Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Software testing and release decision at different statistical confidence levels with consideration of debuggers' learning and negligent factors

Chun-Wu Yeh[a] and Chih-Chiang Fang[b*]

[a]*Department of Information Management, Computer and Game Development Program, Kun Shan University, No. 195, Kunda Rd., Yongkang Dist., Tainan City 71070, Taiwan, R.O.C.*
[b]*School of Information and Technology, Wenzhou Business College, Wenzhou, Zhejiang 325006, China*

C H R O N I C L E

Article history:
Received September 3 2023
Received in Revised Format October 6 2023
Accepted November 8 2023
Available online November 8 2023

Keywords:
Statistical confidence levels
Imperfect debugging
Software reliability
Learning factor
Brown motion

A B S T R A C T

This research delves into the software testing process and its environmental factors to uncover the core elements influencing software reliability. Specifically, it focuses on the learning and negligent aspects of the software reliability growth model. The learning factor accelerates reliability growth, leading to an S-shaped curve in the mean value function, while the negligent factor highlights the occurrence of imperfect debugging. The study also uses Brownian motion and stochastic differential equations to establish statistical confidence intervals for reliability and costs. These intervals aid software managers in assessing potential release risks at various confidence levels, allowing them to make informed decisions considering resource constraints and desired system reliability levels across different scenarios.

1. Introduction

The paramount concern of software developers and users lies in software reliability, a factor that augments user satisfaction and mitigates potential losses stemming from system failures. In recent decades, software reliability has become the predominant benchmark for evaluating software quality and is a fundamental pillar for devising effective software testing strategies. Within the software industry, a central challenge pertains to enhancing reliability while concurrently managing the fiscal outlays associated with software development. Numerous software reliability growth models (SRGMs) have been introduced, with the prevalent reliance on the Non-Homogeneous Poisson Process assumption for characterizing the dynamics of software testing and debugging phenomena. (Goel & Okumoto, 1979; Pham & Zhang, 2003; Huang, 2005; Chiu et al., 2008; Kapur et al., 2011; Li & Pham, 2019; Huang et al., 2022; Tian et al., 2022; Huang et al., 2023).

The prevailing pattern for software test data typically exhibits an S-shaped or exponential trajectory. However, prior models have often been constrained to apply effectively to only one of these data patterns, limiting their adaptability. An ideal model should be capable of accommodating the diverse test data patterns for superior fitting results. For instance, Huang et al. (2007) introduced a software reliability growth model incorporating testing effort, acknowledging the rapid early-stage growth followed by a gradual deceleration, resulting in an S-shaped curve for the growth rate of testing effort over time. Chiu et al. (2008) emphasized learning effects during testing and debugging, introducing two critical factors related to learning effects within their model. Their experiments, focused on fitness excellence, revealed an S-shaped pattern in the test data, substantiating the impact of learning effects during testing. Ramsamy et al. (2012) underscored that many existing software reliability models overlooked the learning factors of testers and proposed a model that integrates debugging and learning

* Corresponding author
E-mail: peterfang0214@yahoo.com.tw (C.-C. Fang)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2024 Growing Science Ltd.
doi: 10.5267/j.ijiec.2023.11.001

indices to enhance adaptability across various software testing projects. Similarly, Ahmad et al. (2010) established NHPP-based inflection S-shaped software reliability models, offering applicability to both exponential and S-shaped data types. Pachauri et al. (2015) advanced a multi-stage SRGM for software upgrades, incorporating an inflection S-shaped curve, fault reduction factors, and imperfect debugging processes. Jin and Jin (2016) optimized parameters of the Huang test-effort model via quantum particle swarm optimization, yielding superior fitting performance for diverse software testing data sets encompassing both exponential and S-shaped patterns. Chiu et al. (2019) and Huang et al. (2022) incorporated learning factors into their models. These models could evolve linearly or exponentially over time, enabling adaptability to S-shaped and exponential testing data, contingent on the learning effects.

Furthermore, it is imperative to acknowledge the inherent imperfection in the debugging process, challenging the idealized scenario of flawless debugging. Imperfections imply that errors may not be entirely eradicated once errors are detected, potentially giving rise to new errors. Recent research endeavors have increasingly considered this notion when developing Software Reliability Growth Models (SRGMs). For instance, Chatterjee et al. (2012) proposed an SRGM incorporating imperfect debugging and time-related error detection capabilities. Aktekin and Caglar (2013) introduced a model featuring a multiplicative failure rate that dynamically fluctuates during the testing phase, elucidating the imperfections in debugging practices. Wang et al. (2015) adopted a fault content function to accommodate the log-logistic distribution, presenting an imperfect debugging model to estimate software reliability. Li and Pham (2017, 2019) introduced an SRGM that considers the uncertainty of the operating environment, acknowledging that debugging and test coverage may not be flawless. Zhao et al. (2018) employed a Bayesian approach to address the uncertainty of software reliability, accounting for both perfection and imperfection factors. Saraf and Iqbal (2019) developed an imperfect debugging model catering to multiple software versions and change points. Shrivastava et al. (2021) introduced a software reliability-based model for handling software scheduling issues, considering change-point scenarios. Ke and Huang (2020) proposed an SRGM featuring multiple change points to depict the consumption of testing effort and enhance software reliability analysis effectiveness. Significantly, this study departs from the conventional imperfect debugging models by considering the error correction process and its efficiency, contingent on detected error patterns. It recognizes that pure testing should avoid introducing new errors as the system code remains unaltered during pure testing phases.

In addition, there have been debates regarding the applicability of the Non-Homogeneous Poisson Process (NHPP) to software reliability growth models. Cai et al. (2008) conducted experiments comparing the mean value function's accumulated errors to its variance and discovered that the variance does not amplify proportionally as testing time elapses. Wang et al. (2016) introduced a global optimization method for NHPP models coupled with a refined dataset to enhance the accuracy of software error prediction. Their approach sought to enhance the performance of traditional NHPP-based models. These conflicting perspectives on NHPP-based software reliability growth models have raised questions about whether any such models adhere to the NHPP.

Additionally, interval estimation holds significance in software reliability research. Yamada and Osaki (1985) initially devised a formula for calculating confidence intervals of the mean value function, expressed as $\widehat{m}(T) \pm Z_{CR/2}\sqrt{\widehat{m}(T)}$. CR denotes the critical region, and $Z_{CR/2}$ signifies the critical value for a given area $CR/2$ in the standard normal distribution. It is worth noting that this formula aligns with the traditional NHPP assumption, as it posits that the variance of the mean value function increases with testing time. Huang (2005) employed confidence intervals to construct a test-effort Software Reliability Growth Model (SRGM), estimating the potential range of errors detectable at various testing times.

Nevertheless, the conventional method for evaluating confidence intervals can decrease accuracy in estimating cost ranges as the recommended confidence intervals expand with time. This may seem counterintuitive in practice, given that the number of remaining software errors naturally decreases with testing time. Fang and Yeh (2016) proposed an alternative estimation method for the traditional model, facilitating the evaluation of confidence intervals for the mean value function. Although their model did not account for imperfect debugging, the suggested confidence intervals did not exhibit expansion as testing time progressed.

The primary application of Software Reliability Growth Models (SRGM) is to support decision-makers in identifying the optimal software release time, considering diverse costs, system reliability, and imposed constraints. Consequently, studies in this domain consistently integrate these factors into their software release models, shaped by their unique testing environments. Cortellessa et al. (2015) devised an optimization-based approach to minimize the cost of software evolution while factoring in software reliability and performance constraints. Awad (2016) proposed allocating testing time in central systems and subsystems to minimize system failures within given time and cost limitations. Kooli et al. (2017) discussed differences between software testing and software fault injection techniques in the context of reliability evaluation concerning time and cost. Li and Pham (2017) considered environmental uncertainties, introducing a software reliability model that encompasses testing coverage, wherein the optimal software release time hinges on software reliability and cost. Zhu and Pham (2018) contended that achieving error-free software releases for every iteration is nearly impossible due to resource constraints, underscoring the necessity of defining an acceptable threshold for multiple software releases. Cao et al. (2019) and Tian et al. (2022) put forth software reliability models to minimize cumulative testing and penalty costs after software release, elucidating the threshold structure that underpins optimal release policies. Levitin et al. (2020) proposed an optimal

rejuvenation policy for real-time software tasks, accommodating arbitrary state transition time distributions within their cost evaluation model.

Considering the preceding discussions, this study introduces a software reliability growth model that incorporates the learning and negligence factors associated with software testers. Moreover, the proposed model provides a rational means of estimating confidence intervals for software reliability and cost. This model is a valuable tool for software engineers to assess software release risks across different confidence levels. The remainder of this paper is structured as follows: Section 2 outlines the model development, addressing the learning and negligence factors. Section 3 offers the mathematical derivation of confidence intervals for the mean value function employing stochastic differential equations. Section 4 presents the validation of the proposed model and comparative analysis with other models. Section 5 introduces a practical decision model for optimal software release, while Section 6 offers an illustrative example and pertinent discussion. Finally, in Section 7, concluding remarks and recommendations for future research are provided.

2. Model Development with Considerations of Learning Factor and Negligent Factor

This research model is based on a real-world software testing scenario. Incorporating the learning factor in this model elucidates the mechanism behind the acceleration of software reliability growth, leading to the formation of an S-shaped mean value function. However, when the impact of the learning factor remains inconspicuous, the mean value function exhibits an exponential pattern. Notably, most previously published Software Reliability Growth Models (SRGMs) can only represent one of these two patterns, either S-shaped or exponential. In contrast, the model introduced in this study can represent both patterns. Furthermore, in the context of imperfect debugging, most previous studies employed a function simulating a rise in the number of initial software errors over time, often in the form of a(t)=a(1+ αt) (Pham et al., 1999). However, this assumption may not align with the realities of software testing, as the testing process does not introduce new software errors. Instead, the generation of new errors stems from the error-removal process, where previously unidentified software errors come to light. In other words, unless the program code is modified to address the error, new errors do not spontaneously emerge. This study explores the concept of software reliability growth from its foundational principles.

In general, the software reliability growth process is typically described using mathematical functions encompassing error detection and correction procedures. Many relevant studies commonly employ the Non-Homogeneous Poisson Process (NHPP) as a foundation for their models, guided by the following key assumptions:

(1) Immediate Removal of Detected Errors: It is assumed that once software errors are detected, they can be promptly and completely eliminated.

(2) No Introduction of New Errors: There is an assumption that no new errors are generated during the error correction process.

(3) NHPP for Error Detection: The error detection process is characterized as a Non-Homogeneous Poisson Process.

With the considerations above, the detection of software errors can be represented as a counting process, denoted as $\{N(t), t \geq 0\}$, where $N(t)$ adheres to a Poisson distribution with a mean value function $m(t)$. (t). This probability can be formally expressed as:

$$\Pr(N(t) = k) = \frac{[m(t)]^k e^{-m(t)}}{k!}, k = 0,1,2,\ldots \quad (1)$$

Looking at it from an alternate viewpoint, the mean value function can be represented as the anticipated count of errors detected over a specified time span $(0, t)$:

$$m(t) = \int_0^t \lambda(x)dx. \quad (2)$$

Additionally, software reliability, denoted as $R(x \mid t)$, is the probability of no error being detected during the interval $[t, t + x]$. It is important to note that the symbol x denotes a particular operational period in alignment with practical needs. Consequently, the formulation for $R(x \mid t)$ can be expressed as follows:

$$R(x \mid t) = e^{-[m(t+x)-m(t)]}, \quad (3)$$

and as the time *t* approaches infinity ($t \to \infty$) , the software reliability value approximates 1, indicating high reliability.

The study employs the following notations consistently:

Notations:

b: the autonomous errors-detected factor
γ: the learning factor
ε: the negligent factor
a: the initial number of all potential errors in the software system
σ: the standard deviation of the detection rate from the testing process
$f(t)$: the intensity function of the fraction of the errors detected at time t
$F(t)$: the cumulative function represents the fraction of errors detected during a specific time interval $(0, t]$
$m(t)$: the mean value function, and it represents the accumulated number of software errors detected during the time interval $(0, t)$
$\lambda(t)$: the intensity function that denotes the number of the errors detected at time t
$D(t)$: the function of the error detection rate
$R(x\|t)$: the software reliability which is defined as the probability of no error detected within the time interval $[t, t+x]$
$\omega(t)$: the function of the residual errors at testing time t
$B(t)$: the variable for the Brownian Motion

2.1 Model Development

In recent years, several studies have begun acknowledging the presence of a learning effect within the software testing process, though they have grappled with its precise identification. Consequently, quantifying and integrating these learning effects into software reliability growth models has emerged as a critical endeavor for software developers. Researchers like Huang et al. (2022) and Chiu et al. (2019) have explored this learning effect when formulating software reliability growth models under NHPP assumptions. However, their models have generally omitted the consideration of the negligent factor. The negligent factor pertains to the potential for software debugging staff to make errors while addressing software defects. This factor's significance is influenced by the number of unidentified error patterns within the software system, and its impact tends to diminish over time as the debugging experience grows through the recognition of previously detected defects. To illustrate our study's software reliability growth process, we have employed a causal loop diagram, presented in Fig. 1, which elucidates the logical concept of the software testing and debugging process.

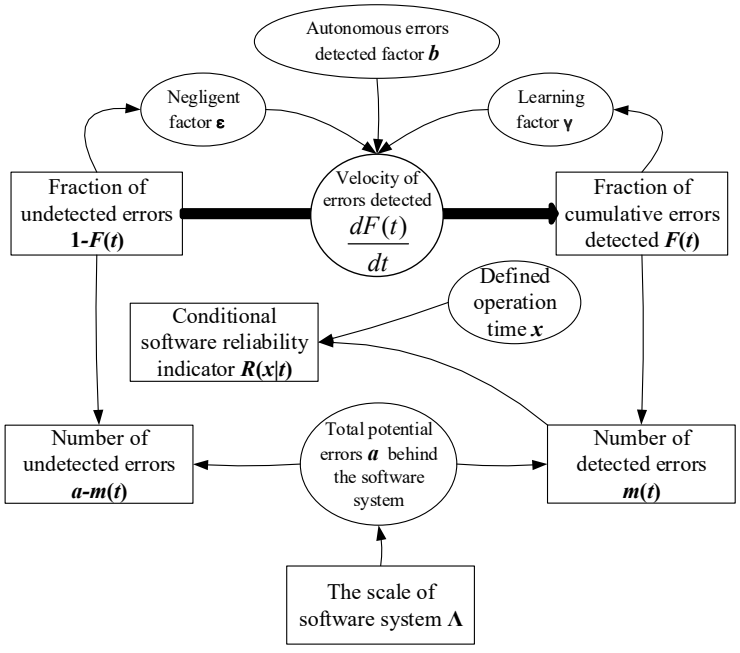

Fig. 1. Causal Loop Diagram of the Proposed SRGM

As delineated in Fig. 1, the pace at which errors are detected is linked to three key factors: the autonomous errors-detected factor denoted as b, the learning factor represented as γ and the negligent factor labeled as ε. Remarkably, the negligent factor ε chiefly contributes to imperfect debugging. Specifically, b implies that testing staff may serendipitously uncover hitherto unnoticed software errors, while γ signifies that testing staff can deliberately seek software errors based on previously recognized patterns. As such, the influence of the learning factor is intricately tied to the testing staff's expertise in error removal, with the process contingent upon how swiftly the erroneous pattern is identified. In this context, we define $F(t)$ as

the fraction of cumulative errors detected during the period (0, t). Consequently, the impact on the detection rate ($D(t)$) will be governed by $\gamma F(t)$. Conversely, the probability of testing staff making errors during the error removal process is linked to the fraction of unseen error patterns. Here, $1 - F(t)$ denotes the fraction of undetected errors at the time t, thus resulting in a decline in the detection rate of $\varepsilon(1 - F(t))$. The detection rate is shaped by three interrelated factors: b and γ exert a positive influence, while ε introduces a negative impact. The causal loop diagram facilitates the simplification of these interrelationships among the factors into a differential equation, articulated as follows:

$$D(t) = \frac{dF(t)/dt}{1 - F(t)} = \frac{f(t)}{1 - F(t)} = b + \gamma F(t) - \varepsilon\big(1 - F(t)\big). \tag{4}$$

The values of b, γ and ε are intricately connected to the experience and capabilities of the testing staff. Equation (4) signifies the existence of imperfect debugging and learning effects within the error detection and debugging process. Through a differential equation analysis, $f(t)$ can be derived from Eq. (4) by solving the following equation:

$$f(t) = \frac{\partial F(t)}{\partial t} = b - \varepsilon - (b - 2\varepsilon - \gamma)F(t) - (\varepsilon + \gamma)F(t)^2 \tag{5}$$

the explicit solutions of $F(t)$ and $f(t)$ are given by

$$F(t) = \frac{e^{(b+\gamma)t} + (b - \varepsilon)e^{(b+\gamma)c}}{e^{(b+\gamma)t} - (\varepsilon + \gamma)e^{(b+\gamma)c}} \tag{6}$$

and

$$f(t) = \frac{-(b + \gamma)^2 e^{(b+\gamma)(t+c)}}{(e^{(b+\gamma)t} - (\varepsilon + \gamma)e^{(b+\gamma)c})^2} \tag{7}$$

It is essential to note that when the software testing/debugging process commences, the number of system errors discovered should be zero. Consequently, a constant c adjusts the $F(t)$ function under the condition $F(0) = 0$. Based on the discussion above, the constant c can be computed as follows:

$$c = \frac{-\ln(\varepsilon - b)}{b + \gamma} \tag{8}$$

By substituting the constant c into Eq. (6) and Eq. (7), $F(t)$ and $f(t)$ can be rearranged as:

$$F(t) = \frac{(b - \varepsilon)(e^{(b+\gamma)t} - 1)}{(b - \varepsilon)e^{(b+\gamma)t} + \varepsilon + \gamma} \tag{9}$$

and

$$f(t) = \frac{(b - \varepsilon)(b + \gamma)^2 e^{(b+\gamma)t}}{((b - \varepsilon)e^{(b+\gamma)t} + \varepsilon + \gamma)^2} \tag{10}$$

It is important to note that to ensure a gradual decrease in software errors through the debugging process, the values of b, γ and ε must all be positive. Furthermore, for practical consistency, it is advisable for b to be greater than ε to avoid the negligent factor's velocity surpassing that of the autonomous errors-detected factor. Additionally, the size of the software system Λ influences the estimation of the mean value function, and thus, an initial estimation of the total potential errors is required. Here, a represents the anticipated count of potential errors, deduced from the software system's size and past testing project experience. Accordingly, the mean value function for the software error detection process can be expressed as follows:

$$m(t) = aF(t) = a\left(\frac{(b - \varepsilon)\big(e^{(b+\gamma)t} - 1\big)}{(b - \varepsilon)e^{(b+\gamma)t} + \varepsilon + \gamma}\right), \tag{11}$$

The intensity function at time *t* is given by:

$$\lambda(t) = af(t) = a\left(\frac{(b - \varepsilon)(b + \gamma)^2 e^{(b+\gamma)t}}{((b - \varepsilon)e^{(b+\gamma)t} + \varepsilon + \gamma)^2}\right). \tag{12}$$

In order to capture the fluctuations in the error detection rate per error at time t, we define the error detection rate as:

$$D(t) = \frac{\lambda(t)}{a - m(t)} = \frac{(b-\varepsilon)(b+\gamma)e^{(b+\gamma)t}}{(b-\varepsilon)e^{(b+\gamma)t} + (\gamma+\varepsilon)} = \frac{(b+\gamma)e^{(b+\gamma)t}}{\frac{\gamma+\varepsilon}{b-\varepsilon} + e^{(b+\gamma)t}} \quad (13)$$

It is worth noting that the boundaries of the error detection rate fall within the range of $b-\varepsilon$ to $b+\gamma$ at any given testing time, signifying an improvement in the error detection rate over time in real-world scenarios.

***Proposition 1**:*

Considering the error detection rate defined as $D(t) = \frac{(b+\gamma)e^{(b+\gamma)t}}{\frac{\gamma+\varepsilon}{b-\varepsilon}+e^{(b+\gamma)t}}$, where the parameters b, γ, ε are all positive, and $b > \varepsilon$, it is evident that the error detection rate consistently exhibits an ascending trajectory. The boundaries of this rate consistently fall within the range of $b-\varepsilon$ to $b+\gamma$.

Proof:

To ascertain whether the function $D(t)$ exhibits strict monotonicity (i.e., it is strictly increasing), we can derive the first-order derivative of $D(t)$ concerning the testing time t resulting in the following equation:

$$\frac{\partial D(t)}{\partial t} = \frac{(b-\varepsilon)e^{(b+\eta)t}(b+\gamma)^2(\varepsilon+\gamma)}{((b-\varepsilon)e^{(b+\eta)t} + (\varepsilon+\gamma))^2} > 0 \ (\because\ b > \varepsilon)$$

Since $\frac{\partial D(t)}{\partial t} > 0$, it is evident that the error detection rate consistently demonstrates an ascending trend regardless of the testing time t.

Moreover, the lower and upper boundaries of the error detection rate can be determined by evaluating the limits as t approaches zero ($\lim_{t\to 0} D(t)$) and as t tends toward infinity ($\lim_{t\to\infty} D(t)$).

The lower boundary:

$$\lim_{t\to 0} D(t) = \lim_{t\to 0} \frac{(b+\gamma)e^{(b+\gamma)t}}{\frac{\gamma+\varepsilon}{b-\varepsilon} + e^{(b+\gamma)t}} = \frac{b+\gamma}{\frac{\varepsilon+\gamma}{b-\varepsilon} + 1} = b-\varepsilon$$

The upper boundary:

$$\lim_{t\to\infty} D(t) = \lim_{t\to\infty} \frac{(b+\gamma)e^{(b+\gamma)t}}{\frac{\gamma+\varepsilon}{b-\varepsilon} + e^{(b+\gamma)t}}$$

By using L'Hôpital's rule, it can be obtained

$$\lim_{t\to\infty} D(t) = \lim_{t\to\infty} \frac{(b+\gamma)^2 e^{(b+\gamma)t}}{(b+\gamma)e^{(b+\gamma)t}} = b+\gamma$$

The differentiation between exponential-shaped and S-shaped behaviors hinges on whether the value of the inflection point is greater than zero. The proposed model exhibits an S-shaped pattern when $\gamma+\varepsilon > b-\varepsilon$ (indicating a positive inflection point). Conversely, it takes an exponential shape when $\gamma+\varepsilon \le b-\varepsilon$. An inflection point suggests that the learning effect plays a relatively more significant role than the autonomous errors-detected rate during the testing and debugging. The exact inflection time point is determined by

$$t_f = \frac{\ln(\gamma+\varepsilon) - \ln(b-\varepsilon)}{b+\gamma} \quad (14)$$

***Proposition 2**:*

Under the premise that the mean value function $m(t)$ is continuous and differentiable, and the condition $\gamma+\varepsilon > b-\varepsilon$ is satisfied, an inflection point can be identified at the time represented by $\frac{ln(\gamma+\varepsilon)-ln(b-\varepsilon)}{b+\gamma}$. Additionally, the maximum value of the intensity function will be $\frac{a(b+\gamma)^2}{4(\varepsilon+\gamma)}$. This maximum value underscores the significance of the learning effect in the software debugging process, particularly in driving the emergence of an S-shaped behavior.

Proof:

Referring to the mathematical form of Eq. (11), it is evident that the mean value function $\mathrm{m}(t)$ *maintains its continuity and differentiability at any given time t within the defined parameters of* $b \geq 0, \gamma \geq 0$ *and* $\varepsilon \geq 0$*. To ascertain the inflection time point within the specified timeframe, we can determine it by computing the second-order derivative of* $\mathrm{m}(t)$ *concerning time t, which results in the following equation:*

$$\frac{\partial^2 m(t)}{\partial t^2} = \frac{a(b-\varepsilon)(b+\gamma)^3 e^{(b+\gamma)t}\left((\gamma+\varepsilon)-(b-\varepsilon)e^{(b+\gamma)t}\right)}{((b-\varepsilon)e^{t(b+\gamma)}+(\gamma+\varepsilon))^3}$$

The inflection point is determined by solving the equation $\frac{\partial^2 m(t)}{\partial t^2} = 0$*for t, and the maximum value of the intensity function is obtained at* $t = \frac{\ln(\gamma+\varepsilon)-\ln(b-\varepsilon)}{b+\gamma}$*. As long as the condition* $\gamma + \varepsilon > b - \varepsilon$ *holds, the inflection time point must be greater than zero to signify the presence of a learning effect (∵* $\ln(\gamma+\varepsilon) - \ln(\gamma-\varepsilon) > 0$*). Furthermore, by substituting the solution* $= \frac{\ln(\gamma+\varepsilon)-\ln(b-\varepsilon)}{b+\gamma}$ *into the intensity function* $\lambda(t)$*, the maximum value of* $\lambda(t)$ *can be calculated as* $\frac{b(b+\gamma)^2}{4(\varepsilon+\gamma)}$*.*

Proposition 3*:*

Assuming that the mean value function $\mathrm{m}(t)$ *remains continuous and differentiable, and given the condition* $\gamma + \varepsilon > b - \varepsilon$*, an inflection point exists at the time represented by* $\frac{ln(\gamma+\varepsilon)-ln(b-\varepsilon)}{b+\gamma}$*. Furthermore, the maximum value of the intensity function is* $\frac{b(b+\gamma)^2}{4(\varepsilon+\gamma)}$*, indicating the pronounced impact of the learning effect in the software debugging process on the emergence of an S-shaped behavior.*

Proof:

Based on the mathematical form of equation (11), it is established that the mean value function $\mathrm{m}(t)$ *maintains its continuity and differentiability over time t within the specified parameters of* $b \geq 0, \gamma \geq 0$*, and* $\varepsilon \geq 0$*. To determine the inflection time point within the planned horizon, we can derive the second-order derivative of* $\mathrm{m}(t)$ *concerning time t, yielding the following equation:*

$$\frac{\partial^2 m(t)}{\partial t^2} = \frac{a(b-\varepsilon)(b+\gamma)^3 e^{(b+\gamma)t}\left((\gamma+\varepsilon)-(b-\varepsilon)e^{(b+\gamma)t}\right)}{((b-\varepsilon)e^{(b+\gamma)t}+(\gamma+\varepsilon))^3}$$

The inflection point can be identified by solving the equation $\frac{\partial^2 m(t)}{\partial t^2} = 0$ *for t, and the maximum value of the intensity function is attained at* $t = \frac{\ln(\gamma+\varepsilon)-\ln(b-\varepsilon)}{b+\gamma}$*. Given that the condition* $\gamma + \varepsilon > b - \varepsilon$ *is met, the inflection time point must be greater than zero to indicate the presence of the learning effect (∵* $\ln(\gamma+\varepsilon) - \ln(b-\varepsilon) > 0$*). Furthermore, by substituting the solution* $\mathrm{t} = \frac{\ln(\gamma+\varepsilon)-\ln(b-\varepsilon)}{b+\gamma}$ *into the intensity function* $\lambda(t)$*, the maximum value of* $\lambda(t)$ *can be calculated as* $\frac{b(b+\gamma)^2}{4(\varepsilon+\gamma)}$*.*

To measure software reliability during the testing phase, we adopt the definition provided by Zhang and Pham (2006) to determine the reliability function of the study, as defined below:

$$R(x \mid t) = e^{-[m(t+x)-m(t)]} = \exp\left[a\left(\frac{(b-\varepsilon)(e^{(b+\gamma)t}-1)}{(b-\varepsilon)e^{(b+\gamma)t}+\varepsilon+r} - \frac{(b-\varepsilon)(e^{(b+\gamma)(t+x)}-1)}{(b-\varepsilon)e^{(b+\gamma)(t+x)}+\varepsilon+\gamma}\right)\right], \tag{15}$$

2.2 Parameters Estimation

In this study, we employ two primary methods for estimating the parameters of the various software reliability growth models: the Least Squares Estimation (LSE) and Maximum Likelihood Estimation (MLE). The LSE method is used to estimate the parameters a, b, γ and ε based on n+1 observed data pairs: $(t_0, m_0), (t_1, m_1), (t_2, m_2), \ldots, (t_n, m_n)$, where m_i represents the total number of errors detected within the time interval $(0, t_i)$. For LSE, we formulate the objective function $Er(a, b, \gamma, \varepsilon)$ as the sum of squared differences between the observed data points and the model's predictions. The objective function is minimized as follows:

$$\min Er(a\ b\ \gamma\ \varepsilon) = \sum_{i=1}^{n}(m_i - m(t_i))^2 = \sum_{i=1}^{n}\left(m_i - a\left(\frac{(b-\varepsilon)(e^{(b+\gamma)t}-1)}{(b-\varepsilon)e^{(b+\gamma)t}+\varepsilon+\gamma}\right)\right)^2. \tag{16}$$

We differentiate this objective function concerning a, b, γ and ε, setting the partial derivatives to zero to obtain the estimates for these parameters.

$$\frac{\partial Er(a,b,\gamma,\varepsilon)}{\partial a} = \sum_{i=1}^{n}\left(m_i - a\left(\frac{(b-\varepsilon)(e^{(b+\gamma)t}-1)}{(b-\varepsilon)e^{(b+\gamma)t}+\varepsilon+\gamma}\right)\right)\left(\frac{(b-\varepsilon)(e^{(b+\gamma)t}-1)}{(b-\varepsilon)e^{(b+\gamma)t}+\varepsilon+\gamma}\right) = 0, \tag{17}$$

$$\frac{\partial Er(a,b,\gamma,\varepsilon)}{\partial b} = \sum_{i=1}^{n}\left(m_i - a\left(\frac{(b-\varepsilon)(e^{(b+\gamma)t}-1)}{(b-\varepsilon)e^{(b+\gamma)t}+\varepsilon+\gamma}\right)\right)\left(\frac{ae^{(b+\gamma)t}(\varepsilon+\gamma+(b-\varepsilon)(b+\gamma)t)-a(\varepsilon+\gamma)}{(e^{(b+\gamma)t}(b-\varepsilon)+\varepsilon+\gamma)^2}\right) = 0, \tag{18}$$

$$\frac{\partial Er(a,b,\gamma,\varepsilon)}{\partial \gamma} = \sum_{i=1}^{n}\left(m_i - a\left(\frac{(b-\varepsilon)(e^{(b+\gamma)t}-1)}{(b-\varepsilon)e^{(b+\gamma)t}+\varepsilon+\gamma}\right)\right)\left(\frac{a(b-\varepsilon)(1+e^{(b+\gamma)t}(t(b+\gamma)-1))}{((b-\varepsilon)e^{(b+\gamma)t}+\varepsilon+\gamma)^2}\right) = 0 \text{ and} \tag{19}$$

$$\frac{\partial Er(a,b,\gamma,\varepsilon)}{\partial \varepsilon} = \sum_{i=1}^{n}\left(m_i - a\left(\frac{(b-\varepsilon)(e^{(b+\gamma)t}-1)}{(b-\varepsilon)e^{(b+\gamma)t}+\varepsilon+\gamma}\right)\right)\left(\frac{a(e^{(b+\gamma)t}-1)(b+\gamma)}{((b-\varepsilon)e^{(b+\gamma)t}+\varepsilon+\gamma)^2}\right) = 0 \tag{20}$$

MLE is the second method for estimating unknown parameters. The likelihood function for the proposed model is expressed as:

$$\begin{aligned} L &= Pr\{N(t_1) = m_1, N(t_2) = m_2, N(t_3) = m_3, \ldots, N(t_n) = m_n\} \\ &= \prod_{i=1}^{n} \frac{(m(t_i)-m(t_{i-1}))^{(m_i-m_{i-1})}(e^{-(m(t_i)-m(t_{i-1}))})}{(m_i-m_{i-1})!}. \end{aligned} \tag{21}$$

Taking the natural logarithm of the likelihood function, we have:

$$\ln(L) = \sum_{i=1}^{n}(m_i - m_{i-1})\ln(m(t_i) - m(t_{i-1})) - \sum_{i=1}^{n}\big(m(t_i) - m(t_{i-1})\big) - \sum_{i=1}^{n}\ln\big((m_i - m_{i-1})!\big). \tag{22}$$

We further simplify ln(L) based on Equation 20 to obtain:

$$\begin{aligned} &\ln\big(m(t_i) - m(t_{i-1})\big) \\ &= \ln\left(a(b+\gamma)\left(\frac{1}{(b-\varepsilon)e^{(b+\gamma)t_{i-1}} + \varepsilon + \gamma} - \frac{1}{(b-\varepsilon)e^{(b+\gamma)t_i} + \varepsilon + \gamma}\right)\right) \\ &= \ln(a) + \ln(b+\gamma) + \ln\left(\frac{1}{(b-\varepsilon)e^{(b+\gamma)t_{i-1}+\varepsilon+\gamma} - \frac{1}{(b-\varepsilon)e^{(b+\gamma)t_i+\varepsilon+\gamma}}}\right). \end{aligned} \tag{23}$$

Thus,

$$\begin{aligned} &\ln(L) = \sum_{i=1}^{n}(m_i - m_{i-1})\left(\ln(a) + \ln(b+\gamma) + \ln\left[\frac{1}{(b-\varepsilon)e^{(b+\gamma)t_{i-1}+\varepsilon+\gamma} - \frac{1}{(b-\varepsilon)e^{(b+\gamma)t_i+\varepsilon+\gamma}}}\right]\right) - \\ &a\left(\frac{(b-\varepsilon)(e^{(b+\gamma)t_n}-1)}{(b-\varepsilon)e^{(b+\gamma)t_n+\varepsilon+\gamma}}\right) - \sum_{i=1}^{n}\ln((m_i - m_{i-1})!). \end{aligned} \tag{24}$$

The estimators for the parameters a, b, γ and ε are obtained by solving the equations $\frac{\partial \ln(L)}{\partial a} = \frac{\partial \ln(L)}{\partial b} = \frac{\partial \ln(L)}{\partial \gamma} = \frac{\partial \ln(L)}{\partial \varepsilon} = 0$ using numerical methods. In addition to parameter estimation, we calculate confidence intervals for the parameters θ_1,θ_2,..,θ_k using the variance-covariance matrix $\sum$. The Fisher information matrix F is used to derive $\sum$. Then, we determine the confidence intervals for the estimated parameters θ_i using the two-sided approximate 100**CR*% confidence limits, where $t_{\frac{CR}{2},n-k}$ is the critical value from the student-*t* distribution.

$$\boldsymbol{F} = \begin{bmatrix} E\left[\frac{-\partial^2 \ln(L)}{\partial \theta_1^2}\right] & \cdots & E\left[\frac{-\partial^2 \ln(L)}{\partial \theta_1 \partial \theta_k}\right] \\ \cdots & \cdots & \cdots \\ E\left[\frac{-\partial^2 \ln(L)}{\partial \theta_k \partial \theta_1}\right] & \cdots & E\left[\frac{-\partial^2 \ln(L)}{\partial \theta_k^2}\right] \end{bmatrix}. \tag{25}$$

$$\sum = \boldsymbol{F}^{-1} = \begin{bmatrix} Var[\hat{\theta}_1] & \dots & Cov[\hat{\theta}_1, \hat{\theta}_k] \\ \dots & \dots & \dots \\ Cov[\hat{\theta}_k, \hat{\theta}_1] & \dots & Var[\hat{\theta}_k] \end{bmatrix}. \tag{26}$$

$$\hat{\theta}_i \pm t_{\frac{CR}{2},n-k}\sqrt{Var[\hat{\theta}_i]},\ i=1..k, \tag{27}$$

3. Model Development based on Stochastic Differential Equation

In previous research, the widely adopted approach for calculating confidence intervals (CIs) relied on the estimation method introduced by Yamada and Osaki in 1985. However, it is essential to note that this method was primarily developed for assessing hardware deterioration based on Non-Homogeneous Poisson Processes (NHPP). In the context of this method, systems degrade over time, leading to a gradual increase in errors and subsequently causing confidence intervals to expand with time. Yamada and Osaki's definition for estimating confidence intervals in the context of software error detection during testing is as follows:

$$\hat{m}(T) \pm Z_{CR/2}\sqrt{\hat{m}(T)}, \tag{28}$$

where CR represents the two-tail critical region, $\sqrt{\hat{m}(T)}$ is the standard deviation of the mean value function $m(T)$, and $Z_{CR/2}$ is the critical value corresponding to a specific $CR/2$ area in the standard normal distribution. It is worth emphasizing that these confidence intervals naturally widen over time because the standard deviation $\sqrt{\hat{m}(T)}$ is positively linked with time T. However, in the context of software systems, the number of remaining errors tends to decrease over time. This fact contradicts the notion that error deviations should increase with time. To illustrate this distinction, Fig. 2 visually compares the conventional and proposed confidence intervals. This graphical representation underscores that traditional CIs tend to expand over time, even though error deviations are, in reality, converging and not escalating with time. Consequently, the CIs deduced in our proposed model align more closely with the actual data. The research presented in Section 3.1 outlines the process of deriving these proposed CIs.

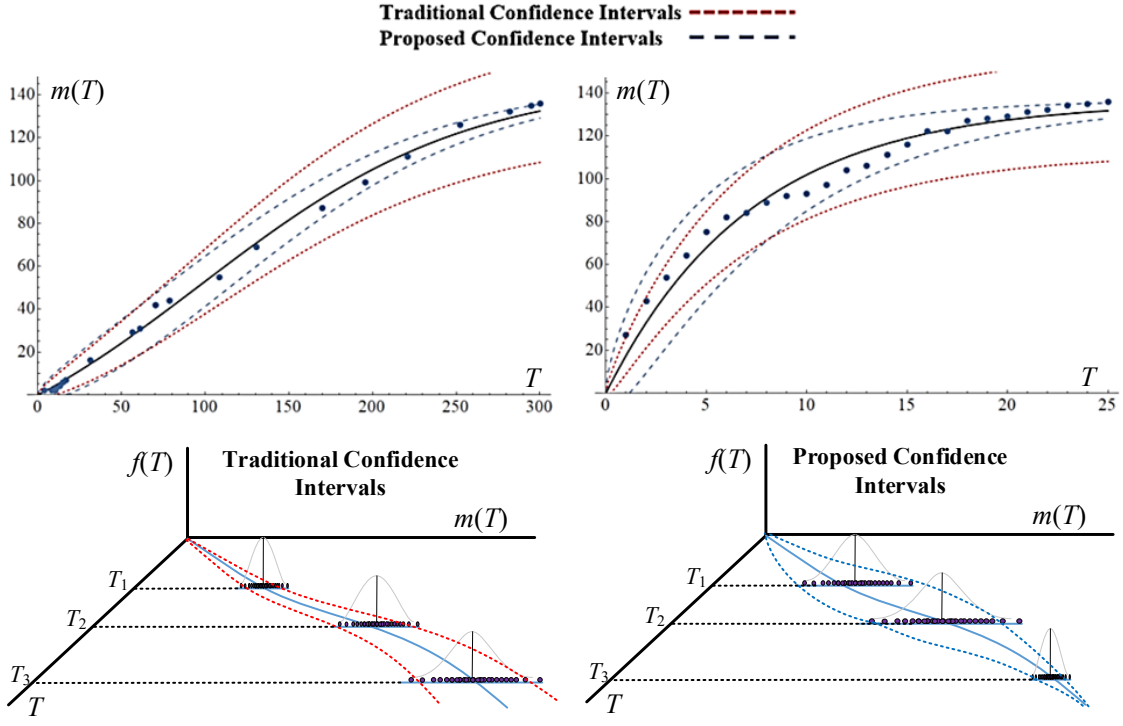

Fig. 2. The Difference between the Traditional and the Proposed Confidence Intervals with *CR*=5%.

3.1 Model Development

For deriving the proposed confidence intervals, certain additional assumptions must be made:

(1) Irregular fluctuations stem from the error detection rate.

(2) These irregular fluctuations follow Brownian motion and a Normal distribution.

Firstly, let $\omega(t) = a - m(t)$. We assume that the irregular fluctuation is related to the error detection rate during the testing period and can be expressed as a stochastic differential equation (SDE):

$$D(t) + \sigma dB(t) = \frac{\lambda(t)}{a-m(t)} + \sigma dB(t) = \frac{\frac{dm(t)}{dt}}{a-m(t)} + \sigma dB(t), \tag{29}$$

where $B(t)$ represents Brownian motion, and σ is a positive constant representing the magnitude of the irregular fluctuation.

Substituting $\omega(t)$ for $a - m(t)$, Equation (29) can be rewritten as:

$$\frac{\omega'(t)}{\omega(t)} = -\big(D(t) + \sigma dB(t)\big) \tag{30}$$

Let $\Psi(t) = \ln[\omega(t)]$. Using Itô's calculus, we obtain the following equation:

$$d\Psi(t) = \left(-D(t) - \frac{1}{2}\sigma^2\right)dt - \sigma dB(t) \tag{31}$$

By integrating from 0 to T, we can determine $\Psi(T)$ as follows:

$$\int_0^T d\Psi(t) = -\int_0^T D(t)\,dt - \int_0^T \frac{\sigma^2}{2}dt - \int_0^T \sigma dB(t) \tag{32}$$

$$\rightarrow \Psi(t)|_0^T = -\int_0^T D(t)\,dt - \int_0^T \frac{\sigma^2}{2}dt - \int_0^T \sigma dB(t)$$

$$\rightarrow \ln[\omega(T)] - \ln[\omega(0)] = -\int_0^T D(t)\,dt - \int_0^T \frac{\sigma^2}{2}dt - \int_0^T \sigma dB(t)$$

$$\rightarrow \ln[\omega(T)] - \ln[a] = -\int_0^T D(t)\,dt - \int_0^T \frac{\sigma^2}{2}dt - \int_0^T \sigma dB(t) \text{ (as } \omega(0) = a\text{, the initial number of software errors must be a)}$$

$$\rightarrow \ln[\omega(T)] = \ln[a] - \int_0^T D(t)\,dt - \int_0^T \frac{\sigma^2}{2}dt - \int_0^T \sigma dB(t)$$

$$\rightarrow e^{\ln[\omega(T)]} = e^{\ln[a] - \int_0^T D(t)dt - \int_0^T \frac{\sigma^2}{2}dt - \int_0^T \sigma dB(t)}$$

$$\rightarrow \omega(T) = ae^{-\int_0^T D(t)dt - \frac{\sigma^2 T}{2} - \sigma B(T)} \tag{33}$$

Assume $\omega(T)$ is a random variable. The corresponding expectation can be determined as:

$$E[\omega(T)] = E\left[ae^{-\int_0^T D(t)dt - \frac{\sigma^2 T}{2} - \sigma B(T)}\right] = ae^{-\int_0^T D(t)dt - \frac{\sigma^2 T}{2}} E[e^{-\sigma B(T)}]. \tag{34}$$

Since $E\left[e^{-\sigma W(T)}\right] = \int_{-\infty}^{+\infty}(e^{-\sigma z})\left(\frac{1}{\sqrt{2\pi T}}e^{-\frac{Z^2}{2T}}\right)dz = e^{\frac{\sigma^2 T}{2}}$, Eq. (6) can be simplified as:

$$E[\omega(T)] = ae^{-\int_0^T D(t)dt - \frac{\sigma^2 T}{2}}\left(e^{\frac{\sigma^2 T}{2}}\right) = ae^{-\int_0^T D(t)dt}. \tag{35}$$

Consequently, the expectation of $m(T)$ can be expressed as:

$$\begin{aligned} E[m(T)] = E[a - \omega(T)] &= a - ae^{-\int_0^T D(t)dt} = a - ae^{-\int_0^T \frac{d\ln[\omega(T)]}{dt}dt} \\ &= a - ae^{\ln[\omega(T)] - \ln[\omega(0)]} = a - a\big(\omega(T)\big)a^{-1} = a - \omega(T) = m(T). \end{aligned} \tag{36}$$

Similarly, the variance of $m(T)$ can be dstermined as:

$$Var[m(T)] = Var[\omega(T)] = E[\omega(T)^2] - E[\omega(T)]^2 \tag{37}$$

To calculate $Var[m(T)]$, we need to find $E[\omega(T)^2]$ and $E[\omega(T)]^2$. The process of deducing $E[\omega(T)^2]$ and $E[\omega(T)]^2$ is as follows:

$$E[\omega(T)^2] = E\left[\left(ae^{-\int_0^T D(t)dt-\frac{\sigma^2 T}{2}-\sigma B(T)}\right)^2\right] = a^2 e^{-2\int_0^T D(t)dt-\sigma^2 T-2\sigma B(T)} = a^2 e^{-2\int_0^T D(t)dt}\left(e^{-\sigma^2 T}\right)E[e^{-2\sigma B(T)}] \quad (38)$$

Since $E\left[e^{-2\sigma B(T)}\right] = e^{2\sigma^2 T}$ (please see the proof of proposition 4), Equation (38) can be simplified as:

$$\begin{aligned} E[\omega(T)^2] &= a^2 e^{-2\int_0^T D(t)dt}\left(e^{-\sigma^2 T}\right)E\left[e^{-2\sigma B(T)}\right] \\ &= a^2 e^{-2\int_0^T D(t)dt}\left(e^{-\sigma^2 T}\right)\left(e^{2\sigma^2 T}\right) = a^2 e^{-2\int_0^T D(t)dt}\left(e^{\sigma^2 T}\right) \end{aligned} \quad (39)$$

Likewise, referring to Eq. (35), since $E[\omega(T)] = ae^{-\int_0^T D(t)dt}$, we can intuitively obtain $E[\omega(T)]^2$ as follows:

$$E[\omega(T)]^2 = a^2 e^{-2\int_0^T D(t)dt} \quad (40)$$

Proposition 4*:*

As $B(T)$ represents a variable in the context of Brownian motion, the expected value $E\left[e^{-2\sigma B(T)}\right]$ *can be obtained as* $e^{2\sigma^2 T}$.

Proof:

Since $B(T)$ follows Normal distribution, the expected value can be obtained as the following equation:

$$E\left[e^{-2\sigma B(T)}\right] = \int_{-\infty}^{+\infty}(e^{-2\sigma z})\left(\frac{1}{\sqrt{2\pi T}}e^{-\frac{z^2}{2T}}\right)\mathrm{d}z = \int_{-\infty}^{+\infty}\left(\frac{1}{\sqrt{2\pi T}}e^{-\frac{z^2}{2T}-2\sigma z}\right)\mathrm{d}z = \frac{1}{\sqrt{2\pi T}}\int_{-\infty}^{+\infty}\left(e^{2\sigma^2 t-\left(\frac{z+2\sigma T}{\sqrt{2T}}\right)^2}\right)\mathrm{d}z.$$

Substitute $u = \frac{z+2\sigma T}{\sqrt{2T}} \rightarrow \frac{\mathrm{d}u}{\mathrm{d}z} = \frac{1}{\sqrt{2T}} \rightarrow \mathrm{d}z = \sqrt{2T}\mathrm{d}u$.

Therefore, the above formulation can be transformed into $\frac{1}{\sqrt{2\pi T}}\int_{-\infty}^{+\infty}\left(e^{2\sigma^2 T-u^2}\right)\sqrt{2T}\mathrm{d}u$.

After the arrangement, the above formulation will be

$$\frac{1}{\sqrt{2\pi T}}\int_{-\infty}^{+\infty}e^{2\sigma^2 T}\left(e^{-u^2}\right)\sqrt{2T}\mathrm{d}u = \frac{e^{2\sigma^2 T}}{\sqrt{\pi}}\int_{-\infty}^{+\infty}\left(e^{-u^2}\right)\mathrm{d}u.$$

Since $\int_{-\infty}^{+\infty}\left(e^{-u^2}\right)\mathrm{d}u = \sqrt{\pi}$, *the original formulation can be obtained as follows:*

$$\frac{e^{2\sigma^2 T}}{\sqrt{\pi}}\int_{-\infty}^{+\infty}\left(e^{-u^2}\right)\mathrm{d}u = \frac{e^{2\sigma^2 T}}{\sqrt{\pi}}\sqrt{\pi} = e^{2\sigma^2 T}.$$

Accordingly, the equation $E\left[e^{-2\sigma B(T)}\right] = e^{2\sigma^2 T}$ *can be proved.*

Hence, in accordance with Eq. (39) and Eq. (40), the variance of $m(T)$ can be expressed as:

$$\begin{aligned} Var[m(T)] &= a^2 e^{-2\int_0^T D(t)dt}\left(e^{\sigma^2 T}\right) - a^2 e^{-2\int_0^T D(t)dt} = a^2 e^{-2\int_0^T D(t)dt}\left(e^{\sigma^2 T}-1\right) \\ &= \left(ae^{-\int_0^T D(t)dt}\right)^2\left(e^{\sigma^2 T}-1\right) = (E[\omega(T)])\left(e^{\sigma^2 T}-1\right) = (a-E[m(T)])^2\left(e^{\sigma^2 T}-1\right) \\ &= \left(a-m(T)\right)^2\left(e^{\sigma^2 T}-1\right) \end{aligned} \quad (41)$$

Furthermore, the integration of the detection rate $D(t)$ under the definition in $[0, T]$ can be calculated as shown below:

$$\int_0^T D(t)\,dt = \int_0^T \frac{(b+\gamma)e^{(b+\gamma)t}}{\frac{\gamma+\varepsilon}{b-\varepsilon}+e^{(b+\gamma)t}}dt = \ln\left[be^{(b+\gamma)T}+\frac{\gamma+\varepsilon}{b+\gamma}\right] \quad (42)$$

Furthermore, understanding how to compute the variance of the fault detection rate, σ^2, using actual software testing data is crucial for deriving the confidence intervals of $m(T)$. Estimating the standard deviation from the testing process, denoted as $\hat{\sigma}$, can be obtained by the relationship between $Var[m(T)]$ and σ in Eq. (41), which is represented as:

$$\hat{\sigma}^2 = \sum_{i=1}^{n}(1/t_i)\ln\left[\frac{(m_i-\hat{m}(t_i))^2}{n-1}/(\hat{a}-\hat{m}(t_i))^2+1\right]. \quad (43)$$

Hence, the confidence intervals can be estimated using the unspecified parameters and $\hat{\sigma}^2$.

3.2 Estimation of Confidence Intervals of the Proposed Model

In Section 3.1, the analysis considers the variance in software debugging efficiency attributed to the error detection rate. The construction of confidence intervals for the mean value function is based on Eq. (36) and Eq. (41). The following equation outlines the proposed confidence intervals for the mean value function:

$$E[m(T)] \pm t_{\frac{CR}{2},n-k}\sqrt{Var[m(T)]\left(1+\frac{1}{n}+\frac{\left(T-\bar{\mathrm{t}}\right)^2}{\sum_{i=1}^{n}\left(t_i-\bar{\mathrm{t}}\right)^2}\right)} \tag{44}$$

$$= \hat{m}(T) \pm t_{CR/2,n-k}\sqrt{(\hat{a}-\hat{m}(T))^2\left(e^{\sigma^2 T}-1\right)\left(1+\frac{1}{n}+\frac{(T-\bar{\mathrm{t}})^2}{\sum_{i=1}^{n}(t_i-\bar{\mathrm{t}})^2}\right)},$$

Here, $t_{CR/2,n-k}$ represents the critical value of the Student's t-distribution with $n-k$ degrees of freedom. To simplify, the upper and lower confidence intervals for $m(T)$ are denoted as $m_{UB}^{CR}(T)$ and $m_{LB}^{CR}(T)$, respectively.

In summary, the traditional model assumes that the variance in the number of software errors detected is linked to $m(T)$, while the proposed model attributes it to $D(t)$. As a result, the confidence intervals in the traditional model exhibit divergence over time, whereas they converge in the proposed model. It explains the decreasing variance of software errors during later testing periods, as depicted in Figure 1. Furthermore, the upper and lower confidence intervals for software reliability, $R(x/T)$, can be deduced from $m_{UB}^{CR}(T)$ and $m_{LB}^{CR}(T)$, as indicated in Eq. (45) and Eq. (46).

$$R_{UB}^{CR}(x/T) = e^{-[m_{UB}^{CR}(T+x)-m_{UB}^{CR}(T)]} \tag{45}$$

$$R_{LB}^{CR}(x/T) = e^{-[m_{LB}^{CR}(T+x)-m_{LB}^{CR}(T)]} \tag{46}$$

4. Models Validation

To validate the suitability of our proposed model, we conducted a comparative analysis with three classic perfecting Software Reliability Growth Models (SRGMs), as presented in Table 1. The SRGM by Wang et al. (2015) employs five parameters, while other SRGMs (Pham et al., 1999; Kapur et al., 2008; our model) use four parameters. In general, models with more parameters have better fitting capabilities. Parameter estimation in this study was performed using the least squares estimation (LSE) method. We evaluated the fitting abilities of these models using eight datasets from prior research, as outlined in Table 2. Detailed information about these datasets is provided in Table 3.

To assess the effectiveness of these models, we employed three criteria:

(1) Mean Square Error (MSE): The definition is $MSE = \frac{\sum_{i=1}^{n}(m_i-m(t_i))^2}{n-k}$, where m_i represents the actual cumulative number of errors at time t_i, $m(t_i)$ is the estimated cumulative number of errors at time t_i derived from the fitted mean value function, n is the number of observations, and k is the number of model parameters.

(2) R-squared (Rsq): The definition is $Rsq = 1 - \frac{\sum_{i=1}^{n}\left(m_i-m(t_i)\right)^2}{\sum_{i=1}^{n}\left(m_i-\sum_{j=1}^{n}m_j/n\right)^2}$.

(3) Akaike information criterion (AIC) (Akaike, 1973): AIC is a measure defined as the log-likelihood term penalized by the number of model parameters and is given by $\mathrm{AIC} = n^*\ln\left[\frac{\mathrm{RSS}}{n}\right]+2k$.

Table 1
Summary of Mean Value and Error Detection Rate Functions for the Imperfect Debugging SRGMs

Imperfect Debugging SRGMs	$\boldsymbol{m(t)}$ **and** $\boldsymbol{d(t)}$
Pham et al. (1999)	$m(t) = \frac{a(1-e^{-bt})(1+\alpha t+\frac{\alpha t}{b})}{1+e^{-bt}\beta}$ and $d(t) = \frac{b}{1+\beta e^{-bt}}$.

Kapur et al. (2008)	$m(t) = \frac{a(1-e^{-bp(1-\alpha)t})}{1-\alpha}$ and $d(t) = \frac{(1-\alpha)bp}{1-\alpha e^{-bpt(-1+\alpha)}}$.
Wang et al. (2015)	$m(t) = \frac{a(\alpha t)^d}{1+(\alpha t)^d} + C(1-e^{-bt}) - ad(\alpha t)^d e^{-bt}\left(\frac{1}{d} + \frac{bt}{1+d} + \frac{b^2t^2}{2(2+d)}\right)$ and $d(t) = b$.
Proposed model	$m(t) = a\left(\frac{(b-\varepsilon)(e^{(b+\gamma)t}-1)}{(b-\varepsilon)e^{(b+\gamma)t}+\varepsilon+\gamma}\right)$ and $d(t) = \frac{(b+\gamma)e^{(b+\gamma)t}}{\frac{\gamma+\varepsilon}{b-\varepsilon}+e^{(b+\gamma)t}}$.

Table 2
Datasets Used for Validation

Dataset	Reference	Source
[1]	Zhang & Pham (2006)	Failure data of Telecommunication system
[2]	Yang et al. (2016)	Mozilla Firefox 3.5 testing data
[3]	Wang et al.(2016)	Medium scale software project
[4]	Hsu et al.(2011)	Open source project management software
[5]	Peng et al.(2014)	Testing data for the Room Air Development Center
[6]	Zhang & Pham (1998)	Failure data of Misra system
[7]	Shyur (2003)	Failure data of Misra system
[8]	Singpurwalla & Willson (1999)	Failure data of NTDS system

Table 3
The Detail of the Datasets

Data1={{0.5,5},{1,5},{1.4,5},{1.9,10},{2.4,15},{2.9,15},{3.3,15},{3.8,25},{4.3,30},{4.8,40},{5.2,45},{5.7,55},{6.2,65},{6.7,65},{7.1,80},{7.6,80},{8.1,85},{8.6,90},{9,90},{9.5,90},{10,100}};

Data2={{1,9},{2,12},{3,16},{4,25},{5,27},{6,29},{7,29},{8,32},{9,34},{10,35},{11,36},{12,36},{13,39},{14,39},{15,40},{16,40},{17,40},{18,41},{19,42},{20,43},{21,43},{22,44},{23,45},{24,45},{25,46},{26,47},{27,47},{28,49},{29,50},{30,50},{31,50},{32,50},{33,51},{34,52},{35,53},{36,54},{37,55},{38,55},{39,55},{40,55},{41,56},{42,59},{43,60},{44,60},{45,60},{46,61},{47,62},{48,62},{49,62},{50,62},{51,62},{52,64},{53,65}};

Data3={{1,12},{2,23},{3,43},{4,64},{5,84},{6,97},{7,109},{8,111},{9,112},{10,114},{11,116},{12,123},{13,126},{14,128},{15,132},{16,141},{17,144}};

Data4={{1,9},{2,15},{3,19},{4,24},{5,28},{6,29},{7,32},{8,36},{9,36},{10,40},{11,41},{12,41},{13,45},{14,47},{15,49},{16,52},{17,52},{18,55},{19,57},{20,58},{21,61},{22,61},{23,64},{24,66},{25,67},{26,69},{27,70},{28,73},{29,74},{30,75},{31,75},{32,78},{33,79},{34,80},{35,82},{36,83},{37,83},{38,84},{39,84},{40,85},{41,85},{42,87},{43,87},{44,87},{45,89},{46,89},{47,91},{48,91},{49,94}};

Data5={{4,2},{8.3,2},{10.3,2},{10.9,3},{13.2,4},{14.8,6},{16.6,7},{31.3,16},{56.4,29},{60.9,31},{70.4,42},{78.9,44},{108.4,55},{130.4,69},{169.9,87},{195.9,99},{220.9,111},{252.3,126},{282.3,132},{295.1,135},{300.1,136}};

Data6={{1,27},{2,43},{3,54},{4,64},{5,75},{6,82},{7,84},{8,89},{9,92},{10,93},{11,97},{12,104},{13,106},{14,111},{15,116},{16,122},{17,122},{18,127},{19,128},{20,129},{21,131},{22,132},{23,134},{24,135},{25,136}};

Data7={{0.625,9},{1.065,13},{1.465,20},{2.145,26},{2.765,31},{3.425,34},{4.155,36},{4.89,41},{5.81,45},{6.524,47},{7.169,51},{7.816,58},{8.176,58},{8.716,63},{9.111,66},{9.791,69},{10.401,72},{11.027,76},{12.014,86},{12.264,89},{12.384,90},{12.934,92},{13.424,96},{14.064,101},{14.324,101},{14.984,103},{15.474,103},{15.994,105},{16.694,108},{17.539,114},{18.369,117},{18.969,118},{19.694,119},{20.594,123},{21.174,126},{21.774,128},{23.454,139},{24.579,148}};

Data8={{9,10},{21,20},{32,30},{36.,40},{43.,50},{45,60},{50,70},{58,80},{63,90},{70,100},{71,110},{77,120},{78,130},{87,140},{91,150},{92,160},{95,170},{98,180},{104,190},{105,200},{116,210},{149,220},{156,230},{250,250}};

The estimated parameters for each of the four models across all eight datasets are listed in Table 4, along with results from goodness-of-fit tests. Notably, when comparing the proposed model with the other three, it consistently ranks among the top two regarding goodness of fit. However, it is worth mentioning that Kapur's imperfect debugging SRGM could perform better with datasets 1 and 2, particularly when assessing criteria such as MSE and Rsq. It attributed to Kapur's model not adapting

effectively to S-type testing data. In contrast, it exhibits a good fit with the other datasets. This distinction becomes evident when examining the figures in datasets 1 and 8 in Fig. 4, where Kapur's model struggles to accommodate both S-type and concave-type testing data simultaneously, especially when compared to the proposed model.

On the other hand, the imperfect debugging SRGM of Wang et al. demonstrates favorable applicability across most of the eight datasets, resulting in low MSE and Rsq values exceeding 95%. It is important to note that models with more parameters offer improved goodness of fit in terms of AIC. However, increasing the number of parameters should be done judiciously to avoid overfitting. In the case of Wang's model, overfitting becomes evident, where the model captures intricate details and noise in the training data to a degree that adversely affects its performance on new data. This phenomenon is particularly notable in datasets 1, 5, and 7, as seen in Fig. 5, especially during the early testing stages. Multiple curves turning points attempt to match the distribution of the testing data, but this overfitting can hinder the model's generalization capabilities.

In summary, apart from Wang's imperfect debugging model, the proposed model consistently emerges as the most suitable choice. It is evident from the MSE and Rsq values presented in Table 5 that this study demonstrates superior suitability, except for dataset 2. Figs. 3-6 illustrate each imperfect debugging model's conventional confidence intervals (represented by red dashed lines). Figure 6 showcases the introduction of a new concept (blue dashed lines) in confidence intervals, in addition to the traditional approach.

Table 4
Estimated Values of Models' Parameters for Different Data Set

Data Set	Pham et al. (1999)	Kapur et al. (2008)	Wang et al. (2015)	Proposed model
[1]	*a=102.07* *b=0.615581* *α=0.000000* *β=27.580800*	*a=103.94* *b=0.028407* *α=2.107800* *p=2.411560*	*a=101.80* *b=1.015990* *α=0.440787* *d=2.391560* *C=2.421650*	*a=102.04* *b=0.021527* *γ=0.594335* *ε=0.000000*
[2]	*a=60.20* *b=0.071598* *α=0.000010* *β=0.000000*	*a=60.21* *b=0.741666* *α=0.000002* *p=0.097826*	*a=69.60* *b=1.292100* *α=0.051785* *d=0.981500* *C=8.950500*	*a=63.50* *b =0.062347* *γ=0.000000* *ε=0.000000*
[3]	*a=146.12* *b=0.211463* *α=0.000000* *β=0.692870*	*a=141.73* *b=0.702811* *α=0.080910* *p=0.217925*	*a=150.70* *b=1.180560* *α=0.223392* *d=1.399540* *C=5.571260*	*a=145.20* *b=0.216806* *γ=0.002745* *ε=0.093055*
[4]	*a=101.67* *b=0.045435* *α=0.000000* *β=0.000000*	*a=101.68* *b=0.768848* *α=0.000004* *p=0.059007*	*a=99.99* *b=1.078420* *α=0.045450* *d=1.472360* *C=14.341600*	*a=100.75* *b=0.046262* *γ=0.000000* *ε=0.000000*
[5]	*a=130.13* *b=0.014867* *α=0.000007* *β=5.137940*	*a=136.47* *b=0.081202* *α=0.757768* *p=0.048613*	*a=150.80* *b=0.167197* *α=0.006806* *d=2.112940* *C=7.755370*	*a=146.33* *b=0.009877* *γ=0.002630* *ε=0.007029*
[6]	*a=136.89* *b=0.135467* *α=0.000009* *β=0.000000*	*a=136.32* *b=0.170887* *α=0.000029* *p=0.804763*	*a=152.47* *b=2.094400* *α=0.095743* *d=1.107060* *C=24.739100*	*a=135.98* *b=0.138236* *γ=0.000000* *ε=0.000000*
[7]	*a=212.32* *b=0.045812* *α=0.000035* *β=0.111308*	*a=201.76* *b=0.612217* *α=0.158631* *p=0.070752*	*a=150.38* *b=2.084310* *α=0.074674* *d=1.913120* *C=20.043100*	*a=234.39* *b=0.037598* *γ=0.000000* *ε=0.000000*
[8]	*a=245.52* *b=0.039804* *α=0.000000* *β=18.390600*	*a=245.76* *b=0.043850* *α=0.379483* *p=0.187553*	*a=246.71* *b=0.070716* *α=0.027353* *d=2.560130* *C=4.590530*	*a=250.11* *b=0.034449* *γ=0.002002* *ε=0.032082*

Table 5
Comparisons of Different Fitting Criteria

MSE	**Data1**	**Data2**	**Data3**	**Data4**	**Data5**	**Data6**	**Data7**	**Data8**
Pham et al. (1999)	7.96	15.79	44.92	7.40	17.35	32.02	13.81	44.66
Kapur et al. (2008)	42.16	15.84	51.86	7.40	9.05	31.94	13.71	463.39
Wang et al. (2015)	6.67	5.44	19.22	2.93	20.48	7.87	14.11	41.78
Proposed model	7.96	17.72	44.19	7.36	8.78	31.92	13.66	42.46
R-Squared	**Data1**	**Data2**	**Data3**	**Data4**	**Data5**	**Data6**	**Data7**	**Data8**
Pham et al. (1999)	99.31%	91.22%	97.22%	98.64%	99.33%	96.63%	99.03%	99.13%
Kapur et al. (2008)	96.37%	91.20%	96.79%	98.64%	99.65%	96.64%	99.04%	90.92%
Wang et al. (2015)	99.43%	96.98%	98.81%	99.46%	99.21%	99.17%	99.01%	99.18%
Proposed model	99.31%	90.15%	97.27%	98.65%	99.66%	96.64%	99.04%	99.17%
AIC	**Data1**	**Data2**	**Data3**	**Data4**	**Data5**	**Data6**	**Data7**	**Data8**
Pham et al. (1999)	50.55	153.25	71.65	105.08	66.90	93.64	106.75	98.16
Kapur et al. (2008)	85.55	153.39	74.09	105.09	53.24	93.58	106.46	154.30
Wang et al. (2015)	48.82	98.76	59.22	61.67	72.39	60.54	109.58	98.55
Proposed model	50.55	159.35	71.37	104.79	52.59	93.56	106.34	96.95

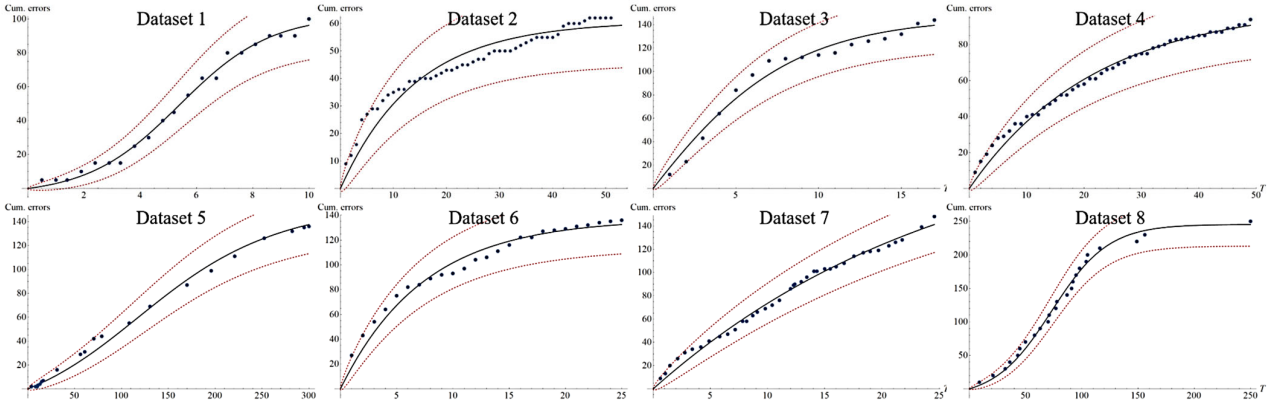

Fig. 3. The Fitting Results for the Imperfect Debugging SRGM proposed by Pham et al. (1999)

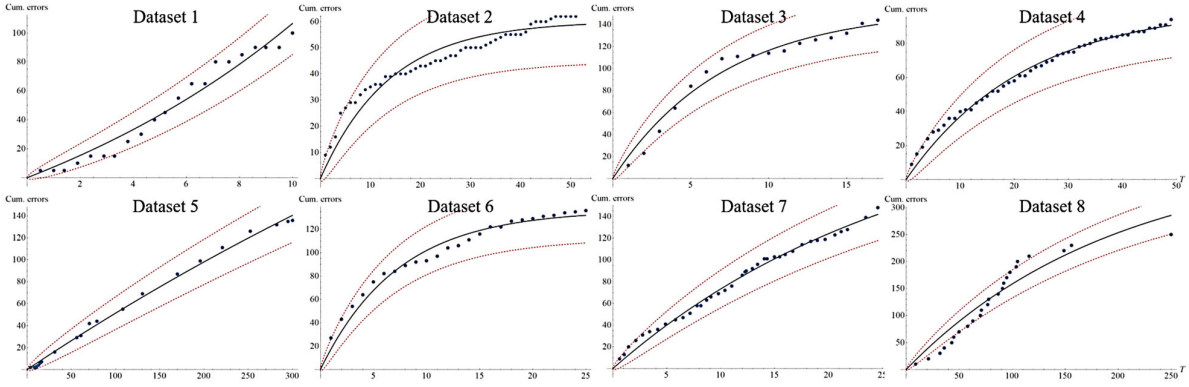

Fig. 4. The Fitting Results for the Imperfect Debugging SRGM proposed by Kapur et al. (2008)

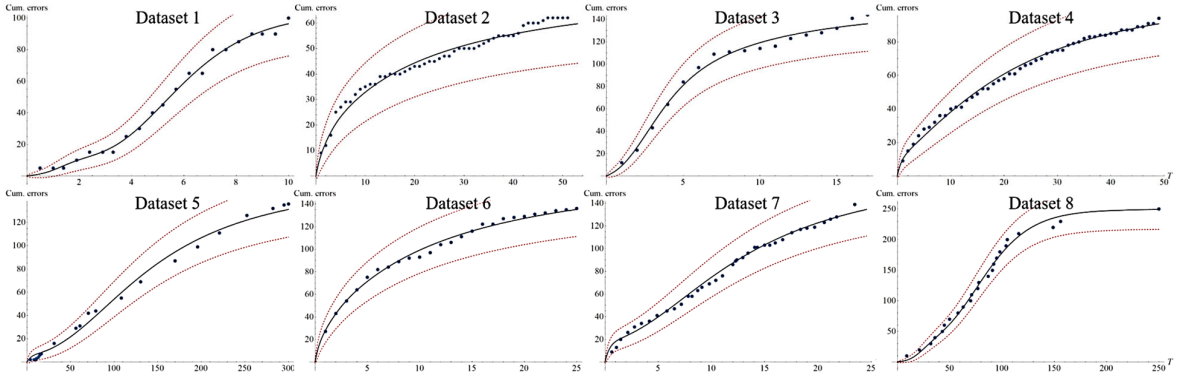

Fig. 5. The Fitting Results for the Imperfect Debugging SRGM proposed by Wang et al. (2015)

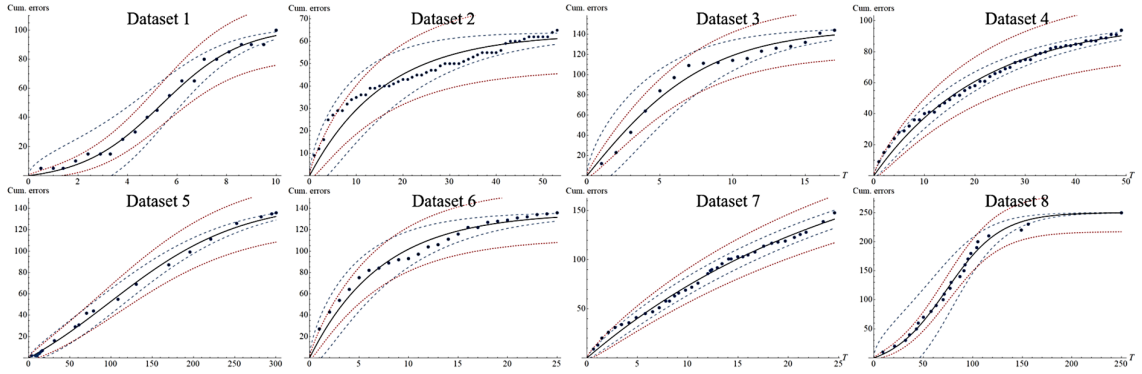

Fig. 6. Comparisons between the Traditional and the Proposed Confidence Intervals for the Proposed Model.

5. Decision for Optimal Software Release

Software developers must carefully plan the duration of software testing before launching a new system online. Inadequate time allocated for testing can adversely affect the software's stability, potentially resulting in increased costs or user dissatisfaction. Conversely, excessive testing time escalates testing costs and leads to opportunity costs due to delayed software release. Therefore, this research model draws inspiration from Zhang and Pham's (1998) decision-making model for software testing costs and devises a novel decision-making model that accounts for confidence intervals. This model incorporates a conservative perspective, empowering decision-makers to approach average and worst-case scenarios with a nuanced mindset.

Six distinct costs are associated with evaluating a software release. The first cost, denoted as C_0, covers the setup expenses of the software testing project, encompassing necessary fixed investments. C_1 pertains to routine costs, including office rent, utility charges, and miscellaneous expenses. C_2 and C_3 represent the expected costs of rectifying a software error per unit of time during the testing and warranty phases, respectively. The estimated time needed for error resolution is u_y during testing and u_w during the warranty period. C_4 signifies the expected losses from a software error, and software experts can assess its impact on subsequent operations. C_5 accounts for the expected loss of business opportunities due to delayed software release, quantified through a power law function, reflecting opportunity loss over time.

Furthermore, while Eq. (47) is employed to pinpoint the optimal software release time in an average-case scenario, Eq. (48) proves valuable for conservative decision-making, addressing potential inefficiencies in future testing work. Consequently, software project managers should consider the lower boundaries, $m_{LB}^{CR}(T)$ and $R_{LB}^{CR}(x|T)$, within a specific critical region (CR). It allows project managers to establish confidence levels for their requirements and determine the optimal release time by balancing software quality and testing cost.

$$min\ C(T) = C_0 + C_1T + C_2m(T)u_y + C_3(m(T+T_w) - m(T))u_w + C_4(1 - R(x|T)) + C_5(v_1 + T)^{v_2}$$
$$subject\ to: R(x|T+T_w) \geq R_{req} \tag{47}$$

$$min\ C_{LB}^{CR}(T) = C_0 + C_1T + C_2m_{LB}^{CR}(T)u_y + C_3\left(m_{LB}^{CR}(T+T_w) - m_{LB}^{CR}(T)\right)u_w + C_4(1 - R_{LB}^{CR}(x|T)) + C_5(v_1 + T)^{v_2}$$
$$subject\ to: R_{LB}^{CR}(x|T+T_w) \geq R_{req} \tag{48}$$

6. Application

Suppose an app company has just completed the development of an industrial management system for a mechanical manufacturer. The company's manager is planning an appropriate testing timeline for the software release. However, they have three distinct candidate test plans for the industrial management system: low-input (P1), medium-input (P2), and high-input (P3). Each testing staff member works 8 hours a day and 22 days a month, with rotating responsibilities. Additionally, the company provides a one-month warranty period, and the minimum software reliability must remain above 95% after this period. The costs of rectifying errors during testing differ from those in the warranty period, and the skill levels of the testing staff vary. Details regarding these test plans and their parameter estimations are presented in Table 6.

Table 6
Related Software Testing Costs and Parameters Estimation of the Three Candidate Test Plans

Plan 1 (P1)	Plan 2 (P2)	Plan 3 (P3)
Low-input human resources	Medium-input human resources	High-input human resources
$\hat{a}$=650		
$\hat{b}$=0.5, $\hat{\gamma}$=0.25 $\hat{\varepsilon}$= 0.1, $\hat{\sigma}$=0.086	$\hat{b}$=0.6, $\hat{\gamma}$=0.35 $\hat{\varepsilon}$= 0.1, $\hat{\sigma}$=0.086	$\hat{b}$=0.7, $\hat{\gamma}$=0.35 $\hat{\varepsilon}$= 0.05, $\hat{\sigma}$=0.086
C_0= \$2,000, C_2= \$19,000, C_3= \$29,000	C_0= \$2,000, C_2= \$20,000, C_3= \$30,000	C_0= \$3,000, C_2= \$25,000, C_3= \$32,000
C_1 = \$5000, C_4= \$120,000, C_5= \$2,800, v_1= 1, v_2= 1.3, χ =0.5 hour, R_{req}= 95%, u_y= 1 hour, u_w=1 hour, $C.R.$ = 5% (Confidence Level=95%) T_w=1 month, working hours/per month=22days*8hours=176hours		

These three plans represent varying talent dispatch options, with higher investment in human resources leading to more efficient testing work and higher staffing costs. Consequently, the critical decision is to select the most suitable staffing plan. Based on the data in Table 6, we compare the total cost against testing time for these three plans in average cases shown in Fig. 7. Among these three plans, it is evident that test plan 2 (P2) is the most appropriate choice. Its expected optimal testing cost is the lowest, amounting to \$125,441 (T^*=3.45 months). While the initial software reliability of P2 stands at 88.92%, it reaches a reliability of 95.38% ($R(x|T^*+T_w)$) after the warranty period, satisfying the crucial 95% software reliability requirement. In contrast, test plan P3, with a minimal testing cost close to P2 (\$125,732, $C(T^*)$), only achieves software reliability of 90.51% after the warranty period, falling short of the essential 95% reliability requirement. To reach the desired reliability level, P3 would need to extend its testing time to over 4.2 months, resulting in a total testing period of 5.2 months (Fig. 8). It lengthens the testing period by 0.8 months compared to P2 and incurs additional costs to meet the minimum 95% software reliability. Consequently, based on the above analysis, test plan P2 emerges as the most rational and cost-effective choice among the three options. Thus, the company manager should opt for test plan P2 for decision-making.

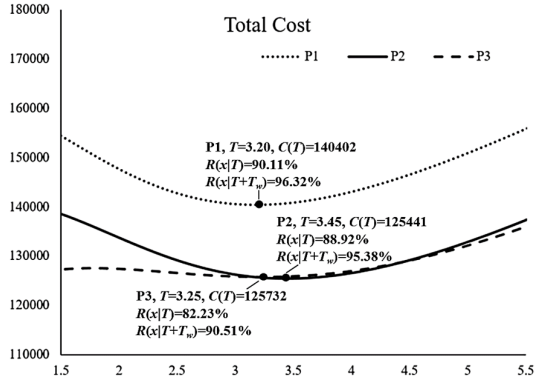

Fig. 7. Comparison of Testing Costs among Test plans P1, P2 and P3

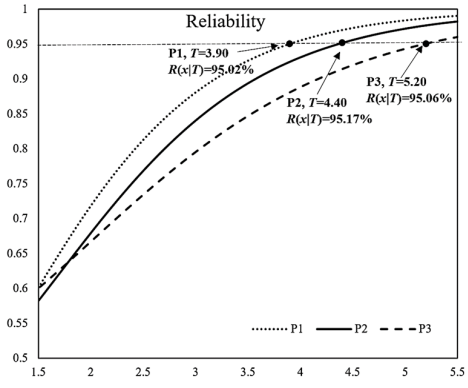

Fig. 8. Comparisons of Reliability among Test plans P1, P2 and P3.

With test plan P2 selected, the company manager is concerned that initial estimates may have been overly optimistic. If we ascertain the best software release time in a worst-case scenario, conservative decisions based on worse-case scenarios are illustrated in Fig. 9 and Fig. 10. In this scenario, the software release time should be 3.7 months to meet the minimum software reliability requirement of 95% ($R_{LB}^{CR}(x|T+T_w)$=95.05%). At this point, the total cost $C_{LB}^{CR}(T^*)$ is estimated at \$124,051. If the manager desires a conservative approach, extending the original software release time by 0.3 months with a 95% confidence level is recommended. Detailed information regarding the upper and lower boundaries of reliability and test costs for test plan P2 is presented in Table 7.

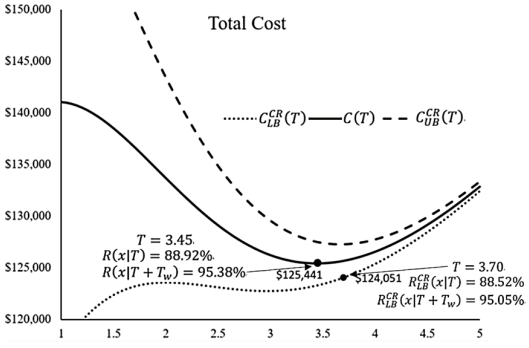

Fig. 9. The Testing Cost of Average Case, Upper Bound and Lower Bound for Test plan 2

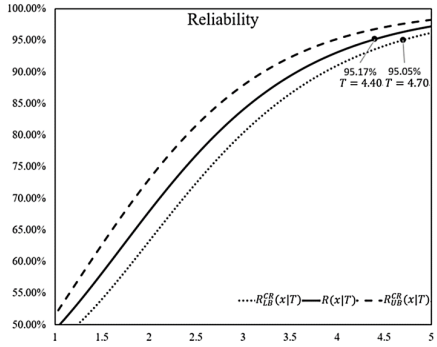

Fig. 10. The Reliability of Average Case, Upper Bound and Lower Bound for Test plan 2

Table 7
The Upper and Lower Boundaries of the Reliability and the Testing Cost for Test plan P2.

T	$R_{LB}^{CR}(x\|T)$	$R(x\|T)$	$R_{UB}^{CR}(x\|T)$	T	$C_{LB}^{CR}(T)$	$C(T)$	$C_{UB}^{CR}(T)$
3	80.35%	84.03%	87.88%	3	$122,748	$126,244	$129,537
3.05	81.05%	84.65%	88.40%	3.05	$122,761	$126,070	$129,189
3.1	81.74%	85.25%	88.90%	3.1	$122,785	$125,918	$128,871
3.15	82.40%	85.82%	89.39%	3.15	$122,820	$125,787	$128,584
3.2	83.05%	86.38%	89.86%	3.2	$122,868	$125,678	$128,327
3.25	83.67%	86.93%	90.30%	3.25	$122,928	$125,589	$128,099
3.3	84.28%	87.45%	90.74%	3.3	$123,001	$125,522	$127,899
3.35	84.87%	87.96%	91.15%	3.35	$123,087	$125,475	$127,728
3.4	85.45%	88.45%	91.55%	3.4	$123,186	$125,448	$127,584
3.45	86.00%	88.92%	91.93%	**3.45**	$123,298	**$125,441**	$127,466
3.5	86.54%	89.37%	92.30%	3.5	$123,422	$125,454	$127,374
3.55	87.06%	89.81%	92.65%	3.55	$123,560	$125,486	$127,308
3.6	87.56%	90.23%	92.99%	3.6	$123,711	$125,537	$127,265
3.65	88.05%	90.64%	93.31%	3.65	$123,875	$125,606	$127,246
3.7	88.52%	91.03%	93.62%	**3.7**	**$124,051**	$125,693	$127,250
3.75	88.98%	91.41%	93.91%	3.75	$124,240	$125,798	$127,277
3.8	89.42%	91.78%	94.20%	3.8	$124,442	$125,920	$127,324
3.85	89.84%	92.13%	94.47%	3.85	$124,655	$126,059	$127,393
3.9	90.25%	92.46%	94.73%	3.9	$124,881	$126,214	$127,482
3.95	90.64%	92.78%	94.98%	3.95	$125,119	$126,385	$127,590
4	91.02%	93.09%	95.22%	4	$125,369	$126,571	$127,717
4.05	91.39%	93.39%	95.44%	4.05	$125,630	$126,772	$127,862
4.1	91.74%	93.68%	95.66%	4.1	$125,902	$126,988	$128,024
4.15	92.08%	93.95%	95.87%	4.15	$126,186	$127,218	$128,204
4.2	92.40%	94.22%	96.06%	4.2	$126,479	$127,461	$128,400
4.25	92.72%	94.47%	96.25%	4.25	$126,784	$127,717	$128,611
4.3	93.02%	94.71%	96.43%	4.3	$127,098	$127,987	$128,838
4.35	93.31%	94.94%	96.61%	4.35	$127,423	$128,268	$129,079
4.4	93.59%	**95.17%**	96.77%	4.4	$127,757	$128,561	$129,334
4.45	93.86%	**95.38%**	96.93%	4.45	$128,101	$128,866	$129,603
4.5	94.11%	95.58%	97.08%	4.5	$128,453	$129,183	$129,884
4.55	94.36%	95.78%	97.22%	4.55	$128,815	$129,509	$130,179
4.6	94.60%	95.97%	97.35%	4.6	$129,185	$129,847	$130,485
4.65	94.83%	96.15%	97.48%	4.65	$129,563	$130,194	$130,803
4.7	**95.05%**	96.32%	97.61%	4.7	$129,949	$130,551	$131,132
4.75	95.26%	96.48%	97.73%	4.75	$130,344	$130,917	$131,471
4.8	95.46%	96.64%	97.84%	4.8	$130,745	$131,292	$131,821
4.85	95.65%	96.79%	97.94%	4.85	$131,154	$131,676	$132,181
4.9	95.84%	96.93%	98.05%	4.9	$131,571	$132,068	$132,551
4.95	96.01%	97.07%	98.14%	4.95	$131,994	$132,468	$132,929
5	96.18%	97.20%	98.24%	5	$132,423	$132,876	$133,317

Modifying the values for the critical region (CR) introduces decision variations. This study shows that CR value adjustments influence software reliability and cost estimations. An increase in the CR value leads to overestimated software reliability and costs, as observed in Fig. 11. It indicates that decision-makers can tailor their worst-case scenarios to suit their conservative perspectives.

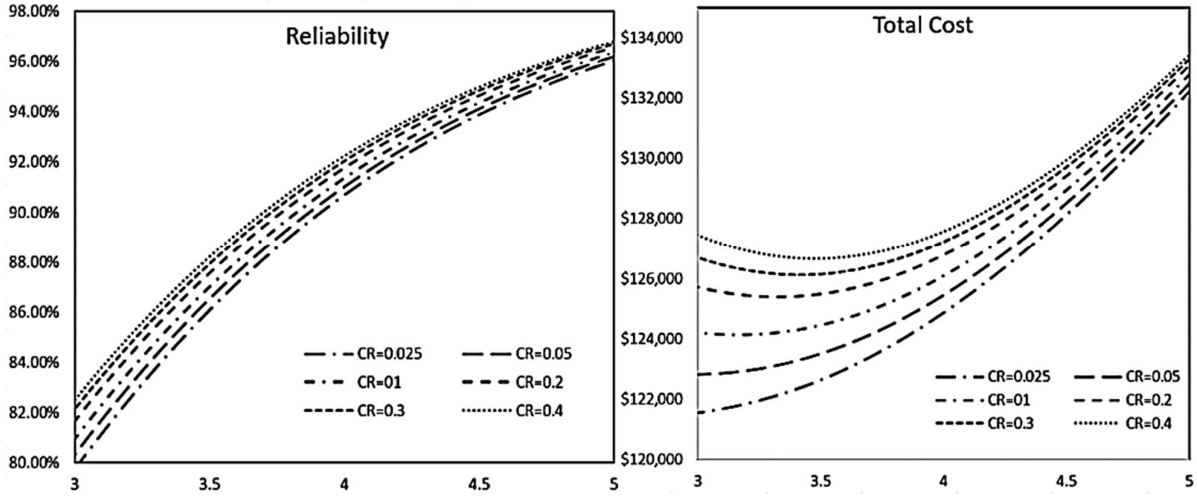

Fig. 11. The Lower Bounds of Reliability and Cost in Various Critical Regions

Understanding the variations in model parameters affecting software reliability and cost estimation is crucial, as numerical changes substantially impact future software testing decisions. As depicted on the left side of Fig. 12, the confidence interval for the detection rate expands with an increase in the standard deviation of the detection rate, subsequently broadening the range and uncertainty of the estimate for $m(t)$ and $R(t)$. Regarding the lower bound of software reliability, a wider confidence interval for $R(t)$ decreases the value of $R_{LB}^{CR}(x|T+T_w)$. Hence, an increase in the parameter σ leads to a more conservative software reliability estimation. Additionally, a cost-benefit sensitivity analysis on the parameters b, γ and ε highlights the role of parameter values in cost implications. While an increase in b and γ proves beneficial for reducing related costs, it also necessitates more training or senior personnel, incurring additional costs. Likewise, lowering the parameter ε in the debugging process enhances testing efficiency. Investment in online case tools and effective interactions among testing staff can prevent recurring mistakes. Therefore, software managers must carefully weigh the trade-off between cost and efficiency when making decisions.

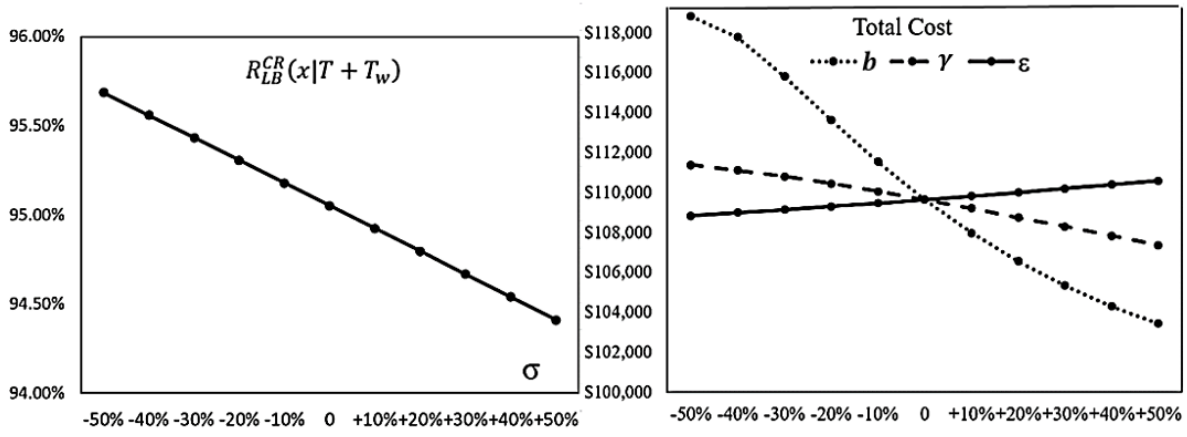

Fig. 12. The Variability of Reliability and Cost when Parameters Change

7. Conclusion

This study introduces an innovative debugging software reliability growth model that acknowledges the influence of learning and negligence factors. Our approach offers a more intuitive means of estimating parameters, facilitating expert evaluation. The negligence factor is employed to elucidate the concept of imperfect debugging. We contend that new errors surface only after previously identified issues are addressed in the source code. These new errors do not typically emerge during error detection but may arise during removal and correction. Unlike prior studies, which often oversimplified the assumption that new software errors only increase with testing time, we hypothesize that the growth rate of new errors will decrease due to the reduction in previously unseen error patterns. Additionally, we enhance the traditional model by broadening the confidence interval. This modification equips decision-makers with moderately conservative decision information, thus enhancing the quality of their choices. Our analytical findings empower decision-makers to grasp the pinnacle of software testing efficiency and understand the variations in detection rates within upper and lower bounds throughout the software testing process. This study offers a practical decision-making framework for managers to select the optimal alternative while considering the imperative of software reliability. It is noteworthy that adjusting the critical region (CR) values may lead to varying decisions, as higher CR values may overestimate software reliability and costs. Moreover, an increased standard deviation (σ) in the detection rate yields a more conservative estimate of software reliability. Regarding the impact of model parameters b, γ and ε on costs, while augmenting b and γ can enhance debugging efficiency, it also demands more on-the-job training or senior expertise. This trade-off challenges decision-makers before arriving at a final verdict. Furthermore, reducing the value of parameter ε hinges on minimizing errors during the correction process, which can be achieved through on-the-job training and online case tools, which can prevent the repetition of the same mistakes.

In conclusion, this study opens up two promising avenues for future research. First, we propose addressing the time delay issue in developing Software Reliability Growth Models (SRGMs). It is a critical factor often neglected in previous studies, as the time taken to correct software errors can significantly impact testing schedules and bias software reliability estimates. Future studies will integrate this time delay factor into the model for a more realistic representation. Second, we suggest expanding our research into a multi-version software testing model. The emergence of issues in multi-version software testing, often related to releasing new software versions for feature updates, makes this a compelling direction for future research, expanding the scope of the investigation.

References

Akaike, H. (1973) Information theory and an extension of the maximum likelihood principle. *Second International Symposium on Information Theory*, 267-281.

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. *International Journal of Quality & Reliability Management*, 27(1), 89-110.

Aktekin, T., & Caglar, T. (2013). Imperfect debugging in software reliability: A Bayesian approach. *European Journal of Operational Research, 227*, 112-121.

Awad, M. (2016) Economic allocation of reliability growth testing using Weibull distributions. *Reliability Engineering and System Safety, 152*, 273–280.

Cao, P., Yang, K., & Liu, K. (2020). Optimal selection and release problem in software testing process: a continuous time stochastic control approach. *European Journal of Operational Research*, *285*(1), 211-222. https://doi.org/10.1016/j.ejor.2019.01.075.

Chiu, K.C., Huang, Y.S., & Huang, I.C. (2019) A Study of Software Reliability Growth with Imperfect Debugging for Time-Dependent Potential Errors. *International Journal of Industrial Engineering, 26*(3), 376-393.

Chiu, K.C., Huang, Y.S. and Lee, T.Z. (2008) A Study of Software Reliability Growth from the Perspective of Learning Effects. *Reliability Engineering and System Safety, 93*(10), 1410-1421.

Cortellessa, V., Mirandola, R., & Potena, P. (2015) Managing the evolution of a software architecture at minimal cost under performance and reliability constraints. *Science of Computer Programming Volume, 98*(41), 439-463.

Fang, C. C., & Yeh, C. W. (2016) Effective confidence interval estimation of fault-detection process of software reliability growth models. *International Journal of Systems Science, 47*(12), 2878-2892.

Goel, A.L., & Okumoto K. (1979) Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability, 28*(3), 206-211.

Hsu, C.J., Huang, C.Y., & Chang, J.R. (2011) Enhancing Software Reliability Modeling and Prediction through the Introduction of Time-Variable Fault Reduction Factor. *Applied Mathematical Modelling, 35*(1), 506-521.

Huang, C.Y. (2005) Performance analysis of software reliability growth models with testing-effort and change-point. *Journal of Systems and Software, 76*(2), 181-194.

Huang, C.Y., Kuo, S.Y., & Michael, R.L. (2007) An assessment of testing-effort dependent software reliability growth models. *IEEE Transactions on Reliability, 56*(2), 198–211.

Huang, Y.S., Chiu, K. C., & Chen, W.M. (2022) A software reliability growth model for imperfect debugging. *Journal of Systems and Software, 188*, 111267.

Huang, Y.S., Fang, C.C., Chou, C.C., & Tseng, T.L. (2023) A Study on Optimal Release Schedule for Multiversion Software. *INFORMS Journal on Computing*, DOI: 10.1287/ijoc.2021.0141.

Jin, C., & Jin, S.W. (2016) Parameter optimization of software reliability growth model with S-shaped testing-effort function using improved swarm intelligent optimization. *Applied Soft Computing, 40*, 283-291.

Kapur, P.K., Gupta, D., Gupta, A., & Jha, P.C. (2008) Effect of introduction of faults and imperfect debugging on release time. *Ratio Mathematica, 18*, 62–90.

Kapur, P.K., Pham, H., Anand, S., & Yadav, K. (2011) A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. *IEEE Transactions on Reliability, 60*(1), 331-340.

Ke, S.Z., & Huang, C.Y. (2020) Software reliability prediction and management: A multiple change-point model approach. *Quality and Reliability Engineering International, 36*(5), 1678-1707.

Kooli, M., Kaddachi, F., Natale, G.D., Bosio, A., Benoit, P., & Torres, L. (2017) Computing reliability: On the differences between software testing and software fault injection techniques. *Microprocessors and Microsystems, 50*, 102-112.

Cai, K.Y., Hu, D.B., C.G., Bai, H.H., & Jing, T. (2008) Does Software Reliability Growth Behavior Follow a Non-homogeneous Poisson Process*? Information and Software Technology, 50*(12), 1232-1247.

Levitin, G., Xing, L., & Xiang, Y. (2020) Cost minimization of real-time mission for software systems with rejuvenation. *Reliability Engineering and System Safety, 193*, https://doi.org/10.1016/j.ress.2019.106593.

Li, Q., & Pham, H. (2017) NHPP software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage. *Applied Mathematical Modelling, 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.

Pachauri, B., Dhar, J., & Kumar, A. (2015) Incorporating Inflection S-Shaped Fault Reduction Factor to Enhance Software Reliability Growth. *Applied Mathematical Modelling, 39*(5), 1463-1469.

Peng, R., Li, Y.F., Zhang, W.J., & Hu, Q.P. (2014) Testing Effort Dependent Software Reliability Model for Imperfect Debugging Process Considering Both Detection and Correction. *Reliability Engineering and System Safety, 126*, 37-43.

Pham, H., Nordmann, L., & Zhang, X. (1999) A general Imperfect software debugging model with S-shaped fault detection rate. *IEEE Transactions on Reliability, 48*(2), 169–175.

Ramsamy, S., Govindasamy, G., & Kapur, P.K. (2012) A Software Reliability Growth Model for Estimating Debugging and the Learning Indices. *International Journal of Perform ability Engineering, 8*(5), 539- 549.

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. *Quality and Reliability Engineering International*, *35*(7), 2358-2370. https://doi.org/10.1002/qre.2516.

Shyur, H.J. (2003) A stochastic software reliability model with imperfect-debugging and change-point. *Journal of Systems and Software, 66*(2), 135-141.

Shrivastava, A. K., & Kapur, P. K. (2021). Change-points-based software scheduling. *Quality and Reliability Engineering*

International, *37*(8), 3282-3296.

Singpurwalla, N.D., & Wilson, S.P. (1999) Statistical Methods in Software Engineering. Springer-Verlang, New York, Inc.

Tian, Q., Fang, C. C., & Yeh, C. W. (2022). Software release assessment under multiple alternatives with consideration of debuggers' learning rate and imperfect debugging environment. *Mathematics*, *10*(10), 1744.

Wang, J., & Wu. Z. (2016) Study of the nonlinear imperfect software debugging model. *Reliability Engineering and System Safety, 153*, 180-192.

Wang, J., Wu, Z., Shu, Y., & Zhang, Z. (2015) An imperfect software debugging model considering log-logistic distribution fault content function. *The Journal of Systems and Software, 100*, 167–181.

Wang, L., Hu, Q., & Liu, J. (2016) Software Reliability Growth Modeling and Analysis with Dual Fault Detection and Correction Processes. *IIE Transactions, 48*(4), 359-370.

Yamada, S., & Osaki, S. (1985) Software reliability growth modeling: Models and applications. *IEEE Transactions on Software Engineering, 11*(12), 1431-1437.

Yang, J., Liu, Y., Xie, M., & Zhao, M. (2016) Modeling and Analysis of Reliability of Multi-Release Open Source Software Incorporating Both Fault Detection and Correction Processes. *Journal of Systems and Software, 115*, 102-110.

Zhang, X., & Pham, H. (1998) A software cost model with error removal times and risk costs. *International Journal of Systems Science, 29*(4), 435-442.

Zhang, X., & Pham, H. (2006) Software field failure rate prediction before software deployment. *The Journal of Systems and Software, 79*, 291-300.

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. *Reliability Engineering and System Safety, 175*, 265–282.

Zhu, M., & Pham, H. (2018) A multi-release software reliability modeling for open source software incorporating dependent fault detection process. *Annals of Operations Research, 269*, 773–790.