a system and thus can have a comprehensive understanding of the design of the system that will accelerate the software development process. However, the effects of learning may be unstable, since the error removal process is not perfect, which may cause a fluctuation of errors in the system. The error fluctuation rate may experience dramatic fluctuations in the beginning phase of errors detection. As more errors have been removed, the learning effects of the debugging team are enhanced, and the error fluctuation rate will become stable in the later phase of detection, which results in a conversion of the lower variability in the number of errors. In addition, imperfect debugging may also be caused by an inability to immediately remove detected errors which can be characterized by debugging efficiency, $p(t)$, to indicate the probability that an error can be immediately removed. This study considers debugging efficiency to also be affected by learning effects.

Investigations of error classification, learning effects, and the debugging efficiency of testing staff suggest that the personnel, resources, and environment that are related to software development may change over time. Suppose that an environmental change occurs that affects the debugging efficiency of the testing staff and that the error detection rate, $d(t)$, will change at change point τ. Since a software development company has limited resources, the contractual level of reliability has to be achieved at the minimum cost, and then the software program can be released to the market or turned to the customer. Assume that the setup cost of the detection process is denoted as C_0, which includes the costs for detection plans, equipment, and personnel. C_1 denotes the daily costs per unit time during the whole testing period which include the office rental, insurance, and utility fees. $C_{2,k}$ denotes the cost of removing a type k error per unit time during the testing period, and $C_{3,k}$ denotes the cost of removing a type k error per unit time during the warranty period. C_4 denotes the failure cost after the software program has been released, and C_5 denotes the opportunity loss of delaying the release of the software to the market. This study places constraints on the contractual level of reliability, R_0, to determine the optimal timing for releasing the software, T^*, with an aim to minimize the total cost, $C(T)$.

The assumptions of this study are as follows:

(1) The counting process for detected errors follows the NHPP.
(2) Software errors can be classified as simple or complex. A complex error requires longer removal time than a simple error does, and the removal times can be described as truncated exponential distributions with different parameters.
(3) It is possible to introduce new software errors when the detected errors are removed and corrected.
(4) The error fluctuation rate can be described by a sine cyclical function with a decreasing fluctuation breadth during the detection period, which indicates that the influence of increasing of new errors during the testing period will be gradually stable as the testing time elapses.
(5) The occurrence of the change point is known.

Based on the above discussion, the study proposes a software reliability growth model with imperfect debugging in which the software reliability is determined by the number of errors that are detected and removed by the testing staff during the testing period. Within the testing period, learning effects emerge because the testing staff repeatedly performs error detection and removal tasks. However, errors in the software program are not exactly the same and thus can be classified according to their complexity. In addition, a change in testing staff, hardware, and resources may impact the debugging efficiency and then the error detection rate, which results in a change point in the proposed model. This study also considers the necessary costs during the testing period to determine the optimal release time of the software program to the market by which acceptable software reliability is achieved with the minimum cost.

The notations used in this study are as follows:

a	The total number of errors initially in the software program
$a(t)$	The total number of errors in the software program from time 0 to t
$N(t)$	The counting process for the cumulative number of detected errors from time 0 to time t
$m_j(t)$	The expected cumulative number of detected errors from time 0 to time t for condition j (i.e., the mean function), where $j = b, f$ denotes the conditions before and after the change point, respectively.
$\lambda_j(t)$	The expected number of detected errors under condition j at time t (i.e., the intensity function).
$d_j(t)$	The average error detection rate at time t under condition j, where $j = b, f$, that is, the testing ability of the testing staff at time t under condition j.
$\gamma(t)$	The error fluctuation rate at time t
$g(t)$	The remaining number of errors in the software program at time t
τ	The time of the occurrence of the change point
$p(t)$	Debugging efficiency
$\eta(t)$	The learning effect at time t
q	The proportion of simple errors and the proportion of the complex errors is thus $(1-q)$
$G_k(\theta_k, \xi_k)$	The probability density function of the time for correcting a type k errors ($k = s$ (simple errors) or $k = c$ (complex errors));
t_r	The time required for performing a correction of software error
ξ_k	The pre-specified upper time limit for performing a correction of software error type k
θ_k	The parameter of the probability density function $G_k(\theta_k, \xi_k)$; $\hat{\theta}_k$ denotes the estimator of θ_k
$E[\theta_k, \xi_k, t_r]$	The expected time for performing a correction of software error type k under parameters ξ_k and θ_k
R_0	The contractual level of reliability
$C(T)$	The total cost
C_0	The setup cost of the testing process
C_1	The daily costs per unit time during the whole testing and warranty period
$C_{2,k}$	The cost of removing a type k error per unit time during the testing period, where $k = s, c$

$C_{3,k}$ The cost of removing a type k error per unit time during the warranty period, where $k = s, c$
C_4 The risk cost which can be described as the loss if an error occurs after the software program has been released to the market
C_5 The opportunity loss of delaying the release of the software program to the market
SC The setup cost during the testing period includes the fixed costs for detection
RC The daily administrative cost during the testing and warranty period
EC The cost of error removal during the testing process
WC The costs of removing errors during the warranty period
FC The cost of losing reputation due to the occurrence of any failure
OC The time-dependent opportunity loss of the delayed time
T The release time of the software program to the market
T_w The warranty period of the software program
T^* The optimal release time of the software program to the market
r The market interest rate

3.1. Model construction

The software debugging process is a process in which errors are detected and counted. It can be described by an NHPP model, in which the cumulative total number of errors detected in a software program is $\{N(t), t \geq 0\}$ and its expectation is given by

$$E\{N(t)\} = m(t) = \int_0^t \lambda(s)\, ds, \tag{1}$$

where $m(t)$ is the mean function, i.e., the expected cumulative total number of detected errors from time 0 to t, and $\lambda(t)$ is the intensity function, i.e., the expected number of detected errors at time t. The probability function of the NHPP model is given by

$$P(N(t) = k) = \frac{[m(t)]^k \cdot e^{-m(t)}}{k!}, k = 0, 1, 2, 3, \ldots. \tag{2}$$

The conditional software reliability, $R(t_{op}|t)$, is where no error is detected between time t and $(t + t_{op})$ conditional on no errors being detected before t, and which is

$$R(s|t) = e^{-[m(t+t_{op})-m(t)]}. \tag{3}$$

The error detection rate is often used to investigate the percentage of error detection under the remaining errors in a system, and it can evaluate or reflect the efficiency of debugging task at time point t. The following error detection rate was proposed by Yamada et al. (1986). It presents an exponential decay trend, and its form is given as follow:

$$d(t) = \alpha\beta e^{-\beta t}. \tag{4}$$

Besides, the original mean value and intensity functions $m(t)$ and $\lambda(t)$ can be deduced by differential equation $\lambda(t) = dm(t)/dt = d(t)(a - m(t))$ from the detection rate with the condition of the initial error number a at time $t = 0$, and therefore the mathematical forms of $m(t)$ and $\lambda(t)$ can be seen as follows:

$$m(t) = a\left(1 - e^{-\alpha(1-e^{-\beta t})}\right) \text{ and } \lambda(t) = a\alpha\beta e^{(e^{-\beta t}-1)\alpha-\beta t}. \tag{5}$$

It should be noted that the above error detection rate has taken testing resources and efforts into consideration, such as human resources, budget, and beneficial facilities. Parameter α denotes the extent to which the resources are launched during the testing process, and parameter β is the scale factor of the exponential function. The testing resources and efforts before and after the time change point τ are differently denoted as α_b and α_f, respectively. The original error detection rate can be rewritten as follow:

$$d_j(t) = \begin{cases} \alpha_b\beta e^{-\beta t}, & 0 < t \leq \tau \\ \alpha_f\beta e^{-\beta t}, & t > \tau \end{cases}, \quad j = b, f. \tag{6}$$

Accordingly, the instantaneous expected number of detected errors with imperfect debugging is often given by

$$\lambda(t) = \frac{dm(t)}{dt} = d_j(t)\left(a_j(t) - x_j(t)\right), \tag{7}$$

where $x_j(t)$ denotes the accumulated number of removed errors at time t, and $a_j(t) - x_j(t)$ denotes the number of remaining errors at time t. Note that $a_j(t)$ denotes the total number of errors in the software system from time 0 to time t, which will be affected by an increase in new errors and a decrease in existing errors due to imperfect debugging and is given by

$$\frac{da_j(t)}{dt} = \gamma(t)\lambda_j(t). \tag{8}$$

By multiplying the error fluctuation rate, $\gamma(t)$, with the instantaneous expected number of detected errors, $\lambda_j(t)$, we can obtain the instantaneous change in the total number of errors in the software program. The error change is the proportion of changes in the number of errors in the software program due to the debugging behavior of the testing staff. This study uses a sine potential error fluctuation rate, which is given by

$$\gamma(t) = \exp[-\eta(t)]\sin\left(\frac{\pi\delta t}{2}\right). \tag{9}$$

The sine function is exponentially convergent with learning effect and time. The additional change in the number of errors will eventually approach zero. The parameter δ is used to describe the characteristics of the software development project, since whether or not the testing staff is familiar with the characteristics of the software development project will affect the error fluctuation rate of the software program. When the project is elusive at the beginning, the testing staff will spend more time dealing with debugging work. The error fluctuation rate will be difficult to be stabilized, and $\gamma(t)$ will present a higher frequency. Once the software project becomes more comprehensible, the testing staff only spend less time dealing with the codes because they become more familiar with the software project. Accordingly, the error fluctuation rate will be relatively easy to be stabilized, and $\gamma(t)$ will present a lower frequency. Moreover, the learning factor also affects the error fluctuation rate during the testing process. This study uses the time-dependent learning effect, which indicates that the testing staff can use previous testing experiences to improve the subsequent error detection and removal tasks. The learning effect function is given by

$$\eta(t) = \eta t, \tag{10}$$

where η denotes the learning factor. Note that the definition of the learning effect function can be changed according to the investigation from historical data. In other words, the fluctuation is a result of some program codes being implemented improperly that must be corrected by the debugging staff. Since members of the testing staff are not initially familiar with the program codes, additional errors will be more likely to emerge at the beginning of the debugging process. After they become more familiar with the program codes, removing errors may also reduce the number of existing errors later. When errors have been removed singly or

$$a_j(t) = a\left(1 + \frac{2\alpha_j\beta e^{\alpha_j(e^{-\beta t}-1)-(\beta+\eta)t}\left(\delta\pi\left(e^{\eta t} - Cos\left[\delta\pi t/2\right]\right) - 2\eta Sin\left[\delta\pi t/2\right]\right)}{\delta^2\pi^2 + 4\eta^2}\right) \text{ and} \tag{11}$$

$$m_j(t) = a\left(1 + e^{\alpha_j(e^{-\beta t}-1)}\left(\alpha_j\beta e^{-\beta t}\left(t + \frac{e^{-\eta t}-1}{\eta} + \frac{\left(2\delta\pi\left(Cos\left[\delta\pi t/2\right] - e^{\eta t}\right) + 4\eta Sin\left[\delta\pi t/2\right]\right)e^{-\eta t}}{\delta^2\pi^2 + 4\eta^2}\right) - 1\right)\right). \tag{12}$$

Box I.

jointly, some errors may still remain to cause additional errors due to the removal and correction process, such as errors in computer syntax, compiling, logic, and implementation, which will all have to be fixed in the software program.

Based on the above discussion, the total number of errors in the software $a_j(t)$ from time 0 to t can be derived by the differential equations (7), (8), and (9) simultaneously with the marginal condition $a_j(0) = a$ and $m_j(0) = 0$. The forms of $a_j(t)$ and $m_j(t)$ can be given respectively as in Box I.

Moreover, imperfect debugging also considers that even if an error is detected, it cannot be guaranteed to be removed immediately. The debugging efficiency, $p(t)$, is the probability that an error can be corrected without producing new errors when it is detected, which will affect the number of removed errors, where their relationship is given by

$$\frac{dx_j(t)}{dt} = p(t)\,\lambda_j(t). \tag{13}$$

Previous studies have often considered debugging efficiency to be a constant by assuming that it remains the same during the entire testing period for the same testing staff. However, debugging efficiency may change with time. Thus, this study assumes that debugging efficiency gradually wanes over time and is affected by the learning effect, which is given by

$$p(t) = 1 - e^{-\eta t}. \tag{14}$$

According to Eqs. (13) and (14), the accumulated number of removed errors $x_j(t)$ from time 0 to t can be derived by differential equation methods with the marginal condition $x_j(0) = 0$. The forms of $x_j(t)$ can be given by

$$x_j(t) = a\left(1 + \left(\alpha_j\beta\left(\frac{e^{-\eta t}-1}{\eta} + t\right)e^{-\beta t} - 1\right)e^{\alpha_j(e^{-\beta t}-1)}\right). \tag{15}$$

Furthermore, we use $dg_j(t)/dt$ to denote the intensity for errors removed and also unexpected errors produced at time t, so the differential equation is given by

$$\frac{dg_j(t)}{dt} = \frac{da_j(t)}{dt} - \frac{dx_j(t)}{dt}. \tag{16}$$

Given the initial condition $g_j(0) = a$, the remaining number of errors in a software at time, t, will be shown as follows:

$$g_j(t) = a\left(1 - \left(\frac{e^{-\eta t}-1}{\eta} + t\right)\alpha_j\beta e^{-\beta t} - \frac{2\alpha_j\beta\left(\delta\pi\left(Cos\left[\delta\pi t/2\right] - e^{\eta t}\right) + 2\eta Sin\left[\delta\pi t/2\right]\right)e^{-(\beta+\eta)t}}{\pi^2\delta^2 + 4\eta^2}\right)e^{\alpha_j(e^{-\beta t}-1)}. \tag{17}$$

The detection rate of the new definition can be obtained as follow:

$$d_j(t) = \frac{\frac{dm_j(t)}{dt}}{g_j(t)} = \frac{\alpha_j\beta e^{-(2\beta+\eta)t}}{\eta\Upsilon_1}\left\{\begin{array}{l}\alpha_j\beta\left(\frac{2\Upsilon_2}{\beta} + (1-\eta t)\Upsilon_1\right)e^{\eta t} \\ \quad +(\Upsilon_1 + 2\Upsilon_2)(2\eta + \beta(1-\eta t))e^{(\beta+\eta)t} \\ \quad -\Upsilon_1(\alpha_j\beta + (\beta+\eta)e^{\beta t}) \\ -2\Upsilon_2(\alpha_j + e^{\beta t})Cos\left[\delta\pi t/2\right] \\ \quad -\eta\left(4\alpha_j\beta\eta + (\Upsilon_1 + 4\beta\eta)e^{\beta t}\right)Sin\left[\delta\pi t/2\right]\end{array}\right\}, \tag{18}$$

where $\Upsilon_1 = \delta^2\pi^2 + 4\eta^2$ and $\Upsilon_2 = \beta\delta\eta\pi$. The above model can be used to deal with the issue with a single type of software errors. However, software errors are diverse in practice and should be classified into different types according to their complexity. Consequently, in dealing with the problem of two error types (c: complex, s: simple), the mean value function above can be extended and rewritten as follows:

$$m_j(t) = \sum_{k\in s,c} m_{j,k}(t) \Rightarrow \left\{\begin{array}{l} m_b(t) = m_{b,s}(0,t) + m_{b,c}(0,t) \\ \quad \Rightarrow \left\{\begin{array}{l} m_{b,s}(0,t) = qm_b(t) \\ m_{b,c}(0,t) = (1-q)m_b(t)\end{array}\right. , 0 < t \le \tau \\ m_f(t) = m_{f,s}(\tau,t) + m_{f,c}(\tau,t) \\ \quad \Rightarrow \left\{\begin{array}{l} m_{b,s}(\tau,t) = q\left(m_f(t) - m_f(\tau)\right) \\ m_{b,c}(\tau,t) = (1-q)\left(m_f(t) - m_f(\tau)\right)\end{array}\right. , t > \tau \end{array}\right. , \tag{19}$$

where q and $(1-q)$ denote the estimated proportions of simple and complex errors respectively. It should be noted that the mean value function within the time period (t_b, t_a) can be calculated by $m_{j,k}(t_b, t_a) = m_{j,k}(t_b) - m_{j,k}(t_a)$.

However, the mean value function is the expected value of the accumulated errors detected, the managers cannot recognize the potential bias of the estimation. Accordingly, it would be necessary to estimate the possible range of software errors at a certain confidence level (usually use 90% or 95%) for providing sufficient information to the managers if they want to make a conservative or compromise decision. Yamada and Osaki (1985) presented a method to calculate the confidence bounds of the mean value function, and it can be estimated as:

$$\hat{m}_j(t) \pm Z_{CR/2}\sqrt{\hat{m}_j(t)}, \tag{20}$$

where $\hat{m}_j(t)$ is the estimated value of $m_j(t)$ which value and parameters can calculated from the sample dataset. CR denotes the critical region, and $Z_{CR/2}$ is the critical value of a given area $CR/2$ of the standard normal distribution. Since the standard deviation $\sqrt{\hat{m}_j(t)}$ is positively correlated with testing time, the corresponding confidence interval is amplified with testing time.

Since the calibration of a complex error requires more time than that of a simple one does, the expected time to correct different error types is assumed to follow a truncated exponential

distribution which is as follow:

$$G_k(\theta_k, \xi_k)dt_r = \left(1 - e^{-\xi_k/\theta_k}\right)^{-1} {\theta_k}^{-1} e^{-t_r/\theta_k}, \quad 0 < t_r < \xi_k \text{ and } 0 < \theta_k < \xi_k, \tag{21}$$

where t_r denotes the processing time for a correction from a software error; ξ_k is the pre-specified upper time limit for performing a correction of software error type k, and the upper time limit of complex errors is greater than that of simple errors ($\xi_c > \xi_s$) due to the difficulty of error correction. θ_k is the parameter for the average processing time for error correction of type k, and the value of the estimator $\hat{\theta}_k$ can be obtained by the maximum likelihood estimation method. Firstly, we take the natural logarithm for the likelihood function of the truncated exponential distribution, and the following equation can be obtained:

$$\ln\left[\prod_{i=1}^{n}\left(1 - e^{-\xi_k/\theta_k}\right)^{-1}{\theta_k}^{-1}e^{-t_r^{(i)}/\theta_k}\right] = -n\ln\left[1 - e^{-\xi_k/\theta_k}\right] - n\ln[\theta_k] - \sum_{i=1}^{n} t_r^{(i)}/\theta_k, \tag{22}$$

where $t_r^{(i)}$ denotes ith sample of the processing time for a correction from the historical testing data ($\mathbf{D}^{(k)}$={$t_r^{(1)}$, $t_r^{(2)}$, $t_r^{(3)}$,..., $t_r^{(n)}$}). Secondly, we set the first-order derivative of the log likelihood function with respect to θ_k to be zero, which is given by:

$$\frac{d\left(n\ln\left[1 - e^{-\xi_k/\theta_k}\right] - n\ln[\theta_k] - \sum_{i=1}^{n} t_r^{(i)}/\theta_k\right)}{d\theta_k} = \frac{n\xi_k e^{-\xi_k/\theta_k}}{\left(1 - e^{-\xi_k/\theta_k}\right){\theta_k}^2} - \frac{n}{\theta_k} + \frac{\sum_{i=1}^{n} t_r^{(i)}}{{\theta_k}^2} = 0. \tag{23}$$

Solving Eq. (23) for θ_k by numerical methods can obtain the estimator, $\hat{\theta}_k$, and the expected time for performing a correction of software error type k under the parameters, ξ_k and $\hat{\theta}_k$, can be estimated by the following equation:

$$E\left[\hat{\theta}_k, \xi_k, t_r\right] = \int_0^{\xi_k} t_r G_k(\hat{\theta}_k, \xi_k)dt_r = \int_0^{\xi_k} t_r\left(1 - e^{-\xi_k/\hat{\theta}_k}\right)^{-1}\hat{\theta}_k^{-1}e^{-t_r/\hat{\theta}_k}dt_r. \tag{24}$$

A numerical integration method is required to obtain the value of Eq. (24) due to no closed-form solution.

3.2. Parameter estimation

In this study, collected software failure data is used to demonstrate that the proposed model has a better fit than that of other existing models. Two estimation methods can be adopted to estimate the model parameters.

(1) The lease square estimation method (LSE) uses n pairs of observed data, i.e., (t_0, m_0), (t_1, m_1), (t_2, m_2), ..., (t_n, m_n), to estimate all parameters of the proposed model, in which m_h is the total number of detected errors within $(0, t_h)$, and the approach is given by

$$Min\ M\left(a, \alpha_j, \beta, \eta, \delta\right) = \sum_{h=1}^{n}\left(m_h - m\left(t_h\right)\right)^2. \tag{25}$$

By letting the first order derivative of Eq. (24) with respect to the parameters, $a, \alpha_j, \beta, \eta,$ andδ be 0, we can use numerical methods to solve the simultaneous equations, which are given by

$$\frac{\partial M\left(a,\alpha_j,\beta,\eta,\delta\right)}{\partial a} = \frac{\partial M\left(a,\alpha_j,\beta,\eta,\delta\right)}{\partial \alpha_j} = \frac{\partial M\left(a,\alpha_j,\beta,\eta,\delta\right)}{\partial \beta} = \frac{\partial M\left(a,\alpha_j,\beta,\eta,\delta\right)}{\partial \eta} = \frac{\partial M\left(a,\alpha_j,\beta,\eta,\delta\right)}{\partial \delta} = 0. \tag{26}$$

(2) The maximum likelihood estimation method (MLE) can also be used to estimate the parameters of the proposed approach, and the likelihood function is given by

$$L = \Pr\{N(t_1) = m_1, \ldots, N(t_n) = m_n\} = \prod_{h=1}^{n}\frac{\left(m\left(t_h\right) - m\left(t_{h-1}\right)\right)^{m_h - m_{h-1}}\left(e^{-\left(m(t_h) - m(t_{h-1})\right)}\right)}{(m_h - m_{h-1})!}. \tag{27}$$

By taking the natural logarithm of the likelihood function, we can have

$$\ln[L] = \sum_{h=1}^{n}(m_h - m_{h-1})\ln\left(m\left(t_h\right) - m\left(t_{h-1}\right)\right) - \sum_{h=1}^{n}\left(m\left(t_h\right) - m\left(t_{h-1}\right)\right) - \sum_{h=1}^{n}\ln\left((m_h - m_{h-1})!\right). \tag{28}$$

Likewise, by letting the first order derivative of the log likelihood function with respect to the parameters be zero, we can obtain the parameter values, which are given by

$$\frac{\partial \ln[L]}{\partial a} = \frac{\partial \ln[L]}{\partial \alpha_j} = \frac{\partial \ln[L]}{\partial \beta} = \frac{\partial \ln[L]}{\partial \eta} = \frac{\partial \ln[L]}{\partial \delta} = 0. \tag{29}$$

Since the derivation of the MLE is more complex than that of the LSE, in this study, the LSE is used to estimate the parameter values.

3.3. Optimal software release policy

Since resources are limited for the software development project, the acceptable software reliability should be achieved with the minimum cost before releasing the software program to the market. The setup cost during the testing period includes the fixed costs for detection, such as the initial cost in testing planning, equipment, and preparation works, which is given by $SC = C_0$. The daily administrative cost during the testing and warranty period is given by $RC = C_1\int_0^{T+T_w} e^{-rt}dt$, where C_1 denotes the daily cost per unit time for the detection process and the warranty period, such as office rental, insurance, and utilities fees; T denotes the software program testing time; T_w denotes the software program warranty period, and e^{-rt} denotes the time discounting process based on the market interest rate, r.

The cost of error removal during the testing process is given by $EC = \sum_{k=s,c} C_{2,k}m_b(\tau)\,E\left[\hat{\theta}_k, \xi_k, t_r\right] + \sum_{k=s,c} C_{2,k}m_f(T)\,E\left[\hat{\theta}_k, \xi_k, t_r\right]$, where $C_{2,k}$ denotes the cost of removing a simple or complex error per unit time; $E\left[\hat{\theta}_k, \xi_k, t_r\right]$ denotes the average required time to remove an error of type k in the testing process. The cost of removing errors during the warranty period is given by $WC = \sum_{k=s,c} C_{3,k}\left(m_f(T + T_w) - m_f(T)\right)E\left[\hat{\theta}_k, \xi_k, t_r\right]$, where $C_{3,k}$ denotes the cost of removing a simple or complex error per unit time during the warranty period and $m_f(T + T_w) - m_f(T)$ is the expected number of removed errors during the warranty period.

After the software program is released to the market, the cost of losing reputation due to the occurrence of any failure is given by $FC = C_4\left(1 - R\left(t_{op}|T\right)\right)$, where C_4 denotes the reputation loss, and $1 - R(t_{op}|T)$ is the probability of the occurrence of failures during the warranty period after the software program is released to the market. Suppose that the delay of the release of the software products may result in both tangible and intangible losses. For instance, an opportunity loss will occur if other similar software programs are introduced to the market earlier, which indicates that the longer the delay, the more the loss. Thus, the time-dependent opportunity loss of the release time delay has to be considered. Zeephongsekul and Chiera (1995)

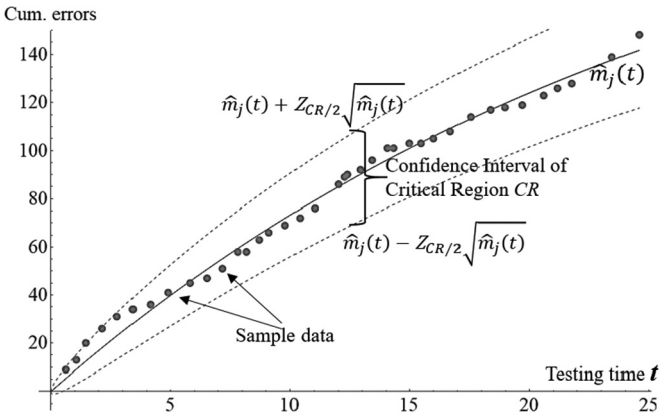

Fig. 1. Testing time vs. confidence interval of the mean value function.

proposed a game model for describing the opportunity loss in decision making of software release. The early model can give the important concept that the managers need to consider the potential loss if they delay the software release in the market. In this study, the assumption of the opportunity loss is referring by Fang and Yeh (2016), and it can present the monotonically increasing trend with the time delay, which is given by $OC = C_5(T) = v_0\,(v_1 + T)^{v_2}$, where v_0 is the scale factor; v_1 is the intercept value, and v_2 is the increasing degree of opportunity loss over time.

By adding up all the costs incurred during the testing process, we can obtain the total cost, which is given by

$$\begin{aligned} C\,(T) &= SC + RC + EC + WC + FC \\ &= C_0 + C_1 \int_0^{T+T_w} e^{-rt} dt + \sum_{k=s,c} C_{2,k} m_b\,(\tau)\, E\left[\hat{\theta}_k, \xi_k, t_r\right] \\ &\quad + \sum_{k=s,c} C_{2,k} m_f\,(T)\, E\left[\hat{\theta}_k, \xi_k, t_r\right] \\ &\quad + \sum_{k=s,c} C_{3,k}\left(m_f\,(T+T_w) - m_f\,(T)\right) E\left[\hat{\theta}_k, \xi_k, t_r\right] \\ &\quad + C_4\left(1 - R\left(t_{op}\,|T\right)\right) + C_5(T) \end{aligned} \tag{30}$$

Therefore, this problem can be formulated as:

$$\begin{aligned} Min\ C\,(T) &= C_0 + C_1 \int_0^{T+T_w} e^{-rt} dt \\ &\quad + \sum_{k=s,c} C_{2,k} m_b\,(\tau)\, E\left[\hat{\theta}_k, \xi_k, t_r\right] \\ &\quad + \sum_{k=s,c} C_{2,k} m_f\,(T)\, E\left[\hat{\theta}_k, \xi_k, t_r\right] \\ &\quad + \sum_{k=s,c} C_{3,k}\left(m_f\,(T+T_w) - m_f\,(T)\right) E\left[\hat{\theta}_k, \xi_k, t_r\right] \\ &\quad + C_4\left(1 - R\left(t_{op}\,|T\right)\right) + C_5(T) \end{aligned} \tag{31}$$

$$Subject\ to: R\left(t_{op}\,|T + T_w\right) \geq R_0, \tag{32}$$

where Eq. (31) is the objective function, and Eq. (32) is the constraint of the acceptable software reliability. Since the objective function is nonlinear and extremely complex, a spectrum analysis and numerical methods will be used to obtain the optimal solution with the constraint. By solving the programming problem, we can obtain the optimal time at which to release the software program that provides the most acceptable reliability with the minimum cost.

Since $a, \alpha_j, \beta, \eta, \delta, T, r, v_0, v_1$, and v_2 are all positive numbers, the objective function, $C(T)$, can strictly increase, strictly decrease, or decrease first and then increase with the reflection point at T^*. The three cases are described as follows:

(1) If $C(T)$ is strictly increasing, T_{R_0} is the time at which the most acceptable software reliability is reached. The optimal release time of the software program to the market, T^*, would then be T_{R_0} because the acceptable software reliability has been achieved at the minimum cost. Case I of Fig. 2 illustrates the situation.

(2) If $C(T)$ is strictly decreasing, it can be presented like Case II in Fig. 1. In such a case, the optimal release time of the software program to the market, T^*, will approach infinity. However, this is not practical, since the costs increase with the testing time, such as the daily cost and error removal cost, and the cost savings since the improvement in software reliability may not be able to offset the expenditures incurred during the testing process.

(3) If $C(T)$ is convex, it will decrease first and then increase, and the lowest point is the release time of the software program to the market, T^*. By using a spectrum analysis to identify the optimal release time point, T_{S_0}, we can assume that T_{R_0} is the time at which the minimum acceptable software reliability is reached. As can be seen in Case III in Fig. 2, if $T_{S_0} \geq T_{R_0}$, the release time of the software program is $T^* = T_{S_0}$ since although the threshold of the reliability has been reached, the minimum cost is not satisfied. On the other hand, if $T_{S_0} \leq T_{R_0}$, the release time of the software program is $T^* = T_{R_0}$ because the software program fails to reach the acceptable reliability at T_{S_0} and still has to be tested. For instance, if the time required to reach the acceptable software reliability is T'_{R_0}, since $T_{S_0} \geq T_{R_0}$, the release of the software program to the market is T_{S_0}. If the time required to reach the acceptable software reliability is T_{R_0}", since $T_{S_0} \leq T_{R_0}$, the release time of the software program is T_{R_0}. Case III in Fig. 2 illustrates the situation.

4. Model verification and comparison

The four most commonly evaluation criteria used in previous studies for different models are selected to validate the effectiveness of the proposed model. The following evaluation criteria are as follows:

(1) Mean square error (MSE) evaluates the difference between the estimated and true values, i.e., $MSE = \frac{\sum_{h=1}^{n}(m_h - m(t_h))}{n-k}$, where m_h is the cumulative number of detected errors from time 0 to m_h; $m\,(t_h)$ is the cumulative number of detected errors from o to t_h estimated by using the mean function; n is the number of observations, and k is the number of parameters.

(2) Mean absolute error (MAE), which is similar to MSE, is the absolute difference between the estimated and true values, i.e., $MAE = \frac{\sum_{h=1}^{n}|m_h - m(t_h)|}{n-k}$.

(3) Accuracy of estimation (AE) indicates the difference between the estimated errors and actual total number of detected errors, i.e. $AE = \left|\frac{M_a - a}{M_a}\right|$, where M_a is the actual cumulative number of detected errors during the testing process, and a is the estimated total number of errors.

(4) R-square explains the variability of data in a model, where the greater the value is, the better fit the model has, which is given by $Rsq = 1 - \frac{\sum_{h=1}^{n}(m_h - m(t_h))}{\sum_{h=1}^{n}\left(m_h - \frac{1}{n}\sum_{h=1}^{n} m_h\right)}$.

(5) Akaike information criterion (AIC) (Akaike, 1973) is defined as the log-likelihood term penalized by the number of model parameters. In regression models, the criterion can be defined as $AIC = n * \ln\left[\frac{RSS}{n}\right] + 2k$, where k denotes the number parameters used, n denotes the number of observations, and RSS denotes residual sums-of-squares. The above equation is for calculating of the AIC in the ordinary least squares (OLS) framework. The advantage of AIC is to consider the influence of different numbers of parameters in a model. However, it should be noted that AIC might be fit for MLE not for LSE since AIC's measures

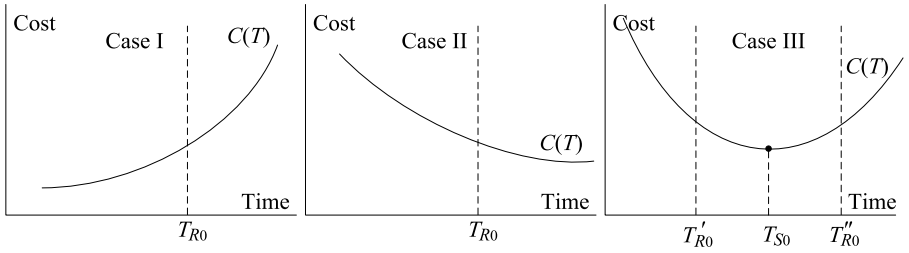

Fig. 2. The optimal decision for software release time in the three cases.

Table 1
Comparison of the mean value functions and error detection rates.

Model	Mean value function, $m(t)$, and Error detection rate, $d(t)$	Summary
Goel and Okumoto (1979)	$m(t) = a\left(1 - e^{-bt}\right); d(t) = b$	Fault occurrences phenomenon with a constant fault-detection rate during testing time
Yamada et al. (1983)	$m(t) = a\left(1 - (1+bt)e^{-bt}\right); d(t) = \frac{b^2 t}{1+bt}$	It is extended from Goel–Okumoto model. It can present a characteristic S-shaped curve.
Pham and Zhang (2003)	$m(t) = \frac{1}{1+\beta e^{-bt}}\left[(a+c)\left(1-e^{-bt}\right) - \frac{ab}{b-\alpha}\left(e^{-\alpha t} - e^{-bt}\right)\right]$ $d(t) = \frac{b}{\left(1-\beta e^{-bt}\right)}$	Take a non-decreasing test-coverage function and imperfect debugging into consideration.
Huang (2005)	$m(t) = a\left(1 - e^{-rW*(t)}\right); W(t) = \frac{N}{\sqrt[\kappa]{1+Ae^{-\alpha\kappa t}}}; W^*(t) = W(t) - W(0)$ $d(t) = \frac{AN\alpha r\left(1+Ae^{\alpha\kappa t}\right)^{\frac{-1}{\kappa}}}{A+e^{\alpha\kappa t}}$	Take testing-effort function into consideration. The testing effort is decided by decision makers.
Chiu et al. (2008)	$m(t) = a\left(1 - \frac{1+\frac{\eta}{\alpha}}{\frac{\eta}{\alpha}+e^{(\alpha+\eta)t}}\right); d(t) = (\alpha+\eta)\left(1 - \frac{\eta}{\alpha e^{(\alpha+\eta)t}+\eta}\right)$	Considering learning effect and autonomous errors-detected factor exists in the debugging process.
Proposed model	$m(t) = a\left(1 + e^{\alpha\left(e^{-\beta t}-1\right)}\left(\alpha\beta e^{-\beta t}\left(t + \frac{e^{-\eta t}-1}{\eta} + \frac{\left(2\delta\pi\left(Cos\left[\frac{\delta\pi t}{2}\right] - e^{\eta t}\right) + 4\eta Sin\left[\frac{\delta\pi t}{2}\right]\right)e^{-\eta t}}{\delta^2\pi^2+4\eta^2}\right) - 1\right)\right)$ $d(t) = \frac{\alpha\beta e^{-(2\beta+\eta)t}}{\eta\Upsilon_1}\left\{\begin{array}{c}\alpha\beta\left(\frac{2\Upsilon_2}{\beta} + (1-\eta t)\Upsilon_1\right)e^{\eta t} + (\Upsilon_1 + 2\Upsilon_2)(2\eta + \beta(1-\eta t))e^{(\beta+\eta)t} - \Upsilon_1(\alpha\beta + (\beta+\eta)e^{\beta t}) \\ -2\Upsilon_2(\alpha + e^{\beta t})Cos[\delta\pi t/2] - \eta\left(4\alpha\beta\eta + (\Upsilon_1 + 4\beta\eta)e^{\beta t}\right)Sin[\delta\pi t/2]\end{array}\right\}$	The imperfect debugging model incorporates with learning effect and cyclical error fluctuation.

require the maximum likelihood. Therefore, the multiple criteria should be considered simultaneously if LSE was used for estimating the parameters in a model.

Data sets from previously published papers are used in this study to compare the effectiveness of the proposed model with other existing models. Table 1 shows the results.

First, it is determined whether or not the proposed model has a good fit for different data sets, and the LSE is used to estimate the parameters based on the three data sets ([1]: (Zhang and Pham, 1998); [2]: (Shyur, 2003); [3]: (Pham and Zhang, 2003)). For simplicity without losing generality, we let the parameter $\delta = 1$ prevent cases in which too many parameters may result in estimation complexity. Fig. 3 shows the resulting fitness of the cumulative number of errors. Table 2 shows a comparison of the proposed model with other models for various criteria.

As can be seen in Table 2, the proposed model has a better fit in MSE, MAE, and R-squared. Note that the more parameters used can increase the model flexibility, and therefore, the fitting ability can be promoted but it still depends on an appropriate model design. In this study, the five parameters are used in the model to adapt for different testing scenarios but the fitting ability is also enhanced to the testing data. Although our performance in AIC criterion is not always the best due to the penalty of increasing parameters, the proposed model is able to deal with changes in testing resources, fluctuation in detection rate, learning effect, imperfect debugging and so on. Accordingly, the main advantage of the proposed model is that it can adapt to different testing conditions and still maintain good fitting results.

Besides, the aforementioned data sets do not have a sharp change point. Thus, three data sets with an obvious change point are artificially created to validate that the proposed model is applicable to deal with conditions with change points. Fig. 4 shows the results for the three data sets with change points, and Table 3 shows the comparison results.

As can be seen in Fig. 4, the cumulative number of errors increases dramatically between the 6th and 7th weeks in the data set [1], and the curve grows with a different trend after the 7th week, indicating that a change point occurs at the 6th week. The data set [2] shows that the curve of the total cumulative number of detected errors seems to be different after 6.524 weeks of testing, which indicates the change point. A change point occurs

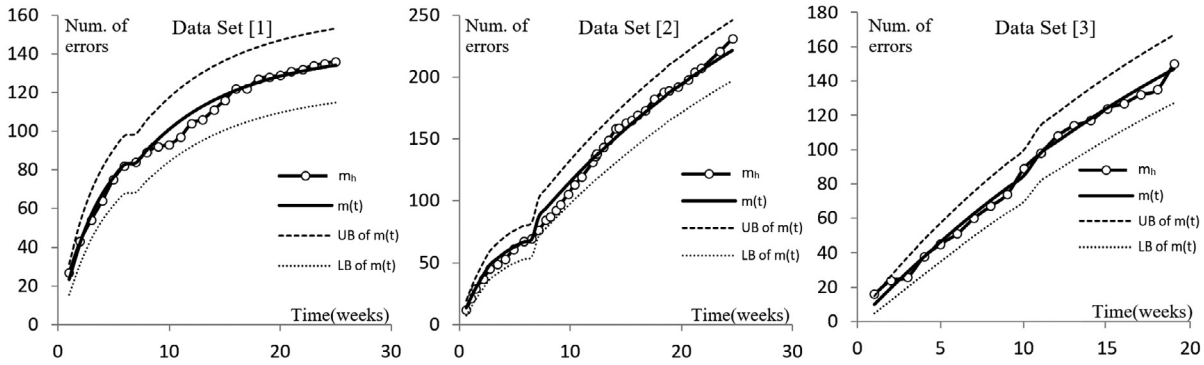

Fig. 3. Results of fit in the data sets (90% confidence interval).

Table 2
Comparison results (I).

Comparison criteria	Data sets	Goel and Okumoto (1979)	Yamada et al. (1983)	Pham and Zhang (2003)	Huang (2005)	Chiu et al. (2008)	The proposed model
MSE	[1]	33.309	134.230	38.668	32.051	34.828	**15.448**
	[2]	38.949	122.563	140.813	60.583	39.051	**31.773**
	[3]	18.159	46.993	46.814	31.508	16.998	**15.068**
MAE	[1]	4.491	8.645	5.186	4.935	4.689	**3.361**
	[2]	5.634	7.557	11.646	7.606	5.679	**4.744**
	[3]	4.047	5.265	6.538	5.353	3.821	**3.760**
AE	[1]	0.311	0.083	0.219	**0.071**	0.347	0.329
	[2]	1.039	**0.033**	0.973	2.005	0.499	0.628
	[3]	1.093	**0.066**	4.964	4.429	0.406	1.483
Rsq	[1]	0.966	0.865	0.966	0.973	0.966	**0.986**
	[2]	0.990	0.969	0.967	0.986	0.990	**0.993**
	[3]	0.991	0.976	0.979	0.987	0.992	**0.993**
AIC	[1]	42.064	57.196	47.684	45.646	44.548	**39.722**
	[2]	**64.439**	83.358	89.648	75.729	66.482	67.078
	[3]	**27.923**	35.769	39.737	36.470	29.378	32.383

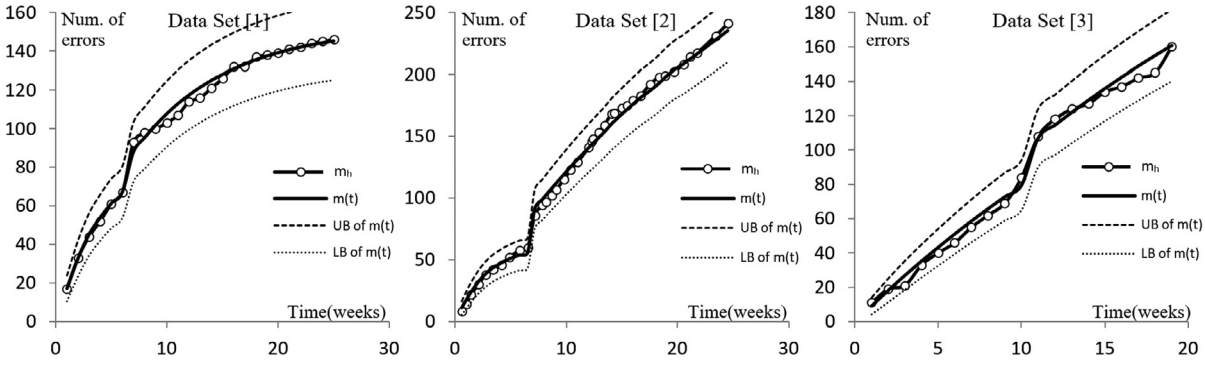

Fig. 4. Results of fit in the data sets with change points (90% Confidence Interval).

at the 10th week of the data set [3]. This study uses the three data sets to compare the effectiveness of the proposed model with that of other models based on the four comparison criteria. As can be seen in Table 3, the proposed model presents a better fit in MSE and MAE.

5. Application and numerical analysis

Suppose that a software company develops a commercial system, and the development of the software program is in the last phase. The manager has to decide the optimal release time of the commercial system. According to the testing data of a previous and similar project, the parameters of the mean value function are estimated to be $\hat{a} = 1000$, $\hat{\alpha}_b = 2.5$, $\hat{\alpha}_f = 3.5$, $\hat{\beta} = 0.1$ and $\hat{\delta} = 0.8$. The testing staff of the team works about 200 h in a month. The team includes two groups, one is in charge of detecting errors and the other is responsible for correction. The expected time of correcting a simple or complex error ($E[\hat{\theta}_s, \xi_s, t_r]$, $E[\hat{\theta}_c, \xi_c, t_r]$) is estimated to be 1.5 or 2.5 h respectively from the historical datasets $\mathbf{D}^{(s)}$ and $\mathbf{D}^{(c)}$. Besides, the relevant costs are: C_0 is \$15,000; C_1 is \$65,000 per month; C_{2s} and C_{2c} are \$550 and \$700, respectively; C_{3s} and C_{3c} are \$800 and \$1,000, respectively. In addition to the fixed setup cost, the daily administrative cost, debugging cost (no matter how long the testing or warranty period is), the risk cost and the opportunity loss are also considered. C_4 is \$3,000, 000 (every one percent reliability for \$30,000 reputation loss; the change point τ will occur at the beginning of the sixth months due to the increase

Table 3
Comparison results (II).

Comparison criteria	Data set	Goel and Okumoto (1979)	Yamada et al. (1984)	Pham and Zhang (2003)	Huang (2005)	Chiu et al. (2008)	Proposed model
MSE	[1]	12.124	54.676	70.305	95.913	12.623	**8.188**
	[2]	46.923	56.675	156.319	167.851	25.837	**18.286**
	[3]	82.308	46.311	64.049	90.537	33.415	**25.223**
MAE	[1]	2.382	6.404	2.660	9.768	2.506	**2.317**
	[2]	5.343	5.276	11.912	15.068	4.124	**3.982**
	[3]	8.361	6.330	7.483	8.219	5.421	**4.952**
AE	[1]	0.073	0.041	0.204	1.965	**0.068**	**0.051**
	[2]	1.107	**0.043**	1.051	1.408	0.107	0.644
	[3]	1.186	0.218	4.801	3.565	**0.086**	1.541
Rsq	[1]	0.992	0.965	0.961	0.949	0.992	**0.995**
	[2]	0.990	0.988	0.970	0.948	0.995	**0.996**
	[3]	0.968	0.982	0.978	0.971	0.988	**0.992**
AIC	[1]	**31.091**	47.445	54.175	57.547	33.529	32.829
	[2]	**67.513**	70.629	91.372	92.547	59.665	57.961
	[3]	40.393	35.648	42.324	45.180	**34.955**	36.634

Table 4
Parameter settings.

Parameter	Value
C_0	\$15,000
C_1	\$65,000 (per month)
C_{2s}, C_{2c}	\$550 (per h/team) and \$700 (per h/team)
C_{3s}, C_{3c}	\$800 (per h/team) and \$1,000 (per h/team)
C_4	\$3,000,000 (\$30,000 reputation loss v.s. 1% reliability)
q	(65% simple errors, 35% complex errors)
τ	The beginning of the 6th months
T_w	1 month
R_0	0.95
v_0, v_1, v_2	12,000, 1, and 1.6
r	2.5% (per year)
$\hat{a}$, $\hat{\alpha}_b$, $\hat{\alpha}_f$, $\hat{\beta}$, $\hat{\eta}$, $\hat{\delta}$, t_{op}	$\hat{a} = 1000$, $\hat{\alpha}_b = 2.5$, $\hat{\alpha}_f = 3.5$, $\hat{\beta} = 0.1$, $\hat{\eta} = 0.5$, $\hat{\delta} = 0.8$, $t_{op} = 1$ h
$\mathbf{D}^{(s)}$, $\mathbf{D}^{(c)}$, ξ_s, $\xi_{\hat{c}}$, $\hat{\theta}_s$, $\hat{\theta}_c$, $E[\hat{\theta}_s, \xi_s, t_r]$, $E[\hat{\theta}_c, \xi_c, t_r]$	$\mathbf{D}^{(s)} = \{1.5, 0.2, 2.9, 0.9, 1.2, 3.1, 0.5, 1.1, 2.3, 2.5, 1.4, 0.4\}$, $\xi_s = 5.5$, $\hat{\theta}_s = 1.75$, $E[\hat{\theta}_s, \xi_s, t_r] \cong 1.5$ (h) $\mathbf{D}^{(c)} = \{3.1, 2.2, 1.3, 2.7, 2.4, 1.3, 3.3, 4.9, 1.1, 4.2, 2.3, 1.4\}$, $\xi_c = 11$, $\hat{\theta}_c = 2.70$, $E[\hat{\theta}_c, \xi_c, t_r] \cong 2.5$ (h)

in testing staff. The warranty period T_w is set up to a month for the requirement of the contract; the final reliability R_0 must reach 0.95 to meet the customer's requirement; v_0, v_1, and v_2 are respectively 15,000, 1, and 2.2, and the interest rate, r, is 2.5% per year. Table 4 summarizes all the settings of the parameters.

In Fig. 5, the cost function $C(T)$ shows the convexity with the testing time, T, and finds the minimum at 8.6 months. The estimation of the expected total cost is \$1,739,338 with the reliability 0.9495. Although it fails to satisfy the requirements of the contractual level 0.95 at present, the warranty period can be used to continually improve the quality of the system. Accordingly, the commercial system will be released at 8.6 months, and the testing and debugging will remain for a month. When the warranty period ends, the reliability of the system can reach 0.9716 to satisfy the customer's requirements. Note that the total cost does not increase with the extending testing time. Although the rise of testing time will increase the cost of workforce, it also decreases the risk cost due to low software reliability and the opportunity loss of software release delayed. Figs. 5 and 6 show the testing cost and reliabilities for different release times. Table 5 shows the comparative results of different time points.

Some crucial parameters are also considered by performing a sensitivity analysis to investigate their impacts on the decision variables and minimum total cost. Figs. 7 and 8 show the impacts of the related costs. As can be seen in Fig. 7, the optimal release time for the software program is rather sensitive to the parameter, v_2, of opportunity loss which means that misevaluating

Table 5
The comparative results of different time points.

	T_{R_0}	T^*	$T^* + T_w$
Time	8.65	8.60	9.60
Reliability	0.95	0.9495	0.9716
Total cost	\$1,738,225	\$1,739,338	\$1,727,728

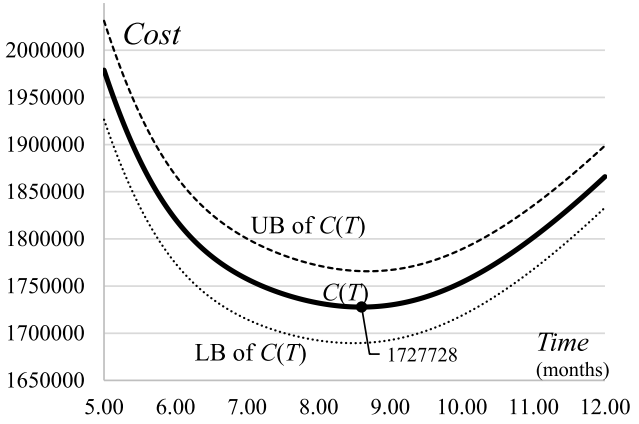

Fig. 5. The expected cost of the release time (90% confidence interval).

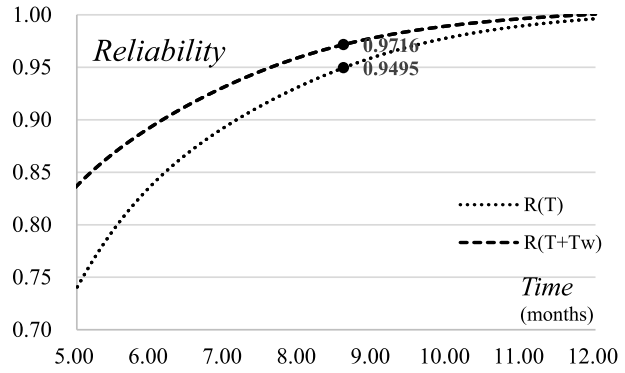

Fig. 6. The expected reliability of the release time.

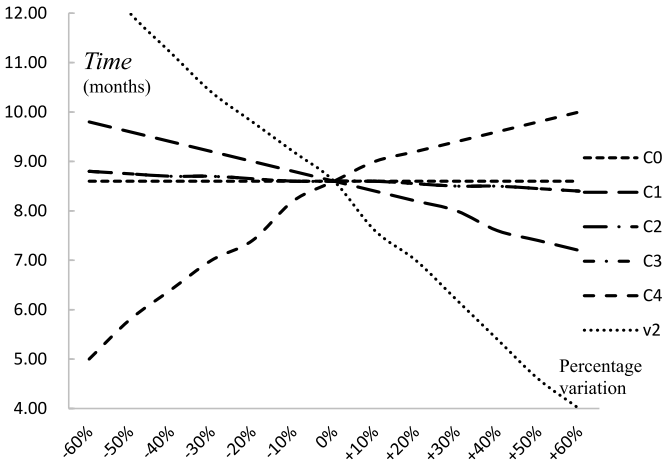

Fig. 7. The impacts of related costs on the time at which the minimum cost is reached.

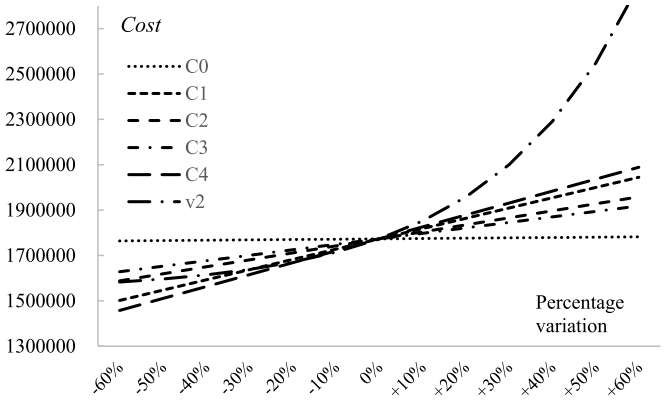

Fig. 8. The impact of related costs on the minimum total cost.

the parameter will lead to severe consequences. That is, if the manager makes an inadequate decision on the release time for the software program, it will cause a large unnecessary loss. The second significant impact is the risk cost C_4, and this can be explained by the fact that if software errors occur after the software system has been released to the market, the software development company will certainly have to elevate debugging costs much higher for recovering the loss of reputation. However, the increase of this cost would surely delay the release time, and thus the manager has to extend the testing period. On the contrary, the manager will shorten the testing period owing to the increase of the daily administrative costs and the opportunity loss, since the greater values of C_1 and v_2 indicate the release of the software system to the market earlier. Note that C_2 and C_3 may not significantly influence the release time of the software program since either the increase or decrease in the fixed costs does not affect the shape of the cost function. The expenditure of error removal depends merely on the number of errors and has no impacts on the testing process. Furthermore, in Fig. 8, the opportunity loss still has the most impact on the total cost, and C_0, C_1, C_2, and C_3 seem insignificant to that. This can be explained by the fact that the saving of the daily administrative cost, fixed costs, and debugging costs cannot cover the opportunity loss. In addition, it is obvious that failures occur after the software release also result in a greater loss. Although the error removal cost is high after the release of the software program, protecting the reputation of the software development company for providing a reliable system to customers is more than the expenditure of the extending testing period.

The changes in the error detection rate will significantly influence the number of errors which can found and corrected, and thus the reliability of the software program and the total cost spending for different release times. As can be seen in Fig. 9, promoting the learning factor η facilitates to the rise of reliability due to the increase of errors removal, and thus the cost will decrease with the testing time. However, to promote the learning factor needs more in-job training to the testing staffs in advance which may boost the cost of staff education. It should be noted that the marginal effect on the total cost decreases with the value of the learning factor. Accordingly, the software developer has to consider the trade-off between the investment of staffs' skills and the benefits of the cost reduction from enhanced reliability. Moreover, the most straightforward way for debugging efficiency is to increase the testing resources, such as, expand workforce or working hours. According to the sensitive analysis of α in Fig. 9, the reliability can be improved with the increase of the testing resource, α, but the total cost will not definitely decrease. If the testing resource is insufficient, the debugging efficiency may be insufficient to cover the increment of testing resources and efforts. In this case, the total cost is still going up if α is smaller than 2.0 which means that the software developer must launch ample testing resources and efforts to achieve the benefits of the reduction of the total cost. With regard to the scale and shape factor, β, it is sensitive to both reliability and the total cost, and misestimating the value of the factors will cause a serious consequence of inappropriate software release decisions. Therefore, the software developer should evaluate it carefully with a reasonable confidence interval to avoid over-optimistic expectations.

To investigate the impacts that extending warranty length has on the increments of total cost and reliability, the sensitivity analyses are performed and presented in Fig. 10. Generally, extending the warranty length can increase and elevate the customer satisfaction through the rise of reliability, but it also bears the burden of cost even though the marginal cost decreases. However, the effect still decreases with the extending warranty length. Therefore, the software developer can only provide a contact with an applicable warranty length to the customer after negotiation, and it implies that overly extending the warranty length is unrealistic.

6. Conclusion

The construction of a software reliability growth model assists managers in forecasting software reliability, making it possible for them to evaluate and monitor the development process of a software program and then make decisions regarding resource allocation and the optimal time at which the program is released.

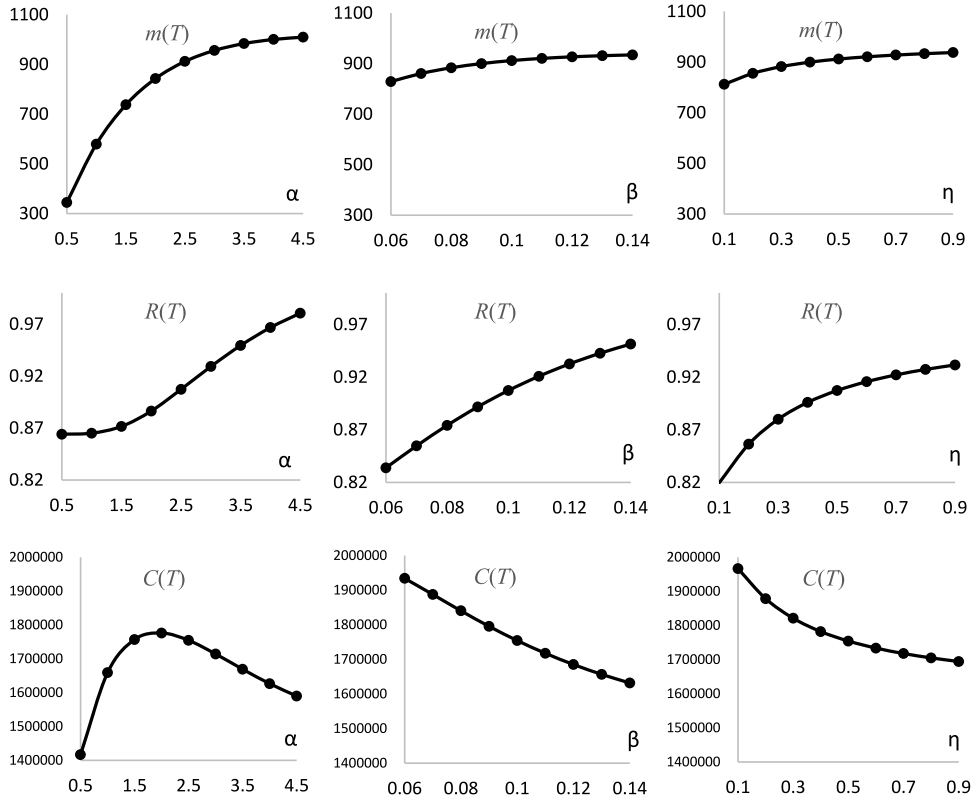

Fig. 9. The impact of related parameters α, β, η on mean value function, reliability, and cost.

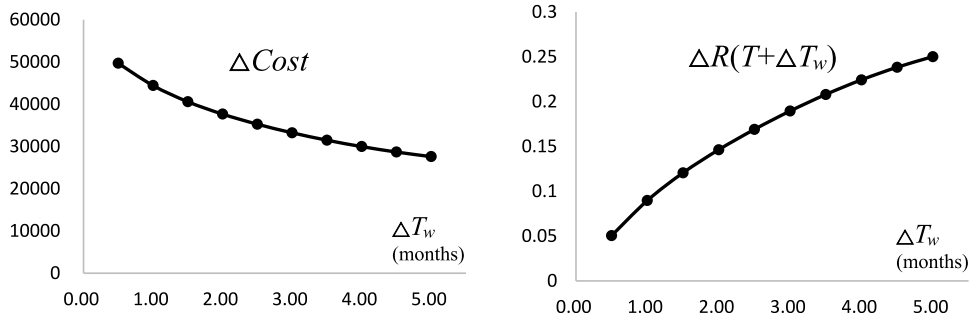

Fig. 10. The Impact of the marginal warranty length on the increment of cost and reliability.

Thus, if a model is constructed by considering sufficient factors and practical failure data, the software reliability forecast will be more applicable for decision makers in practice. The proposed model is based on an NHPP model and is used to describe the software failures process in which the mean function is the cumulative number of errors, which changes over time, to describe the instantaneously cumulative number of detected errors during the testing period. Since the perfect debugging assumed by previous studies seems unrealistic, a software reliability growth model with imperfect debugging is constructed in this study in which the skills and abilities of the testing staff will affect the change in the number of errors in the software program. Error detection and removal are often performed by software engineers, and repeated testing activities will thus result in learning effects. Moreover, errors are not the same. This study investigates the impacts of both learning effects and error classification on the change in the number of errors during the debugging process. In addition, this study also considers the occurrence of a change point. Changes in the resources available during the testing period will affect the debugging efficiency of the testing staff and result in different reliability growth curves before and after the change point.

The proposed model can be used to determine the optimal time to release a software program to the market at which the contractual level of reliability is maintained with the minimum testing cost. With regard to costs, instead of merely considering the setup cost for testing and debugging during the testing and warranty period, this study also considers reputation losses incurred due to errors found after the software program has been released and the opportunity losses related to delaying the release of the program.

Multiple change points can be considered for future research because many environmental factors may impact the debugging process at different times and thus result in multiple change points. Error classification can also be extended by considering the impacts that errors have on the system. The testing staff could first correct errors that have more serious impacts on the system. The exponential functional form of error detection rate can also be changed to a more general one, such as a Weibull function.

Besides, Hausdorff approximation may give useful measurements to avoid the situation of overfitting for estimating parameters. Therefore, the issue could be incorporated in this study for future directions. It is hoped that such integrated studies can provide the advantageous strategies for software developers.

CRediT authorship contribution statement

Yeu-Shiang Huang: Conceptualization, Methodology, Writing – revised and editing, Supervision. **Kuei-Chen Chiu:** Methodology, Reviewing and editing, Validation. **Wan-Ming Chen:** Data curation, Visualization, Investigation, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Ahmad, N., Khan, M.G.M., Rafi, L.S., 2010. A study of testing-effort dependent inflection S-shaped software reliability growth models with imperfect debugging. Int. J. Qual. Reliab. Manag. 27 (1), 89–110.

Akaike, H., 1973. Information theory and an extension of the maximum likelihood principle. In: Second International Symposium on Information Theory. pp. 267–281.

Aktekin, T., Caglar, T., 2013. Imperfect debugging in software reliability: A Bayesian approach. European J. Oper. Res. 227, 112–121.

Boh, W.F., Slaughter, S.A., Espinosa, J.A., 2007. Learning from experience in software development: A multilevel analysis. Manage. Sci. 53 (8), 1315–1331.

Chatterjee, S., Misra, R.B., Alam, S.S., 1997. Joint effect of test effort and learning factor on software reliability and optimal release policy. Internat. J. Systems Sci. 28 (4), 391–396.

Chatterjee, S., Nigam, S., Singh, J.B., Upadhyaya, L.N., 2012. Effect of change point and imperfect debugging in software reliability and its optimal release policy. Math. Comput. Model. Dyn. Syst. 18 (5), 1–13.

Chiu, K.C., Huang, Y.S., Huang, I.C., 2019. A study of software reliability growth with imperfect debugging for time-dependent potential errors. Int. J. Ind. Eng. 26 (3), 376–393.

Chiu, K.C., Huang, Y.S., Lee, T.Z., 2008. A study of software reliability growth from the perspective of learning effects. Reliab. Eng. Syst. Saf. 93 (10), 1410–1421.

Dadkhah, M., Araban, S., Paydar, S., 2020. A systematic literature review on semantic web enabled software testing. J. Syst. Softw. 162, http://dx.doi.org/10.1016/j.jss.2019.110485.

Duffey, R.B., Fiondella, L., 2014. Software, hardware, and procedure reliability by testing and verification: Evidence of learning trends. IEEE Trans. Human–Mach. Syst. 44 (3), 395–405.

Fang, C.C., Yeh, C.W., 2016. Effective confidence interval estimation of fault-detection process of software reliability growth models. Internat. J. Systems Sci. 47 (12), 2878–2892.

Garmabaki, A.H.S., Aggarwal, A.G., Kapur, P.K., 2011. Multi up-gradation software reliability growth model with faults of different severity. In: Proceedings of the 2011 IEEE IEEM. pp. 1539–1543.

Goel, A.L., Okumoto, K., 1979. Time dependent fault detection rate model for software and other performance measures. IEEE Trans. Reliab. 28 (2), 206–211.

Ho, J.W., Fang, C.C., Huang, Y.S., 2008. The determination of optimal software release times at different confidence levels with consideration of learning effects. Softw. Test. Verif. Reliab. 18 (4), 221–249.

Hou, R.H., Kuo, S.Y., Chang, Y.P., 1994. Applying various learning curves to hyper-geometric distribution software reliability growth model. In: Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering. pp. 8–17.

Huang, C.Y., 2005. Performance analysis of software reliability growth models with testing-effort and change-point. J. Syst. Softw. 76 (2), 181–194.

Huang, C.Y., Lin, C.T., 2010. Analysis of software reliability modeling considering testing compression factor and failure-to-fault relationship. IEEE Trans. Comput. 59 (2), 283–288.

Huang, C.Y., Lyu, M.R., 2011. Estimation and analysis of some generalized multiple change-point software reliability models. IEEE Trans. Reliab. 60 (2), 498–514.

Inoue, S., Ikeda, J., Yamada, S., 2016. Bivariate change-point modeling for software reliability assessment with uncertainty of testing-environment factor. Ann. Oper. Res. 244 (1), 209–220.

Inoue, S., Taniguchi, S., Yamada, S., 2015. An all-stage truncated multiple change point model for software reliability assessment. Int. J. Reliab. Qual. Saf. Eng. 22 (4), 1550017–1–1550017–17.

Inoue, S., Yamada, S., 2018. Markovian software reliability modeling with change-point. Int. J. Reliab. Qual. Saf. Eng. 25 (2), 1–13.

Jain, M., Agrawal, S.C., Agarwal, P., 2012. Markovian software reliability model for two types of failures with imperfect debugging rate and generation of errors. Int. J. Eng. Trans. A: Basics 25 (2), 177–188.

Jain, M., Manjula, T., Gulati, T.R., 2014. Imperfect debugging study of SRGM with fault reduction factor and multiple change point. Int. J. Math. Oper. Res. 6 (2), 155–175.

Kapur, P.K., Basirzadeh, M., Inoue, S., Yamada, S., 2010. Stochastic differential equation based SRGM for errors of different severity with testing-effort. Int. J. Reliab. Qual. Saf. Eng. 17 (3), 179–197.

Kapur, P.K., Garg, R.B., Aggarwal, A.G., Tandon, A., 2009. General framework for change point problem in software reliability and related release time problem. Int. J. Reliab. Qual. Saf. Eng. 16 (6), 567–579.

Kapur, P.K., Goswami, D.N., Gupta, A., 2004. A software reliability growth model with testing effort dependent learning function for distributed systems. Int. J. Reliab. Qual. Saf. Eng. 11 (04), 365–377.

Kapur, P.K., Kumar, A., Yadav, K., Khatri, S.K., 2007. Software reliability growth modeling for errors of different severity using change point. Int. J. Reliab. Qual. Saf. Eng. 14 (4), 311–326.

Kapur, P.K., Pham, H., Chanda, U., Kumar, V., 2013. Optimal allocation of testing effort during testing and debugging phases: a control theoretic approach. Internat. J. Systems Sci. 44 (9), 1639–1650.

Kaushal, R., Khullar, S., 2012. PSO Based neural network approaches for prediction of level of severity of faults in NASA's public domain defect dataset. Int. J. Inf. Technol. Knowl. Manag. 5 (2), 453–457.

Khatri, S., Chhillar, R.S., 2012. Designing debugging models for object oriented systems. Int. J. Comput. Sci. Issues 9 (1), 350–357.

Lai, R., Gard, M., 2012. A Detailed Study of NHPP Software Reliability Models. J. Soft. 7 (6), 1296–1306.

Lemos, O.A.L., Silveira, F.F., Ferrari, F.C., Garcia, A., 2018. The impact of software testing education on code reliability: An empirical assessment. J. Syst. Softw. 137, 497–511.

Li, Q., Pham, H., 2017. NHPP Software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage. Appl. Math. Model. 51, 68–85.

Li, Q., Pham, H., 2019. A generalized software reliability growth model with consideration of the uncertainty of operating environments. IEEE Access 7, 84253–84267.

Nagaraju, V., Jayasinghe, C., Fiondella, L., 2020. Optimal test activity allocation for covariate software reliability and security models. J. Syst. Softw. 168, http://dx.doi.org/10.1016/j.jss.2020.110643.

Okamura, H., Watanabe, Y., Dohi, T., 2003. An iterative scheme for maximum likelihood estimation in software reliability modeling. In: 14th International Symposium on Software Reliability Engineering. pp. 246–256.

Pavlov, N., Iliev, A., Rahnev, A., Kyurkchiev, N., 2018a. Ome Software Reliability Models: Approximation and Modeling Aspects. LAP LAMBERT Academic Publishing, ISBN: 978-613-9-82805-0.

Pavlov, N., Iliev, A., Rahnev, A., Kyurkchiev, N., 2018b. Analysis of the Chen's and Pham's software reliability models. Cybern. Inf. Technol. 18 (3), 37–47.

Pham, H., 2007. An imperfect-debugging fault-detection dependent-parameter software. Int. J. Autom. Comput. 4 (4), 325–328.

Pham, H., Zhang, X., 2003. NHPP Software reliability and cost models with testing coverage. European J. Oper. Res. 145, 443–454.

Rafi, S.M., Rao, D.K.N., 2010. SRGM With logistic-exponential testing-effort function with change-point and analysis of optimal release policies based on increasing the test efficiency. Int. J. Comput. Sci. Eng. 2 (3), 504–516.

Raju, O.N., 2011. Software reliability growth models for the safety critical software with imperfect debugging. Int. J. Comput. Sci. Eng. 3 (8), 3019–3026.

Saraf, I., Iqbal, J., 2019. Generalized multi-release modelling of software reliability growth models from the perspective of two types of imperfect debugging and change point. Qual. Reliab. Eng. Int. http://dx.doi.org/10.1002/qre.2516.

Shibata, K., Rinsaka, K., Dohi, T., 2006. Metrics-based software reliability models using non-homogeneous Poisson processes. In: 17th International Symposium on Software Reliability Engineering. IEEE, pp. 52–61.

Shyur, H.J., 2003. A stochastic software reliability model with imperfect-debugging and change-point. J. Syst. Softw. 66 (2), 135–141.

Wang, H., Fei, H., Yu, Q., Zhao, W., Yan, J., Hong, T., 2019. A motifs-based maximum entropy Markov model for realtime reliability prediction in system of systems. J. Syst. Softw. 151, 180–193.

Wang, J., Wu, Z., 2016. Study of the nonlinear imperfect software debugging model. Reliab. Eng. Syst. Saf. 153, 180–192.

Wang, J., Wu, Z., Shu, Y., Zhang, Z., 2015. An imperfect software debugging model considering log–logistic distribution fault content function. J. Syst. Softw. 100, 167–181.

Wang, J., Wu, Z., Shu, Y., Zhang, Z., 2016. An optimized method for software reliability model based on nonhomogeneous Poisson process. Appl. Math. Model. 40, 6324–6339.

Xia, G., Zeephongsekul, P., Kumar, S., 1992. Optimal software release policies for models incorporating learning in testing. Asia-Pac. J. Oper. Res. 9 (2), 221–234.
Yadavalli, V.S.S., Aggarwal, A.G., Kapur, P.K., Kumar, J., 2010. Unified framework for developing testing effort dependent software reliability growth models with change point and imperfect debugging. In: Proceedings of the 4th National Conference. INDIA Computing For Nation Development.
Yamada, S., Ohba, M., Osaki, S., 1983. S-Shaped Reliability Growth Modeling for Software Error Detection. IEEE Trans. Reliab. R-32 (5), 475–484.
Yamada, S., Ohba, M., Osaki, S., 1984. S-Shaped Software Reliability Growth Models and Their Applications. IEEE Trans. Reliab. R-33 (4), 289–292.
Yamada, S., Ohtera, H., Narihisa, H., 1986. Software reliability growth models with testing effort. IEEE Trans. Reliab. 4, 19–23.
Yamada, S., Osaki, S., 1985. Software reliability growth modeling: Models and applications. IEEE Trans. Softw. Eng. 11 (12), 1431–1437.
Zeephongsekul, P., Chiera, C., 1995. Optimal software release policy based on a two-person game of timing. J. Appl. Probab. 32 (2), 470–481.
Zhang, X., Pham, H., 1998. A software cost model with warranty cost, error removal times and risk costs. IIE Trans. 30, 1135–1142.
Zhao, X., Littlewood, B., Povyakalo, A., Strigini, L., Wright, D., 2018. Conservative claims for the probability of perfection of a software-based system using operational experience of previous similar systems. Reliab. Eng. Syst. Saf. 175, 265–282.
Zhu, M., Pham, H., 2017. Environmental factors analysis and comparison affecting software reliability in development of multi-release software. J. Syst. Softw. 132, 72–84.

Yeu-Shiang Huang is currently a professor in the Department of Industrial and Information Management at National Cheng Kung University, Taiwan. He earned both his M.S. and Ph.D. degrees in Industrial Engineering from the University of Wisconsin–Madison, U.S.A. His research interests include operations management, supply chain management, reliability engineering, and decision analysis. Related papers have appeared in such professional journals as *IIE Transactions, Naval Research Logistics, Decision Support Systems, IEEE Transactions on Engineering Management, European Journal of Operational Research, Reliability Engineering and System Safety, Software Testing, Verification and Reliability, IEEE Transactions on Reliability, International Journal of Production Research, Computers and Operations Research, Computers and Industrial Engineering, Communications in Statistics*, and others.

Kuei-Chen Chiu is currently an assistant professor in the Department of Finance at Shih Chien University (Kaohsiung Campus), Taiwan. She earned both her M.S. and Ph.D. degrees from Industrial and Information Management of National Cheng Kung University. Her research interests include software reliability, operations management, human factor, human resource management, and performance assessment. Related papers have appeared in such professional journals as *Reliability Engineering and System Safety, Software Quality Journal, Journal of Taiwan Issue Economics* and others. She has also earned the second M.S. degree from the Department of Counseling, National Chiayi University and has the second Ph.D. degree program proceeding in the Institute of Allied Health Sciences, College of Medicine, National Cheng Kung University. She currently devotes to interdisciplinary researches of industrial engineering, industrial management, psychology, and cognitive science.

Wan-Ming Chen is a graduate student in the Department of Industrial and Information Management at National Cheng Kung University, Taiwan.