IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 23 December 2024, accepted 12 January 2025, date of publication 14 January 2025, date of current version 22 January 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3529775

RESEARCH ARTICLE

Integrating Human Learning Factors and Bayesian Analysis Into Software Reliability Growth Models for Optimal Release Strategies

CHIH-CHIANG FANG[1,2], **LIPING MA**[1], **AND WENFENG KUO**[1]
[1]School of Information Engineering, Shanghai Zhongqiao Vocational and Technical University, Shanghai 201514, China
[2]School of Computer Science and Software, Zhaoqing University, Zhaoqing 526061, China

Corresponding authors: Liping Ma (malp@shzq.edu.cn) and Wenfeng Kuo (kuowf@shzq.edu.cn)

This work was supported in part by Guangdong Basic and Applied Basic Research Foundation; and in part by Guangdong Soft Science Foundation, China, under Grant 2024A0505050040.

ABSTRACT This study presents a Software Reliability Growth Model (SRGM) that incorporates imperfect debugging and employs Bayesian analysis to optimize the timing of software releases. The primary objective is to reduce software testing costs while enhancing the model's practical applicability. A significant limitation of traditional estimation techniques, such as MLE and LSE, is their challenge in accurately estimating model parameters when historical data is limited. To overcome this issue, the proposed Bayesian approach utilizes prior knowledge from domain experts and integrates available software testing data to predict both the software's reliability and associated costs. This method facilitates both prior and posterior analyses, making it effective even in scenarios with limited data. The model also considers the efficiency of the debugging process, which can be influenced by factors such as the testing team's learning curve and human error. By integrating these human elements and the intrinsic characteristics of the debugging process, the model becomes more comprehensive and realistic. This results in parameter estimates that more accurately represent real-world scenarios, making the model more intuitive for experts to apply. Additionally, the study incorporates numerical examples and sensitivity analyses that provide essential insights for management. These examples offer strategic guidance for software release decisions, assisting stakeholders in balancing the trade-offs between testing costs, reliability, and release timing. To further enhance decision-making, a computerized application system is proposed to help determine the optimal software release point. This tool streamlines the process, ensuring a more efficient approach to addressing this critical challenge in software development.

INDEX TERMS Bayesian analysis, imperfect debugging, NHPP, software reliability growth model.

I. INTRODUCTION

Software reliability is essential in the software development lifecycle, as failures in computer systems can lead to substantial financial losses or even catastrophic outcomes. This is particularly relevant for technological applications that directly affect system safety. Therefore, ensuring software reliability poses a significant challenge in these areas. In addition, effective management of software testing and debugging costs necessitates a comprehensive understanding of the relationship between software reliability and testing expenses over time. Within development teams, software reliability is a critical factor that influences decision-making processes. Historically, research in this area has operated under the assumption that debugging can be flawless, meaning that errors are completely resolved as soon as they are identified. However, in practice, this is often not the case. Debugging efforts may not fully eliminate errors, and new issues can arise during the error correction process, highlighting the challenge of imperfect debugging, which has garnered increasing attention in recent years. As debugging teams consistently address software defects, they gradually

The associate editor coordinating the review of this manuscript and approving it for publication was Diego Bellan.

enhance their skills, which enables them to identify and resolve issues more quickly and effectively. Simultaneously, many software reliability growth models (SRGMs) utilize the non-homogeneous Poisson process (NHPP) as a foundational framework for analyzing statistical patterns. This approach has gained widespread recognition as an effective method for capturing and predicting reliability improvements in software systems over time [1], [3], [12], [20].

In most existing research, software reliability models are predicated on the assumption of perfect debugging, wherein errors are completely eliminated once detected. However, this assumption is overly simplistic and fails to reflect real-world conditions. In practice, software testers and debuggers may inadvertently introduce new errors while rectifying previous ones. Consequently, recent studies have shifted their focus toward models that incorporate imperfect debugging within SRGMs. For example, Pham et al. [1] introduced a comprehensive model of imperfect debugging that incorporates an S-shaped error detection rate. This model highlights how the testing process evolves over time, with efficiency improving dynamically due to a learning effect. This learning phenomenon is reflected in the varying rate at which faults are detected. Singpurwalla and Willson [2] introduced models that elucidate the uncertainty surrounding the software failure process, noting that the parameters involved are unobservable and should be modeled using prior distributions that vary among individuals. Zhang and Pham [3] developed a methodology for predicting field failure rates by analyzing both system test data and field data using both perfect and imperfect SRGMs. Kapur et al. [4] investigated the optimal release timing problem under an imperfect debugging scenario, addressing the distinct effects of perfect and imperfect debugging on total software costs. Peng et al. [5] introduced an imperfect debugging SRGM that incorporates testing effort allocation, utilizing logistic, Weibull, and constant functions to represent the distribution of testing resources over time. Wang et al. [6] expanded upon this by applying a log-logistic function to model the process of fault introduction within an imperfect debugging SRGM. Building on these concepts, Wang and Wu [7] proposed a nonlinear imperfect debugging model based on a NHPP, effectively capturing the nonlinear behavior of fault introduction. Their model was validated through experiments, demonstrating strong performance in terms of both fitting accuracy and predictive capability. Saraf and Iqbal [8] contributed to the field by integrating two types of imperfect debugging and change-point environments into a decision model designed to optimize software release timing. Verma et al. [9] developed a cost model that incorporates imperfect debugging into software release decisions, highlighting the necessity of balancing debugging costs with customer satisfaction, particularly when determining appropriate software warranty periods. Li et al. [10] introduced an SRGM that accounts for testability growth efforts and delays in fault resolution to enhance the model's goodness of fit. Bibyan [11] presented an SRGM based on testing coverage, incorporating concepts such as change points, error generation, and variable fault detection rates to address the irregularities in fault observation and resolution processes. Yeh and Fang [12] investigated the impact of environmental factors on software testing, utilizing Brownian motion and stochastic differential equations to establish confidence intervals for reliability and cost metrics. These contributions have significantly expanded the application of SRGMs by addressing imperfect debugging in various contexts. However, most of the existing research assumes that new software errors inevitably arise over time as a natural aspect of the development process. In contrast, our study shifts the focus to human factors, such as inexperience and negligence among staff. We propose that these human elements, particularly negligence, can be measured and quantified to assess the effectiveness of software debugging. This perspective provides a novel approach to understanding and modeling imperfect debugging processes.

Additionally, most Software Reliability Growth Models (SRGMs) rely on a substantial amount of historical data to estimate model parameters using methods such as Maximum Likelihood Estimation (MLE) or Least Squares Estimation (LSE). However, when sufficient historical data is unavailable, traditional statistical methods become ineffective for parameter estimation. In such cases, Bayesian statistical analysis presents a viable alternative, as it facilitates estimation with limited data and incorporates expert judgment. Recent research has applied Bayesian methods to SRGMs, with Bai et al. [13] noting that Bayesian networks are particularly well-suited for addressing complex, variable factors in software reliability estimation. Pievatolo et al. [14] developed a Bayesian hidden Markov model to represent the imperfect debugging process, utilizing a Gamma distribution as the prior for estimating model parameters. Aktekin and Caglar [15] employed Markov Chain Monte Carlo methods in conjunction with Bayesian techniques to assess error detection rates. Similarly, Zhao et al. [16] proposed a Bayesian-based SRGM designed to address uncertainties in software reliability, taking into account both perfect and imperfect debugging conditions. Wang et al. [17] employed a Bayesian entropy Markov method to predict real-time software reliability, estimating conditional probabilities through Bayesian techniques. Insua et al. [18] developed Bayesian life tests and accelerated life tests to optimize software reliability decisions, incorporating software warranties to propose cost-effective release strategies. Furia et al. [19] advocated for the flexibility and effectiveness of Bayesian techniques in analyzing software engineering data, while also acknowledging the limitations of these methods. Tian et al. [20] developed an imperfect debugging SRGM utilizing Bayesian analysis, with the objective of identifying the optimal software release point. Their model is intended to minimize testing costs while enhancing its practical applicability. Oveisi et al. [21] utilized machine learning and Bayesian inference to improve prediction accuracy in software reliability, demonstrating through their experiments that non-parametric models can surpass classical approaches. Although these studies in the past emphasize the advantages of Bayesian methods for estimating software reliability, they frequently neglect to incorporate testing costs into software release decisions, especially when considering opportunity costs and minimum reliability thresholds. To address this gap, our research enhances the Bayesian framework by integrating practical considerations, thereby facilitating a more balanced optimization of both reliability and cost in software release strategies.

Based on the previous discussion, our study offers several contributions over existing research:

(1) Traditional models often rely on parameters that are difficult for domain experts to interpret, making their evaluation more challenging. In contrast, our proposed model addresses this limitation by incorporating intuitive human factors such as the learning curve, negligence, and the autonomy of debugging teams.

(2) Our model not only provides deeper insights into the phenomenon of imperfect debugging but also simplifies the estimation process by converting it into a straightforward, measurable coefficient.

(3) We propose a software release model that integrates the required levels of software reliability into both preliminary and subsequent analyses. This approach supports the development of tailored testing strategies, enabling more efficient manpower allocation and delivering a more precise assessment of testing costs.

The remainder of this paper is organized as follows: Section II outlines the development of the proposed model, addressing aspects such as parameter estimation, model validation, and comparisons with existing models. In Section III, we present a Bayesian analysis of the model, along with a decision-making framework for determining the optimal timing for software releases. Section IV emphasizes practical applications and includes numerical examples to demonstrate the model's effectiveness. Finally, Section V concludes the paper by summarizing key findings and providing recommendations for future research directions.

II. MODELING SOFTWARE RELIABILITY GROWTH WITH DEBUGGERS' LEARNING AND NEGLIGENCE FACTORS

In recent years, statistical and stochastic methods have gained widespread adoption in reliability engineering for both hardware and software systems. Among these methods, the NHPP has emerged as a particularly effective tool, leading to the development of numerous software reliability growth models. Unlike the Homogeneous Poisson Process (HPP), the NHPP allows the expected number of failures to vary over time, providing greater flexibility for modeling software reliability. To ensure that a system meets quality standards prior to market release, comprehensive software testing and debugging are essential. Software teams or companies can evaluate and develop several viable testing strategies, with decision-makers responsible for selecting the most appropriate approach to optimize the testing process. Consequently, it is essential for managers to understand how to optimize system reliability through efficient resource allocation, balancing system stability with associated costs.

Within the NHPP framework, the software reliability growth process is modeled as a counting process, represented by $\{N(t), t \geq 0\}$. The probability of encountering $N(t)$ failures by time t is given by the following expression:

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

Here, $M(t)$ refers to the mean value function, which indicates the cumulative number of errors detected within the time interval $[0, t]$ in the software or system. The mean value function is connected to the intensity function $\lambda(\cdot)$ through the following relationship:

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

Zhang and Pham [3] introduced a reliability metric that allows software developers to evaluate and monitor the system's reliability and quality throughout the development process. This reliability measure is expressed as:

$$R(x|t) = e^{-[M(t+x)-M(t)]} \quad (3)$$

This metric represents the probability that no errors will be detected within the time interval $[t, t+x]$, where x is the operational time required to meet stability demands in practical applications. It should be noted that increasing the operational time typically decreases the reliability metric. Additionally, as testing time approaches infinity, software reliability will approach 1 (indicating a perfect system), but in reality, pursuing a perfect system is impractical due to the limitations of testing time and cost.

Figure 1 illustrates the research framework of the proposed model. The research framework begin with Model Inference for SRGM, which calculates the number of software errors while considering factors such as debugging efficiency and learning effects. When sufficient historical data is available, Parameter Estimation is performed using LSE to fit and validate the model's accuracy. Conversely, in cases of insufficient historical data, Bayesian Analysis is employed to estimate the parameter ranges. This process utilizes Prior Probability Distributions, based on expert knowledge, and updates them with observed data to derive Posterior Probability Distributions, thereby enhancing accuracy. Regardless of whether the data is sufficient or insufficient, the framework employs a Cost Model for Determining Optimal Software Release Timing. This cost model balances testing costs, reliability requirements, and release schedules, aiding decision-makers in determining the most appropriate time to release the software.

The following sections will provide a detailed introduction and explanation of the proposed model, following the structure of the research framework. Each component, including Model Inference, Parameter Estimation, Bayesian Analysis,

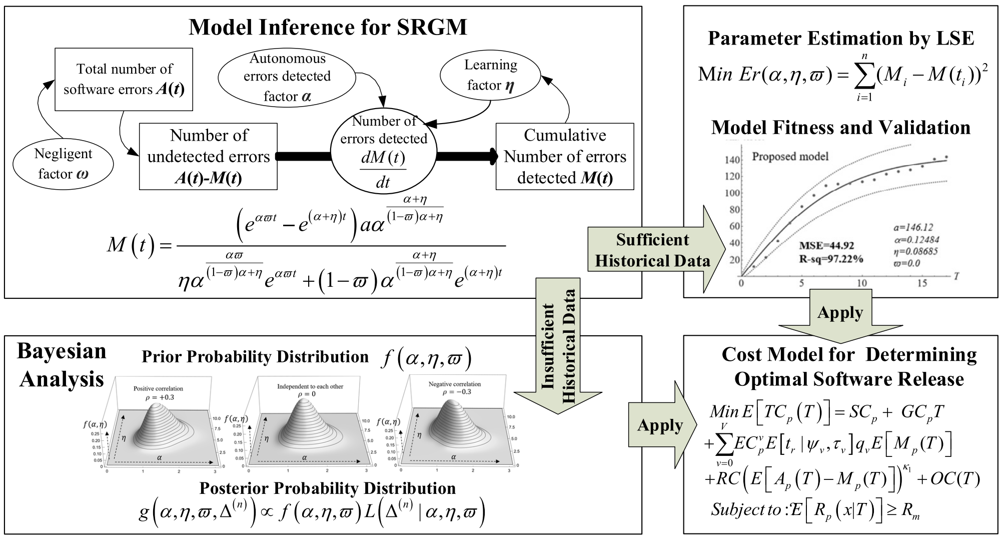
Model Inference for SRGM
Total number of software errors A(t)
Autonomous errors detected factor α
Learning factor η
Negligent factor ϖ
Number of undetected errors A(t)-M(t)
Number of errors detected dM(t)/dt
Cumulative Number of errors detected M(t)
Parameter Estimation by LSE
Model Fitness and Validation
Proposed model
MSE=44.92
R-sq=97.22%
a=146.12
α=0.12484
η=0.08685
ϖ=0.0
Sufficient Historical Data
Insufficient Historical Data
Apply
Apply
Bayesian Analysis
Prior Probability Distribution f(α,η,ϖ)
Positive correlation ρ = +0.3
Independent to each other ρ = 0
Negative correlation ρ = −0.3
Posterior Probability Distribution
Cost Model for Determining Optimal Software Release

FIGURE 1. The research framework for the proposed model.

and the Cost Model for optimal software release timing, will be discussed to clarify the model's methodology and practical applications.

A. BASIC MODEL DEVELOPMENT

The model inference section introduces the fundamental concepts and mathematical reasoning behind the study. The proposed model assumes that three primary factors influence the rate at which errors are detected: the autonomous error-detection factor α of the testing staff, the learning factor η, and the negligence factor ϖ. Specifically, factor α directly impacts the number of errors detected, while factor η influences this number by learning from the cumulative error pattern $M(t)$. Both factors α and η positively contribute to the efficiency of error detection, whereas the negligence factor ϖ has a negative impact, potentially leading to an increase in new errors.

However, estimating these factors poses significant challenges in the absence of historical data, as statistical modeling relies on an adequate dataset for accurate parameter estimation. In such scenarios, Bayesian analysis proves invaluable, forming a key component of the study framework. This approach allows software developers to incorporate expert insights into the prior analysis while simultaneously collecting current data to perform a posterior analysis. By combining these methods, the prior analysis can be refined, resulting in more reliable and robust estimations.

Finally, cost analysis is a critical factor in evaluating various software testing strategies. Developers must consider all associated expenses to ensure that the desired improvements in reliability are achieved within the project's financial constraints, which constitutes the third component of the study framework. In deriving the primary model, it is assumed that the total number of software errors can be represented as a function of testing time, denoted by $A(t)$. This function incorporates both the initial number of errors, a, and the emergence of new errors due to the negligence factor ϖ. Consequently, the total number of software errors increases over time, as described by the following equation:

$$A(t) = a + \varpi M(t), \tag{4}$$

The detection rate is a metric that quantifies the speed at which errors are identified during the testing process. It is represented by the following differential equation:

$$D(t) = \frac{dM(t)}{A(t) - M(t)} = \alpha + \eta F(t). \tag{5}$$

In this context, $F(t)$ represents the cumulative fraction of errors detected within the time interval $[0,t]$, while $A(t) - M(t)$ refers to the remaining undetected errors at time t. It is crucial for the parameters α and η in Equation (5) to be non-negative, as this ensures they positively influence the testing process. This condition indicates the presence of a learning effect, which is dependent on the value of $F(t)$. Given that the cumulative error detection pattern is defined as $F(t) = M(t)/a$, and the total error function is expressed as $A(t) = a + \varpi M(t)$, Equation (5) can be reformulated as follows:

$$D(t) = \frac{dM(t)}{a-(1-\varpi)M(t)} = \alpha + \left(\frac{\eta}{a}\right) M(t). \tag{6}$$

Using this, the mean value function $M(t)$ can be derived through differential equation methods, as shown below:

$$\frac{dM(t)}{\left(\alpha + \left(\frac{\eta}{a}\right) M(t)\right)(a - (1-\varpi)M(t))} = 1. \tag{7}$$

The left side of the equation can be separated into the two factions as follows:

$$\frac{\eta dM(t)}{(\alpha(1-\varpi)+\eta)(a\alpha+\eta M(t))} + \frac{(1-\varpi)dM(t)}{(\alpha(1-\varpi)+\eta)(a-(1-\varpi)M(t))} = 1. \tag{8}$$

By integrating both sides of the equation, the solution is obtained as:

$$\int \left(\frac{\eta dM\,(t)}{(\alpha\,(1-\varpi)+\eta)\,(a\alpha+\eta M\,(t))} + \frac{(1-\varpi)\,dM\,(t)}{(\alpha\,(1-\varpi)+\eta)\,(a-(1-\varpi)\,M\,(t))} \right) dt = \int 1\,dt. \tag{9}$$

Since $\int \frac{d\xi(t)}{\xi(t)}dt = \ln(\xi\,(t))+c$, Equation (9) can be derived and simplified in the following manner:

$$\frac{\ln(a\alpha+\eta M\,(t))}{\eta+\alpha(1-\varpi)} - \frac{\ln(a-(1-\varpi)M\,(t))}{\eta+\alpha(1-\varpi)} = t+c. \tag{10}$$

To solve for $M\,(t)$ by transforming exponential functions, we first isolate it in Equation (10), and the form of $M\,(t)$ with an unknown constant c is given by:

$$M\,(t) = \frac{ae^{(\alpha+\eta)(t+c)} - a\alpha e^{\alpha\varpi(t+c)}}{(1-\varpi)\,e^{(\alpha+\eta)(t+c)} + \eta e^{\alpha\varpi(t+c)}} \tag{11}$$

With the initial condition $M\,(0) = 0$ (indicating that no errors have been detected at time $t = 0$), Equation (10) can be simplified by removing the unknown constant. This leads to the following expression:

$$M(0) = \frac{ae^{(\alpha+\eta)c} - a\alpha e^{\alpha\varpi c}}{(1-\varpi)\,e^{(\alpha+\eta)c} + \eta e^{\alpha\varpi c}} = 0 \tag{12}$$

Solving for the constant c in Equation (10), we find:

$$c = \frac{\ln(\alpha)}{\eta+\alpha(1-\varpi)} \tag{13}$$

By substituting this constant back into Equation (11), the full expression for $M\,(t)$ is obtained (14), as shown at the bottom of the page.

This equation represents the original form of the mean value function, which estimates the average cumulative number of detected errors. Moreover, since $e^{ln(\alpha)} = \alpha$, the expression for $M\,(t)$ can be simplified to the following form:

$$M\,(t) = \frac{\left(e^{\alpha\varpi t} - e^{(\alpha+\eta)t}\right) a\alpha^{\frac{\alpha+\eta}{(1-\varpi)\alpha+\eta}}}{\eta\alpha^{\frac{\alpha\varpi}{(1-\varpi)\alpha+\eta}} e^{\alpha\varpi t} + (1-\varpi)\,\alpha^{\frac{\alpha+\eta}{(1-\varpi)\alpha+\eta}} e^{(\alpha+\eta)t}} \tag{15}$$

Software managers often need to track the number of errors detected at a specific time t to monitor the progress of the testing phase. To accomplish this, the intensity function $\lambda\,(t)$, representing the rate of error detection, must be derived. By taking the first derivative of $M\,(t)$, the intensity function \λ (t) is obtained (16), as shown at the bottom of the next page.

The intensity function $\lambda\,(t)$ represents the number of errors detected at any given time t, enabling software managers to identify when peak error detection occurs. Furthermore, to evaluate debugging efficiency during the testing period, software managers can use an indicator for practical measurement. Zhang and Pham et al. [3] defined the detection rate as $(t) = dM\,(t)\,/(a-M\,(t))$. However, in this study, the remaining errors in the system at time t have been redefined as $A\,(t) - M\,(t)$, resulting in a revised expression for the error detection rate: $D\,(t) = dM\,(t)\,/(A\,(t) - M\,(t))$. It is important to emphasize that this error detection rate is a strictly increasing function, indicating that the efficiency of debugging improves as testing progresses over time.

B. BAYESIAN ANALYSIS IN THE ABSENCE OF ADEQUATE HISTORICAL DATA

When there is a lack of sufficient historical data to estimate the values of the unknown parameters α, η, and ϖ in the debugging phase of a new project, a manager must find alternative methods to estimate these parameters. This is crucial for evaluating the testing efficiency and cost of the new project in preparation for a software release. Additionally, managers might consider various testing strategies based on different allocations of human resources. However, these strategies often cannot rely on similar historical datasets for parameter estimation, rendering the use of the Least Squares Estimation (LSE) method ineffective in such cases.

Fortunately, Bayesian statistical methods offer a solution by allowing domain experts to provide initial estimates (prior judgments) of parameter values, which can then be refined using collected data. Bayesian analysis can be categorized into two phases: (1) prior analysis and (2) posterior analysis. While a prior analysis can be conducted beforehand, more accurate estimations can be achieved through posterior analysis once data has been collected.

To perform the prior analysis, it is essential to define an appropriate joint prior distribution that captures the uncertainty of the parameters α, η and ϖ. For this analysis, we assume that α and η follow a bivariate gamma distribution, $f\,(\alpha, \eta)$, while ϖ follows a separate gamma distribution, $f\,(\varpi)$. The bivariate gamma distribution was introduced by Schucany et al. [22] and Moran [23], specifically to address challenges in reliability and bioinformatics. These distributions are applied here to define the joint prior distribution for α, η and ϖ as (17) and (18), as shown at the bottom of the next page.

Since $f\,(\alpha, \eta)$ is independent of $f\,(\varpi)$, the joint prior distribution $f\,(\alpha, \eta, \varpi)$ is the product of $f\,(\alpha, \eta)$ and $f\,(\varpi)$. The term $\rho_{\alpha,\eta}$ represents the correlation coefficient between α and η. Typically, since instinct debugging skills (α) often correlate with strong learning abilities (η), the correlation coefficient $\rho_{\alpha,\eta}$ is positive in most scenarios. Figure 2 presents the joint probability distribution $f\,(\alpha, \eta)$ under varying

$$M\,(t) = \frac{ae^{(\alpha+\eta)(t+\ln\alpha)/((1-\varpi)\alpha+\eta))} - a\alpha e^{\alpha\varpi(t+\ln(\alpha)/((1-\varpi)\alpha+\eta))}}{\eta e^{\alpha\varpi(t+\ln(\alpha)/((1-\varpi)\alpha+\eta))} + (1-\varpi)e^{(\alpha+\eta)(t+\ln(\alpha)/((1-\varpi)\alpha+\eta))}}. \tag{14}$$

correlation coefficients $\rho_{\alpha,\eta}$, illustrating the influence of different correlations on the shape of the joint distribution.

The Gamma and incomplete Gamma functions are defined as follows: $\boldsymbol{\Gamma}[z] = \int_0^\infty x^{z-1}e^{-x}dx$ and $\boldsymbol{\Gamma}[z_0, z_1, z_2] = \int_{z_1}^{z_2} x^{z_0-1}e^{-x}dx$ respectively. In these equations, $\Phi^{-1}[\cdot]$ refers to the inverse of the standard normal cumulative distribution function. The scale parameters for α, η and ϖ are represented by θ_α, θ_η and θ_ϖ, respectively, while their corresponding shape parameters are γ_α, γ_η and γ_ϖ. Although these scale and shape parameters cannot be directly observed in practice, they can be estimated by experts based on the statistical properties—such as the mean and standard deviation—of α, η and ϖ. The relationships between the scale and shape parameters and the statistical characteristics are given by the following equations, based on experts' judgment for $E(\alpha)$, $\sigma(\alpha)$, $E(\eta)$, $\sigma(\eta)$, $E(\varpi)$ and $\sigma(\varpi)$: $\gamma_\alpha = \frac{\sigma(\alpha)^2}{E(\alpha)}$, $\theta_\alpha = \left(\frac{E(\alpha)}{\sigma(\alpha)}\right)^2$, $\gamma_\eta = \frac{\sigma(\eta)^2}{E(\eta)}$, $\theta_\eta = \left(\frac{E(\eta)}{\sigma(\eta)}\right)^2$, $\gamma_\varpi = \frac{\sigma(\varpi)^2}{E(\varpi)}$, and $\theta_\varpi = \left(\frac{E(\varpi)}{\sigma(\varpi)}\right)^2$.

Furthermore, the expected mean value of total detected errors, $E[M(t)]$, and the expected conditional software reliability, $E[R|t)]$, in the prior analysis can be computed using the following equations:

$$E[M(T)] = \int_0^\infty \int_0^\infty \int_0^\infty M(T) f(\alpha, \eta, \varpi)\, d\varpi\, d\eta\, d\alpha \tag{19}$$

$$E[R(x|T)] = \int_0^\infty \int_0^\infty \int_0^\infty R(x|T) f(\alpha, \eta, \varpi)\, d\varpi\, d\eta\, d\alpha \tag{20}$$

However, managers may not fully rely on the prior analysis results and could decide to gather additional testing data to refine their estimates. To obtain the mean value and software reliability in the posterior analysis, it is first necessary to derive the posterior distribution. According to the natural conjugate distribution property, the posterior distribution belongs to the same family as the prior distribution. The posterior distribution can be written as:

$$\begin{aligned} g\left(\alpha, \eta, \varpi, \Delta^{(n)}\right) &\propto f(\alpha, \eta, \varpi)\, L\left(\Delta^{(n)}|\alpha, \eta, \varpi\right) \\ &= Kf(\alpha, \eta, \varpi)\, L\left(\Delta^{(n)}|\alpha, \eta, \varpi\right) \end{aligned} \tag{21}$$

Here, $L\left(\Delta^{(n)}|\alpha, \eta, \varpi\right)$ represents the likelihood function of the NHPP based on the observed dataset $\Delta^{(n)} = \{t_1, t_2, .., t_n\}$, and can be calculated as:

$$L\left(\Delta^{(n)} \mid \alpha, \eta, \varpi\right) = \prod\nolimits_{i=1}^{n} \lambda(t_i) e^{-M(t_n)}. \tag{22}$$

The constant K is a normalizing factor that ensures the posterior distribution integrates to 1, and it is determined by the following equation:

$$K = \frac{1}{\int_0^\infty \int_0^\infty \int_0^\infty f(\alpha, \eta, \varpi)\, L\left(\Delta^{(n)} \mid \alpha, \eta, \varpi\right) d\varpi\, d\eta\, d\alpha} \tag{23}$$

Once the posterior distribution is determined, the mean value $M(T)$ and the software reliability $R(x|T)$ in the posterior analysis can be calculated as:

$$\begin{aligned} &E'[M(T)] \\ &= \int_0^\infty \int_0^\infty \int_0^\infty M(T) g\left(\alpha, \eta, \varpi, \Delta^{(n)}\right) d\varpi\, d\eta\, d\alpha \end{aligned} \tag{24}$$

$$\begin{aligned} &E'[R(x|T)] \\ &= \int_0^\infty \int_0^\infty \int_0^\infty R(x|T) g\left(\alpha, \eta, \varpi, \Delta^{(n)}\right) d\varpi\, d\eta\, d\alpha \end{aligned} \tag{25}$$

It is important to note that calculating $E'[M(T)]$ and $E'[R(x|T)]$ involves complex multidimensional numerical integration, which may require the assistance of a computational engine to obtain results efficiently.

$$\lambda(t) = \frac{dM(t)}{dt} = \frac{a\alpha^{\frac{\alpha+\eta}{(1-\varpi)\alpha+\eta}}\left((1-\varpi)\alpha+\eta\right)\left((1-\varpi)\alpha^{\frac{\alpha+\eta}{(1-\varpi)\alpha+\eta}} + \eta\alpha^{\frac{\alpha\varpi}{(1-\varpi)\alpha+\eta}}\right)e^{((1+\alpha)\varpi+\eta)t}}{\left((1-\varpi)\alpha^{\frac{\alpha+\eta}{(1-\varpi)\alpha+\eta}}e^{(\alpha+\eta)t} + \eta\alpha^{\frac{\alpha\varpi}{(1-\varpi)\alpha+\eta}}e^{\alpha\varpi t}\right)^2} \tag{16}$$

$$\begin{aligned} f(\alpha, \eta, \varpi) = &\left(\frac{\frac{\varpi^{\theta_\varpi - 1}e^{-\left(\frac{\varpi}{\gamma_\varpi}\right)}}{\boldsymbol{\Gamma}[\theta_\varpi]\gamma_\varpi^{\theta_\varpi}}}{\sqrt{1-\rho_{\alpha,\eta}^2}}\right)\left(\frac{\alpha^{\theta_\alpha-1}\eta^{\theta_\eta-1}e^{-\left(\frac{\alpha}{\gamma_\alpha}+\frac{\eta}{\gamma_\eta}\right)}}{\boldsymbol{\Gamma}[\theta_\alpha]\boldsymbol{\Gamma}[\theta_\eta]\gamma_\alpha^{\theta_\alpha}\gamma_\eta^{\theta_\eta}}\right) \\ &e^{-\frac{\left(\rho_{\alpha,\eta}\Phi^{-1}\left[1-\frac{\boldsymbol{\Gamma}\left[\theta_\alpha, \frac{\alpha}{\gamma_\alpha}\right]}{\boldsymbol{\Gamma}[\theta_\alpha]}\right]\right)^2 + \left(\rho_{\alpha,\eta}\Phi^{-1}\left[1-\frac{\Gamma\left[\theta_\eta, \frac{\eta}{\gamma_\eta}\right]}{\boldsymbol{\Gamma}[\theta_\eta]}\right]\right)^2 - 2\rho_{\alpha,\eta}\Phi^{-1}\left[1-\frac{\boldsymbol{\Gamma}\left[\theta_\alpha, \frac{\alpha}{\gamma_\alpha}\right]}{\boldsymbol{\Gamma}[\theta_\alpha]}\right]\Phi^{-1}\left[1-\frac{\Gamma\left[\theta_\eta, \frac{\eta}{\gamma_\eta}\right]}{\boldsymbol{\Gamma}[\theta_\eta]}\right]}{2\left(1-\rho_{\alpha,\eta}^2\right)}} \end{aligned} \tag{17}$$

$$f(\alpha, \eta, \varpi) = \frac{\left(\frac{\varpi^{\theta_\varpi-1}e^{-\left(\frac{\varpi}{\gamma_\varpi}\right)}}{\boldsymbol{\Gamma}[\theta_\varpi]\gamma_\varpi^{\theta_\varpi}}\right)\left(\alpha^{\theta_\alpha-1}e^{-\frac{\alpha}{\gamma_\alpha}}\right)\left(\eta^{\theta_\eta-1}e^{-\frac{\eta}{\gamma_\eta}}\right)\left(3\rho_{\alpha,\eta}\left(\frac{\mathbf{2}\Gamma\left[\theta_\alpha, 0, \frac{\alpha}{\gamma_\alpha}\right]}{\boldsymbol{\Gamma}[\theta_\alpha]} - 1\right)\left(\frac{\mathbf{2\Gamma}\left[\theta_\eta, 0, \frac{\eta}{\gamma_\eta}\right]}{\boldsymbol{\Gamma}[\theta_\eta]} - 1\right) + 1\right)}{\left(\gamma_\alpha^{\theta_\alpha}\boldsymbol{\Gamma}[\theta_\alpha]\right)\left(\gamma_\eta^{\theta_\eta}\boldsymbol{\Gamma}[\theta_\eta]\right)} \tag{18}$$

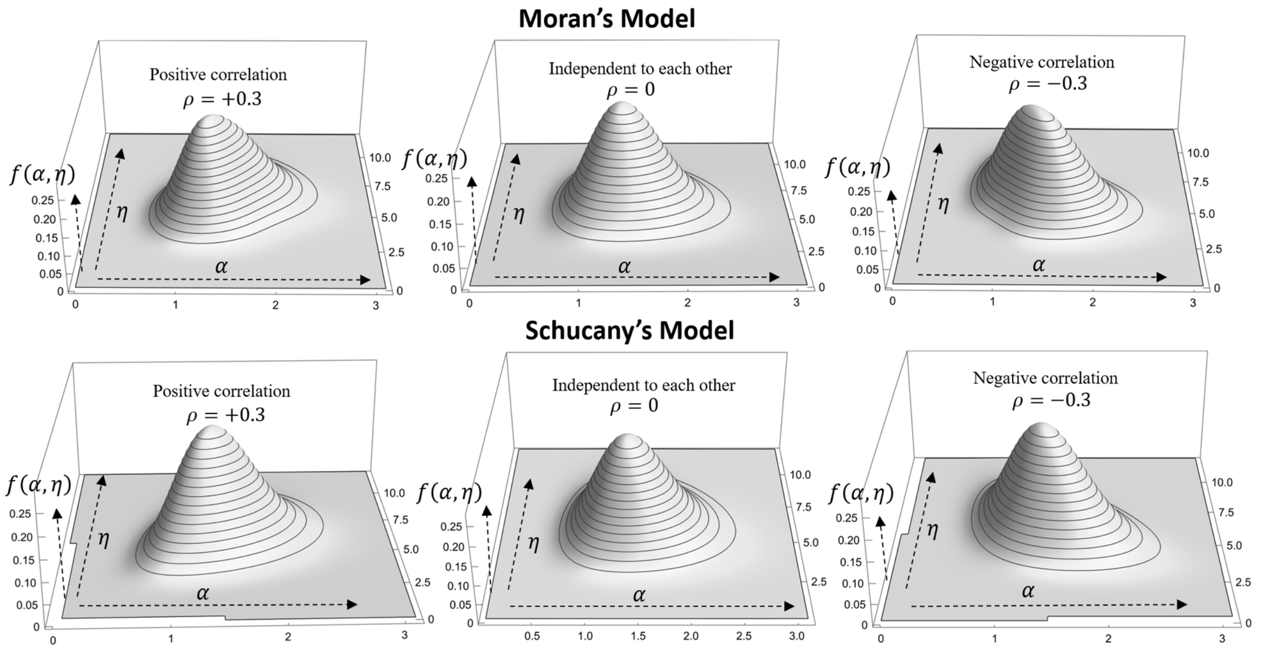

FIGURE 2. The joint probability distributions $\mathbf{f}(\alpha,\eta)$ with different correlations.

To incorporate cost functions into the prior analysis and determine the optimal testing cost, software reliability, and release timing, software engineers will develop several alternatives for the project manager to evaluate based on specific management needs. This process requires a detailed assessment and estimation of both cost-related and model parameters beforehand. Once these parameters are established, the project manager can select the most cost-effective option for the prior analysis. If the manager remains uncertain about the outcomes of the prior analysis, they may collect additional data from the ongoing testing process to further refine their decisions. This leads to a posterior analysis, which allows managers to reassess the situation and adjust the software release schedule as needed based on the updated insights.

C. COST MODELS FOR DETERMINING OPTIMAL SOFTWARE RELEASE TIMING

Typically, a software development manager is interested in determining the optimal time to stop testing in order to minimize costs while meeting specific software quality requirements. It is widely accepted that prolonged testing results in more reliable software. However, extended testing also leads to higher costs and may postpone the opportunity to commercialize the software. Therefore, the decision to release the software must be made with careful consideration.

In this study, the total expected cost is divided into several key components: setup cost, routine cost, error correction cost, risk cost, and opportunity cost. To facilitate both prior and posterior analyses, two mathematical programming models are proposed for alternative p:

$$\begin{aligned} &Min\ E\left[TC_p\left(T\right)\right] \\ &\quad = SC_p + GC_pT + \sum\nolimits_{v=0}^{V} EC_p^vE\left[t_r|\psi_v,\tau_v\right]q_vE\left[M_p(T)\right] \\ &\qquad + RC\left(E\left[A_p\left(T\right) - M_p(T)\right]\right)^{\kappa_1} + OC(T) \\ &Subject\ to: E\left[R_p\left(x\,|\,T\right)\right] \geq R_m. \end{aligned} \tag{26}$$

$$\begin{aligned} &Min\ E'\left[TC_p\left(T\right)\right] \\ &\quad = SC_p + GC_pT + \sum\nolimits_{i=0}^{V} EC_p^vE\left[t_r|\psi_v,\tau_v\right]q_vE'\left[M_p(T)\right] \\ &\qquad + RC\left(E\left[A_p\left(T\right) - M_p(T)\right]\right)^{\kappa_1} + OC(T) \\ &Subject\ to: E'\left[R_p\left(x\,|\,T\right)\right] \geq R_m. \end{aligned} \tag{27}$$

For the cost model associated with alternative p, the terms are defined as follows: SC_p refers to the setup cost, which includes the initial expenses and necessary preparatory work for alternative p. $E\left[\cdot\right]$ and $E'\left[\cdot\right]$ represent the expected values in the prior and posterior analyses, respectively. GC_pT denotes the routine cost incurred over the testing period $[0,T]$. Software errors are categorized into three types: simple(s), complex (c), and difficult (d), with each type requiring a different amount of correction time. In this study, we assume that the proportion and correction time for each type of error can be reasonably estimated. The correction times are modeled using a truncated exponential distribution (Huang et al. [43]), described as follows:

$$G(\psi_v,\tau_v) = \left(1 - e^{-\tau_v/\psi_v}\right)^{-1}\psi_v^{-1}e^{-t_r/\psi_v}. \tag{28}$$

q_v represents the proportion of errors of type v in the software system, while τ_v is the maximum allowable time to correct an error of type v. The random variable t_r corresponds to the actual time required to resolve an error, and ψ_v denotes the parameter representing the expected time needed to fix an error of type v. Consequently, the maximum correction time for difficult errors will be longer than that for complex or simple errors ($\tau_d > \tau_c > \tau_s$). Based on these assumptions, the expected correction time for an error of type v, given the parameters ψ_v and τ_v, can be estimated using the following equation:

$$E[t_r|\psi_v, \tau_v] = \int_0^{\tau_v} G_y(\psi_v, \tau_v)t_r dt_r. \tag{29}$$

Considering all the factors mentioned, the total cost of error correction during the software testing process can be expressed as: $\sum_v EC_p^v E\left[t_r|\psi_v, \tau_v\right] q_v E\left[M_p(T)\right]$. In this expression, EC_p^v refers to the cost associated with correcting errors of type v, while $E\left[t_r|\psi_v, \tau_v\right]$ represents the expected time required to fix these errors, given the parameters ψ_v and τ_v. This formula encapsulates the cost implications of error correction across various error types, accounting for their respective proportions and time to resolution.

The risk cost is represented by $RC\left(E\left[A_p\left(T\right)- M_p(T)\right]\right)^{\kappa_1}$, where κ_1 is the risk aversion factor that influences how much weight is placed on the risk associated with undetected errors remaining in the system at time T. This risk cost is crucial because the number of residual errors can have a direct impact on the software's reliability and performance once released, affecting customer satisfaction and potentially increasing long-term maintenance costs. The higher the risk aversion factor, the more weight the manager places on reducing these remaining errors.

Opportunity cost, denoted as $OC(T)$, reflects the potential loss of opportunities due to extended testing or delayed software release. It can be expressed by the following equation: $OC(T) = \omega_0(\omega_1 + T)^{\omega_2}$. In this equation, ω_0 is the scale coefficient, ω_1 represents the intercept value, and ω_2 determines the rate at which opportunity cost escalates over time. This cost accounts for the impact of delaying the software release, such as missed market opportunities, delayed revenue, or a reduced competitive edge. As time progresses, the opportunity cost increases, which incentivizes a timely release to minimize lost opportunities.

In addition to these cost considerations, the manager must ensure that the software reliability, R_m, meets or exceeds a predefined minimum threshold to satisfy both internal management objectives and external client requirements. This reliability constraint is critical to maintaining customer trust and avoiding post-release issues that could lead to reputational damage or increased support costs.

Taking all of these factors into account—including error correction costs, risk costs, opportunity costs, and reliability constraints—the manager can apply two programming models to evaluate various testing and release strategies. These models enable the decision-maker to determine the optimal timing for the software release, both in the prior analysis (based on initial estimates) and in the posterior analysis (refined with additional data gathered during the testing phase). By balancing these costs and constraints, the manager can make informed decisions that minimize expenses while ensuring the software meets quality standards and is released at the most advantageous time.

III. MODEL VERIFICATION AND DECISION PROCESS

A. PARAMETER ESTIMATION AND MODEL VALIDATION

The Least Squares Estimation (LSE) method is widely used for estimating the parameters of the mean value function in statistical modeling. This approach aims to minimize the sum of squared errors, making it an effective tool for evaluating the fitting accuracy of both the proposed model and other existing models. Consider a dataset of n observed pairs, (t_0, M_0), (t_1, M_1), (t_2, M_2),.., (t_n, M_n), M_i represents the cumulative number of detected errors at time t_i. The total sum of squared errors for this dataset can be expressed as:

$$Er(\alpha, \eta, \varpi) = \sum\nolimits_{i=1}^{n} (M_i - M(t_i))^2. \tag{30}$$

To estimate the parameters α, η, and ϖ, the first-order derivatives of Equation (29) with respect to each parameter are taken and set to zero. Solving the resulting system of equations provides the estimated parameter values:

$$\frac{\partial Er(\alpha, \eta, \varpi)}{\partial \alpha} = \frac{\partial Er(\alpha, \eta, \varpi)}{\partial \eta} = \frac{\partial Er(\alpha, \eta, \varpi)}{\partial \varpi} = 0. \tag{31}$$

This approach allows the parameters to be determined by solving the simultaneous equations. To assess the goodness of fit of the proposed SRGM, publicly available datasets were analyzed. The proposed SRGM was also compared with other existing imperfect debugging SRGMs to evaluate their respective fitting performances. Table 1 lists the sources of the four open datasets used for model validation, and Table 2 provides the mean value functions and error detection rate functions for the imperfect debugging SRGMs examined. This comprehensive comparison aids in determining how effectively the proposed model performs relative to other models in terms of accurately fitting the observed data.

TABLE 1. The source of the open datasets.

Dataset	Reference	Source
(1)	Zhang & Pham [3]	Failure data of Telecommunication system
(2)	Wang & Wu [7]	Medium scale software project
(3)	Peng et al. [5]	Testing data for the Room Air Development Center
(4)	Singpurwalla & Wilson [1]	Failure data of NTDS system

In this study, we assessed the performance of four different models using mean squared error (MSE) and R-squared (R-sq) as primary metrics to evaluate their accuracy in fitting the data. Figures 2 illustrates the fitting results and parameter estimates for each model and each data when applied to

TABLE 2. Mean value functions and error detection rate functions for the imperfect debugging SRGMs.

Imperfect Debugging SRGMs	Functions of Mean Value $M(t)$ and Detection Rate $D(t)$
Pham et al. [1]	$M(t) = \frac{a(1-e^{-bt})(1+\alpha t+\frac{\alpha t}{b})}{1+e^{-bt}\beta}, D(t) = \frac{b}{1+\beta e^{-bt}}.$
Kapur et al. [3]	$M(t) = \frac{a(1-e^{-bp(1-\alpha)t})}{1-\alpha}, D(t) = \frac{(1-\alpha)bp}{1-\alpha e^{-bpt(-1+\alpha)}}.$
Wang et al. [7]	$M(t) = \frac{a(\alpha t)^d}{1+(\alpha t)^d} + C(1-e^{-bt}) - ad(\alpha t)^d e^{-bt}\left(\frac{1}{d} + \frac{bt}{1+d} + \frac{b^2t^2}{2(2+d)}\right), D(t) = b.$
Proposed model	$M(t) = \frac{\left(e^{\alpha\varpi t}-e^{(\alpha+\eta)t}\right)a\alpha^{\frac{\alpha+\eta}{(1-\varpi)\alpha+\eta}}}{\eta\alpha^{\frac{\alpha\varpi}{(1-\varpi)\alpha+\eta}}e^{\alpha\varpi t}+(1-\varpi)\alpha^{\frac{\alpha+\eta}{(1-\varpi)\alpha+\eta}}e^{(\alpha+\eta)t}},$ $D(t) = \frac{((1-\varpi)\alpha+\eta)\alpha^{\frac{\alpha+\eta}{(1-\varpi)\alpha+\eta}}}{(1-\varpi)\alpha^{\frac{\alpha+\eta}{(1-\varpi)\alpha+\eta}}+\eta\alpha^{\frac{\alpha\varpi}{(1-\varpi)\alpha+\eta}}e^{(\alpha\varpi-(\alpha+\eta))t}}.$

publicly available datasets. In addition to the fitted curves, the figures also present the estimated parameter values. The dashed lines surrounding the $M(t)$ curve represent the 95% confidence intervals, providing a visual representation of the uncertainty in the estimates.

As demonstrated in Figure 3, the proposed model effectively captures the trends in the nonlinear regression analysis, performing particularly well across multiple datasets. In fact, in most cases, the proposed model outperformed the other models in terms of fitting accuracy, with R-sq values of 99.31%, 97.22%, 99.83%, and 99.20% for datasets 1 through 4, respectively. Pham's model also exhibited strong performance, especially in adapting to S-shaped and concave datasets, as illustrated in Figure 3-Datasets (1,4). The flexibility of this model allowed it to provide a robust fit in various scenarios. However, as shown in Figure 3-Dataset (4), Kapur's model struggled with S-shaped data and appeared better suited for datasets with exponential or concave patterns. Despite this limitation, it still performed adequately on datasets (2) and (3).

In contrast, Wang's model, illustrated in Figure 3, performed exceptionally well when applied to datasets (1) and (2), achieving R-sq values of 99.43% and 98.81%, respectively. The enhanced flexibility of Wang's model can be attributed to its incorporation of five parameters, compared to the four parameters utilized by the other models. This additional parameter enables the model to adapt more effectively to variations in the data. However, it is important to note that while an increased number of parameters generally enhances a model's flexibility and adaptability, it does not necessarily guarantee superior performance. In fact, despite having fewer parameters, the proposed model remained competitive with Wang's model in terms of fitting accuracy, demonstrating that a well-constructed model can yield excellent results even with a reduced number of parameters.

Overall, the proposed model demonstrated superior performance across all cases, consistently achieving high fitting accuracy. In addition to its robustness, the model's parameters are relatively straightforward and intuitive to interpret. This clarity enables managers and practitioners to easily adjust the model's parameters to accommodate evolving testing scenarios, ensuring that the model remains applicable and relevant as new data emerges. Therefore, the proposed model not only delivers exceptional fitting performance but also offers practical flexibility for future adjustments.

B. PROCEDURE FOR DEALING WITH COST MODELS

While Section II-C introduces cost models that assist in determining the optimal timing for software releases, the application of these models necessitates a more detailed solution process, particularly when addressing the expected values involved in Bayesian analyses. These expected values are crucial for accurately evaluating the cost functions; however, they pose a challenge because they cannot always be expressed in closed mathematical forms. Consequently, although the optimal functions (25) and (26) are formulated within the framework of mathematical programming, their solutions must be obtained through numerical integration techniques. Specifically, some expected value functions involved in these calculations do not have straightforward analytical solutions, rendering numerical methods indispensable.

To address this issue, multidimensional numerical integration becomes essential, often in conjunction with a high-performance computational engine to manage the complexity and precision required for these calculations. The expected values of $E\left[M_p(T)\right]$, $E'\left[M_p(T)\right]$, $E\left[R_p(x \mid T)\right]$, and $E'\left[R_p(x \mid T)\right]$ must be evaluated using these advanced techniques. Once these expected values are obtained, they facilitate the construction of cost curves for both the prior cost function, $E\left[TC_p(T)\right]$, and the posterior cost function, $E'\left[TC_p(T)\right]$, at various points in time. These cost curves provide a visual and analytical framework for understanding how costs evolve over time, thereby guiding decision-makers in determining the optimal release point for the software.

Figure 4 illustrates the flow chart for the Bayesian decision process.

It is essential to emphasize the significance of the Bayesian updating process in this context. Bayesian methods facilitate an adaptive and dynamic adjustment of initial assumptions or judgments (the prior) based on newly collected data. This process involves continuously refining the prior distribution by incorporating new data sets, with the posterior distribution at any given stage serving as the new prior for future updates. As new data is gathered, this iterative process enables ongoing adjustments to the expected values and cost functions. In practice, this means that the cost models evolve in tandem with the software testing and development process, allowing for real-time recalibration of the optimal release strategy. The flow of this solution procedure, including the Bayesian updating mechanism, is illustrated in Figure 4, which provides a clear depiction of how the optimal functions (25) and (26) are addressed.

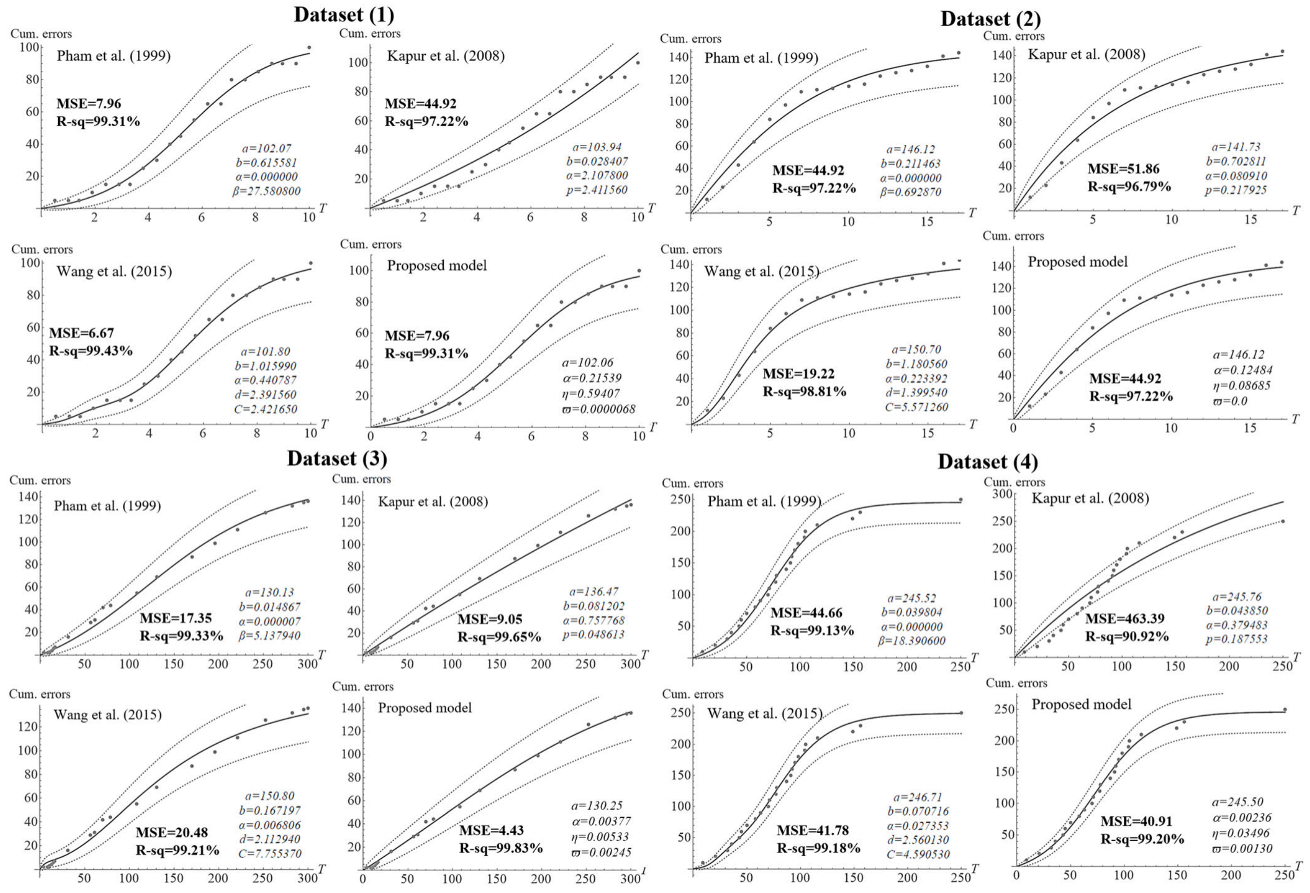

FIGURE 3. The fitting results and parameters estimation.

By evaluating the cost curves generated through this procedure, the optimal software release time, denoted as T^*, can be identified. This T^* represents the ideal moment for releasing the software to balance development costs, quality, and market timing. The determination of T^* depends on the shape and behavior of the cost curves, which can vary based on the specific conditions of the software project. These conditions lead to three distinct scenarios for the optimal release time, each providing a different perspective on managing the trade-offs between cost and software reliability.

Each of these cases will be discussed in detail below, providing insights into how they influence the decision-making process regarding the software release. By examining these various cases, a more informed and strategic decision can be made—one that balances the necessity for thorough testing with the constraints of time, budget.

Case I: In situations where the cost function $E\left[TC_p\left(T\right)\right]$ (or $\mathrm{E}'\left[\mathrm{TC_p}\left(\mathrm{T}\right)\right]$) exhibits a strictly increasing trend over time, the optimal release time is determined by the moment when the desired level of software reliability, denoted as R_m, is first achieved. At this point, the release time is defined as $\mathrm{T} = \mathrm{T}(\mathrm{R_m})$, as illustrated in Case I of Figure 5. Although pure theoretical models may suggest that the best release time is at $T^* = 0$, this approach is impractical in real-world software development environments. Releasing software at the very beginning of the development cycle ($T^* = 0$) would overlook the necessity of meeting baseline quality requirements. Such a strategy would likely lead to significant quality issues that could adversely impact user experience and tarnish the software provider's reputation, ultimately resulting in a loss of customer trust. In practice, this scenario often arises when the costs associated with potential software failures or risks are deemed insignificant or negligible, leading to minimal concern regarding testing.

Case II: If the cost function $\mathrm{E}\left[\mathrm{TC_p}\left(\mathrm{T}\right)\right]$ (or $E'\left[TC_p\left(T\right)\right]$) exhibits a strictly decreasing trend over time, the optimal release time will occur after the point $T(R_m)$, as illustrated in Case II of Figure 5. In theory, extending the testing period indefinitely would lead to continuously increasing reliability, particularly as time approaches infinity. The rationale is that prolonged testing facilitates the identification and resolution of more issues, thereby enhancing software reliability. However, this theoretical model does not align with the practical constraints faced in the software industry. In practice, testing cannot be extended indefinitely due to limited resources, including time, budget, and personnel. Furthermore, as the testing period extends, the benefits of additional testing diminish, resulting in diminishing returns. Beyond a

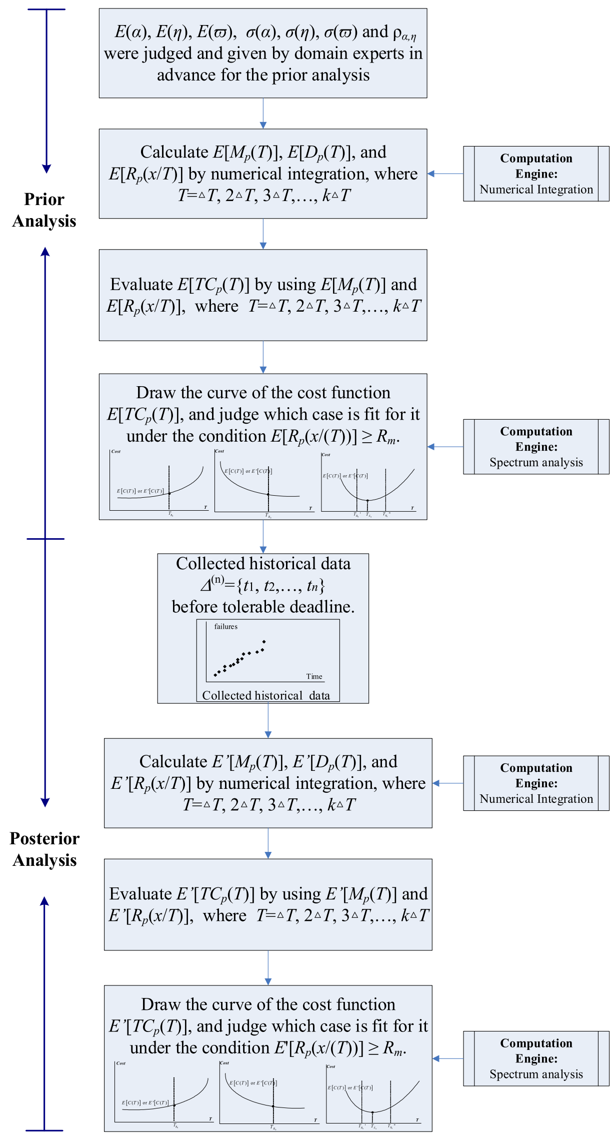
Prior Analysis
Posterior Analysis
E(α), E(η), E(ϖ), σ(α), σ(η), σ(ϖ) and ρα,η were judged and given by domain experts in advance for the prior analysis
Calculate E[Mp(T)], E[Dp(T)], and E[Rp(x/T)] by numerical integration, where T=△T, 2△T, 3△T,…, k△T
Computation Engine: Numerical Integration
Evaluate E[TCp(T)] by using E[Mp(T)] and E[Rp(x/T)], where T=△T, 2△T, 3△T,…, k△T
Draw the curve of the cost function E[TCp(T)], and judge which case is fit for it under the condition E[Rp(x/(T))] ≥ Rm.
Computation Engine: Spectrum analysis
Collected historical data Δ(n)={t1, t2,…, tn} before tolerable deadline.
failures
Time
Collected historical data
Calculate E'[Mp(T)], E'[Dp(T)], and E'[Rp(x/T)] by numerical integration, where T=△T, 2△T, 3△T,…, k△T
Computation Engine: Numerical Integration
Evaluate E'[TCp(T)] by using E'[Mp(T)] and E'[Rp(x/T)], where T=△T, 2△T, 3△T,…, k△T
Draw the curve of the cost function E'[TCp(T)], and judge which case is fit for it under the condition E'[Rp(x/(T))] ≥ Rm.
Computation Engine: Spectrum analysis

FIGURE 4. Flow chart for the Bayesian decision process.

certain threshold, the increase in reliability becomes marginal compared to the resources expended on extended testing. Therefore, in real-world scenarios, if the cost function no longer demonstrates significant reductions after a specific period, it becomes economically prudent to conclude testing and release the software. Prolonging the release only incurs unnecessary costs without substantial gains in reliability. This situation is often observed when developers prioritize reducing testing expenses without adequately considering the opportunity costs associated with delayed market entry and the potential loss of revenue or competitive advantage.

Case III: When the cost function $E\left[TC_p\left(T\right)\right]$ (or $E'\left[TC_p\left(T\right)\right]$) exhibits a convex shape, the optimal release time is identified as the point occurring after $T(R_m)$ where the total cost reaches its minimum value. This scenario is visually represented in Case III of Figure 5. The convex nature of the cost function indicates a complex relationship between the duration of testing and the associated costs. In the initial stages, extending the testing period leads to a reduction in costs as issues are identified and resolved, thereby enhancing reliability and decreasing the likelihood of post-release defects. However, beyond a certain threshold, the cost of testing begins to rise again. This increase occurs because the benefits gained from extended testing no longer outweigh the additional time, effort, and resources being expended. As testing continues, the effort required to identify and rectify remaining defects becomes increasingly burdensome, resulting in higher costs. The optimal release time in this scenario is the point at which the cost function reaches its minimum, striking a balance between the advantages of improved reliability from extended testing and the escalating costs associated with prolonged testing duration. This situation often arises when all relevant factors—such as the risk of defects, opportunity costs of delayed release, and testing efficiency—are carefully evaluated. By considering these elements, decision-makers can arrive at a well-informed conclusion regarding the ideal time to release the software, optimizing both cost-effectiveness and product quality.

In summary, determining the optimal time to release software is a complex decision that necessitates a thorough analysis of the underlying cost functions across various scenarios. Each of the cases outlined above highlights how different factors, such as testing duration, software reliability, and associated costs, influence the timing of the release. By understanding these dynamics, software development managers can make informed decisions that achieve an appropriate balance between ensuring product quality, controlling costs, and meeting market demands. This careful equilibrium is essential for maximizing the value of the software while minimizing risks, ultimately leading to a successful product release that satisfies both business objectives and customer expectations.

C. DESIGN OF COMPUTERIZED IMPLEMENTATION SYSTEM

To effectively address this programming problem, a computerized application system is essential for guiding the decision-making process and determining the optimal timing for software releases. To ensure the system operates efficiently, several key components must be integrated into its design. This study incorporates several key components, including a specialized database, a model repository, a data normalization module, a custom application programming interface (API), and a powerful computation engine.

The database is utilized to store a diverse array of critical information, including cost parameters, historical failure data from various systems, and expert knowledge. Similarly, the model repository is intended to house a range of software reliability growth models (SRGMs) and the mathematical models that underpin software release decision policies. To ensure efficient storage and retrieval of information from both the database and the model repository, a data normalization mechanism is necessary to standardize inconsistent

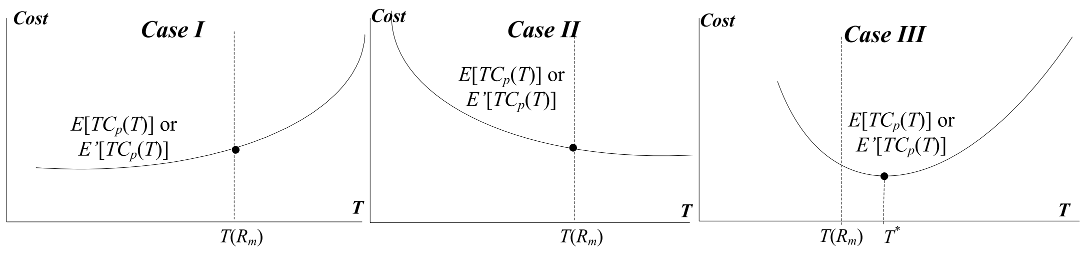

FIGURE 5. Cost functions for software testing in cases I, II, and III.

data formats. This can be accomplished through a data normalization program module, which optimizes the processes of storing and accessing data, thereby enhancing their effectiveness and efficiency.

Additionally, solving the proposed models involves advanced mathematical programming and numerical integration, which calls for a highly capable computational engine. This engine can be developed internally by software engineers or sourced from third-party providers, such as R packages, Wolfram Research or Lingo Systems, to ensure the necessary computational power and accuracy. To facilitate seamless interaction between the designed system and the computational engine, an API serves as the most effective means of exchanging information, ensuring that the computational engine is utilized efficiently and conveniently. The following sections will introduce the system design and provide detailed information on its implementation.

To enhance manageability, the system is organized into two distinct subsystems. The first is the model management system, which is specifically designed for software engineers, domain experts, and testing personnel to efficiently manage and maintain the database and model repository. The second subsystem is the decision support system, which provides decision-makers with the essential information required to make well-informed choices. Within the model management system, engineers begin by collecting historical data from previous software testing projects and selecting suitable SRGMs while taking the cost structure into account. This gathered data is then organized and stored in the database through the model management system. Given that the accuracy of test cost estimates is closely tied to the efficiency of software testing, domain experts are instrumental in identifying which SRGM is most appropriate for the specific project. To safeguard sensitive commercial information, access to the system is strictly controlled, ensuring that only engineers, domain experts, and technical managers can interact with their respective subsystems. Additionally, during the initial development phase, system programmers establish programming models in the model repository, excluding specific parameters to allow for adaptation to different projects as needed. This clear separation of subsystems ensures secure and streamlined operations, enhancing the overall effectiveness of software testing and decision-making processes.

The decision support system is designed for use by upper management decision-makers, providing comprehensive and integrated information to enhance their decision-making processes. The system can provide not only prior and posterior analyses but also sensitive, risk, and various alternatives tracking analyses. By leveraging the data stored in the database and model repository, decision-makers can analyze all relevant information and confidently arrive at optimal decisions. However, due to the complexity involved in determining the best course of action, a computation engine may be necessary. By utilizing the API, decision-makers can easily access and employ various computational tools developed by both internal and external programmers. Figure 6 presents a visual representation of the system's implementation architecture.

IV. APPLICATION AND NUMERICAL ANALYSIS

Let us consider a scenario in which a software company has successfully developed industrial engineering software. After completing the intricate coding process, the project manager is now responsible for creating a comprehensive software testing plan and determining the optimal timing for the software's market release. Given the software's complexity, it is likely that numerous hidden errors are embedded within the code. To address this challenge, the manager must carefully allocate testing resources and assign skilled personnel to perform the critical task of debugging.

In this context, estimating the number of software errors is essential, and one effective method to achieve this is through the application of an error seeding technique. This approach allows the manager to gain a clearer understanding of the potential errors that may exist in the software. To ensure that the software meets industry standards and user expectations, a minimum reliability threshold of 0.9 has been established. The manager is responsible for ensuring that the software's quality exceeds this threshold before the product can be safely released to the market.

To streamline the testing process, the software engineers have proposed three distinct testing alternatives. Each alternative presents a unique combination of cost and reliability, necessitating that the manager thoroughly assess these choices to identify the most appropriate one for the project. However, the lack of historical data for parameter evaluation poses a considerable challenge. To address this issue, the

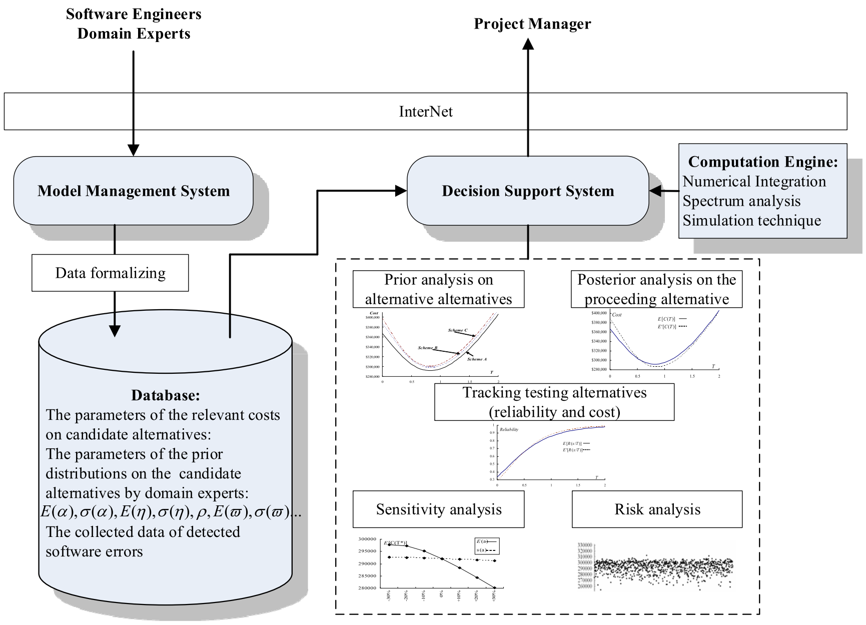
Software Engineers
Domain Experts
Project Manager
InterNet
Model Management System
Decision Support System
Computation Engine:
Numerical Integration
Spectrum analysis
Simulation technique
Data formalizing
Prior analysis on alternative alternatives
Posterior analysis on the proceeding alternative
Tracking testing alternatives (reliability and cost)
Sensitivity analysis
Risk analysis
Database:
The parameters of the relevant costs on candidate alternatives:
The parameters of the prior distributions on the candidate alternatives by domain experts:
$E(\alpha), \sigma(\alpha), E(\eta), \sigma(\eta), \rho, E(\varpi), \sigma(\varpi)$..
The collected data of detected software errors

FIGURE 6. Computerized implementation architecture.

manager seeks the expertise of domain specialists, soliciting their insights on critical parameters, including the expected values and standard deviations for each alternative.

Following a comprehensive investigation and evaluation conducted by both software engineers and domain experts, detailed information about the three candidate testing alternatives has been compiled and is presented in Table 3. This data will serve as a crucial resource for the manager in making an informed decision that balances cost and reliability, ultimately ensuring the software's readiness for a successful market release.

TABLE 3. Information of debugging costs and parameter estimations for the three testing alternatives.

Alternative 1	Alternative 2	Alternative 3
Low-Intensity Testing Resource	Medium-Intensity Testing Resource	High-Intensity Testing Resource
$\hat{a} = 1980$, $q_v = \{0.45, 0.30, 0.25\}$, $R_m = 0.9$, RC=\$280, $\kappa_1 = 1.15$, ω_0=1700, $\omega_1 = 1.6$, ω_2=1.85		
$E[\alpha] = 0.3$, $\sigma[\alpha] = 0.1$, $E[\eta] = 0.25$, $\sigma[\eta] = 0.28$, $\rho_{\alpha,\beta} = 0.15$, $E[\varpi] = 0.12$, $\sigma[\varpi] = 0.08$, $SC_p = \$5500$, $GC_p = \$14500$, $EC_p = \$60$ $E[t_r \mid \psi_v, \tau_v] = 1.7$, $EC_p^v = \{47,54,28\}$	$E[\alpha] = 0.35$, $\sigma[\alpha] = 0.1$, $E[\eta] = 0.4$, $\sigma[\eta] = 0.16$, $\rho_{\alpha,\beta} = 0.2$, $E[\varpi] = 0.08$, $\sigma[\varpi] = 0.04$, $SC_p = \$6500$, $GC_p = \$14500$, $EC_p = \$60$, $E[t_r \mid \psi_v, \tau_v] = 2.5$, $EC_p^v = \{55,52,58\}$	$E[\alpha] = 0.4$, $\sigma[\alpha] = 0.2$, $E[\eta] = 0.45$, $\sigma[\eta] = 0.17$, $\rho_{\alpha,\beta} = 0.3$, $E[\varpi] = 0.09$, $\sigma[\varpi] = 0.06$, $SC_p = \$8500$, $GC_p = \$14500$, $EC_p = \$60$, $E[t_r \mid \psi_v, \tau_v] = 3.9$, $EC_p^v = \{62,72,83\}$

The three alternatives under consideration represent distinct staffing arrangements, each with its own implications for the efficiency and cost-effectiveness of the software testing process. The inclusion of more experienced team members typically enhances the efficiency of the testing process; however, this advantage is accompanied by higher salaries, which must be carefully weighed against the associated benefits. Therefore, the manager must assess which staffing arrangement provides the optimal balance between efficiency and cost.

Upon thorough analysis of the three alternatives, it is evident that Alternative 2 emerges as the most viable option. In this scenario, the manager should plan for a testing period of 5.6 weeks, resulting in a total cost of \$539,075. However, if the goal is to enhance software reliability to exceed 0.95, it will be necessary to extend the testing duration to 6.2 weeks, which will subsequently increase the expected cost to approximately \$526,167. Detailed comparisons of the three alternatives, including their associated costs, reliability levels, and error detection rates, are presented in Table 4 and illustrated in Figure 7.

Alternative 3, while capable of reducing the testing time to 4.8 weeks and achieving the minimum required reliability of 0.9, incurs a significantly higher total cost of \$603,249. This cost difference is substantial when compared to Alternative 2, even though Alternative 3 offers a slightly higher detection rate. However, the mean value of errors detected in Alternative 2 is comparable to that of Alternative 3, suggesting that increasing testing resources and personnel may not result in a proportional improvement in debugging effectiveness. Therefore, the manager must carefully weigh the trade-off between the additional costs and the marginal gains in debugging efficiency. In conclusion, the analysis indicates that merely increasing testing resources and staffing does not ensure a significant enhancement in debugging outcomes. The manager must carefully balance testing costs with software quality to make the most informed decision.

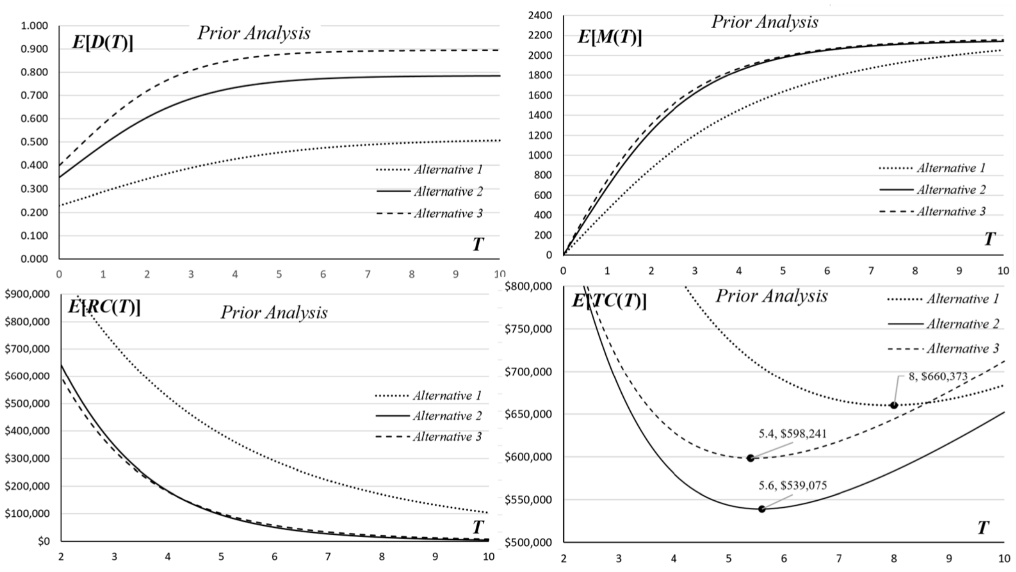

FIGURE 7. The comparisons among alternatives 1, 2, and 3 on error detection rate, Mean value, Risk cost, and Expected total cost.

TABLE 4. Detection rate, Mean value, Expected total cost, Risk cost and reliability of alternatives 1, 2, and 3.

Time T	$E[D_p(T)]$			$E[M_p(T)]$			$E[TC_p(T)]$			$E[RC_p(T)]$			$E[R_p(x\|T)]$		
	1	*2*	*3*	*1*	*2*	*3*	*1*	*2*	*3*	*1*	*2*	*3*	*1*	*2*	*3*
4.0	0.428	0.734	0.854	1453	1847	1872	$820,583	$580,276	$628,803	$524,843	$183,156	$179,706	0.7246	0.8089	0.8405
4.2	0.434	0.740	0.860	1494	1879	1902	$800,490	$568,963	$619,847	$493,650	$161,094	$159,687	0.7386	0.8296	0.8610
4.4	0.440	0.746	0.865	1533	1908	1929	$782,250	$559,862	$612,744	$464,553	$141,704	$141,991	0.7525	0.8486	0.8792
4.6	0.445	0.751	0.869	1570	1934	1952	$765,738	$552,716	$607,275	$437,407	$124,672	$126,343	0.7661	0.8659	0.8954
4.8	0.451	0.755	0.873	1604	1957	1974	$750,854	$547,293	$603,249	$412,098	$109,718	$112,499	0.7795	0.8815	0.9097
5	0.455	0.759	0.876	1636	1977	1993	$737,449	$543,388	$600,496	$388,450	$96,589	$100,244	0.7926	0.8956	0.9221
5.2	0.460	0.763	0.879	1667	1996	2010	$725,436	$540,819	$598,870	$366,365	$85,064	$89,390	0.8053	0.9082	0.9330
5.4	0.464	0.766	0.881	1696	2012	2026	$714,719	$539,428	**$598,241**	$345,731	$74,947	$79,769	0.8175	0.9195	**0.9425**
5.6	0.468	0.768	0.883	1723	2026	2040	$705,209	**$539,075**	$598,495	$326,442	$66,065	$71,236	0.8293	**0.9295**	0.9507
5.8	0.472	0.770	0.885	1748	2039	2053	$696,820	$539,636	$599,535	$308,398	$58,265	$63,662	0.8406	0.9383	0.9578
6	0.475	0.772	0.886	1772	2051	2064	$689,479	$541,005	$601,272	$291,517	$51,413	$56,936	0.8513	0.9461	0.9639
6.2	0.478	0.774	0.887	1795	2061	2074	$683,113	$543,088	$603,632	$275,711	$45,392	$50,956	0.8616	0.9530	0.9692
6.4	0.481	0.775	0.889	1816	2071	2084	$677,657	$545,803	$606,549	$260,904	$40,098	$45,638	0.8713	0.9590	0.9737
6.6	0.484	0.777	0.889	1836	2079	2092	$673,050	$549,078	$609,964	$247,025	$35,443	$40,903	0.8805	0.9643	0.9775
6.8	0.486	0.778	0.890	1855	2086	2100	$669,235	$552,852	$613,826	$234,006	$31,346	$36,685	0.8892	0.9690	0.9809
7.0	0.489	0.779	0.891	1873	2093	2106	$666,162	$557,070	$618,092	$221,788	$27,739	$32,924	0.8974	0.9730	0.9837
7.2	0.491	0.779	0.891	1890	2099	2113	$663,782	$561,685	$622,721	$210,316	$24,562	$29,568	0.9051	0.9766	0.9861
7.4	0.493	0.780	0.892	1906	2104	2118	$662,052	$566,656	$627,680	$199,537	$21,761	$26,571	0.9123	0.9796	0.9882
7.6	0.494	0.781	0.892	1921	2109	2123	$660,932	$571,947	$632,939	$189,403	$19,291	$23,894	0.9190	0.9823	0.9899
7.8	0.496	0.781	0.893	1936	2113	2128	$660,384	$577,528	$638,471	$179,872	$17,111	$21,499	0.9252	0.9847	0.9914
8.0	0.498	0.782	0.893	1950	2117	2132	**$660,373**	$583,370	$644,253	$170,900	$15,186	$19,356	**0.9311**	0.9867	0.9927

Note: The bold text indicates the optimal solutions for various testing alternatives.

It is essential to acknowledge that the initial estimates provided by domain experts may not always be entirely accurate. If these early projections are not revised to account for potential discrepancies, the manager risks adopting a suboptimal testing strategy, ultimately missing opportunities for optimization. To facilitate more reliable decision-making, it is advisable to conduct a post hoc analysis, adjusting the original testing plan as new data emerges during the early stages of the testing process.

In one instance, the manager, uncertain about the results of the initial analysis, utilized data from the early testing phase ($\Delta^{(n)} = \{0.007, 0.011, 0.018, 0.023, 0.027, 0.036, 0.042, 0.053, 0.060, 0.068, 0.079, 0.094\}$) to conduct a posterior analysis. The results indicated that the original projections were overly optimistic, as the actual debugging efficiency fell short of expectations. Consequently, the manager extended the testing period from 5.6 weeks to 6 weeks, resulting in an increase in the total expected cost to $554,734. Figure 8 visually illustrates the differences between the prior and posterior

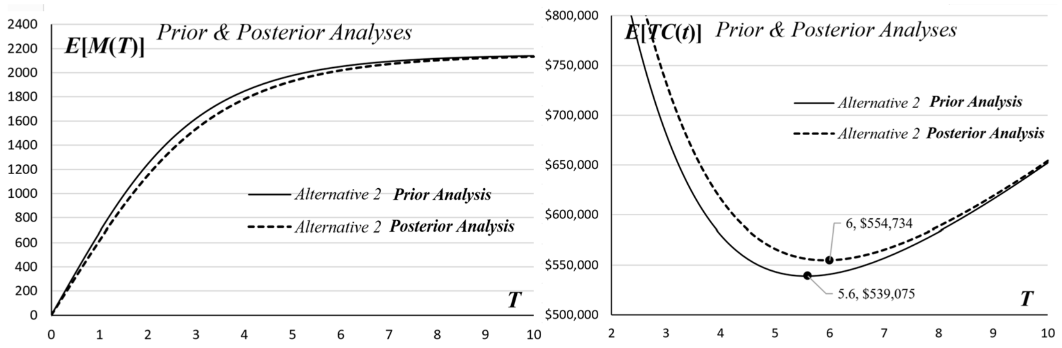

FIGURE 8. The comparisons between the prior and posterior analyses on expected total cost and expected mean value.

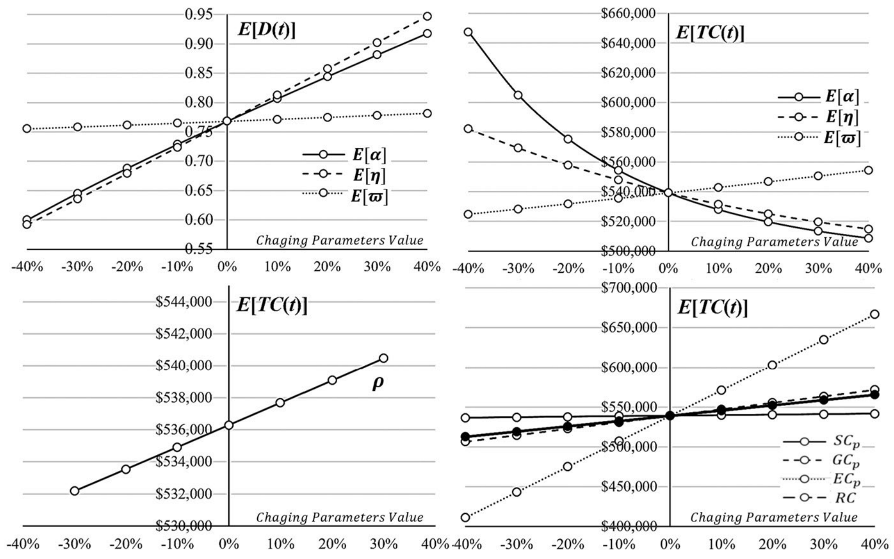

FIGURE 9. The impacts of model's parameters and related costs on error detection rate and expected total cost.

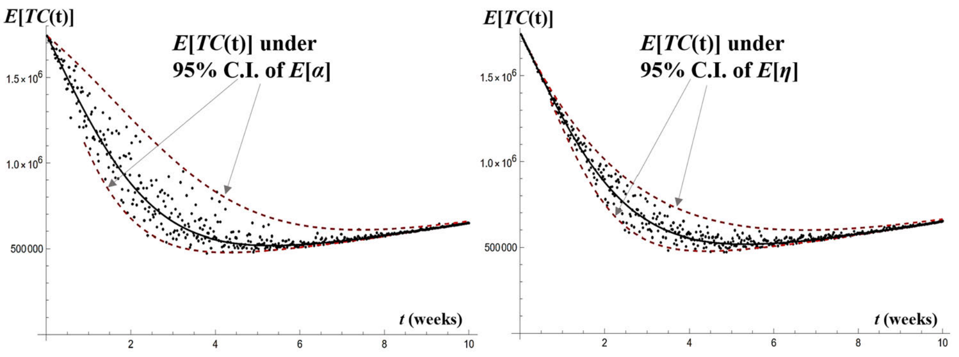

FIGURE 10. Scatter plots and statistical confidence interval of expected total cost under standard deviations α and η.

analyses, highlighting the changes in total testing costs and the number of errors detected.

Moreover, given the possibility that domain experts might misestimate key parameters—such as $E[\alpha]$, $E[\eta]$, $E[\varpi]$, and $\rho_{\alpha,\beta}$—during the prior analysis, the manager conducted sensitivity analyses to understand how uncertainties in these parameters could affect the results. The outcomes, illustrated in Figure 10, demonstrate the impact of varying the model parameters on the mean error detection rate, total testing cost, and overall detection rate. The analysis revealed that changes in $E[\alpha]$ and $E[\eta]$ have the most significant effect on total costs and the error detection rate, emphasizing the necessity for experts to carefully evaluate these parameters to avoid costly errors and misguided strategies.

From a strategic perspective, enhancing the values of $E[\alpha]$ and $E[\eta]$ through initiatives such as on-the-job training or the recruitment of more experienced personnel could improve debugging efficiency. However, this approach would also lead to an increase in overall testing costs. Therefore, the manager must carefully evaluate the benefits of improving factors α and η against the additional expenses incurred. Furthermore, the correlation between factors α and η is crucial for enhancing debugging performance. As illustrated in the upper-left section of Figure 10, increasing the correlation coefficient $\rho_{\alpha,\beta}$ significantly increases the number of errors detected. In contrast, the influence of the negligent factor ϖ is relatively minor, as variations in ϖ have little effect on the detection rate. When considering factors related to testing costs, it becomes evident that the cost per unit time for correcting an error (EC_p) is the most sensitive parameter. Therefore, it is crucial for the manager to estimate this value with a high degree of accuracy. The standard deviations of other parameters, such as $\sigma[\alpha]$ and $\sigma[\eta]$, also influence the overall cost estimation. To address the uncertainty in these estimates, a Monte Carlo simulation was employed in this study to model the distribution of total testing costs under varying $\sigma[\alpha]$ and $\sigma[\eta]$ values. This simulation provides the manager with a more comprehensive understanding of the risks associated with parameter uncertainty. Figure 10 presents scatter plots of total testing costs ($TC_p(T)$) under 95% confidence intervals (C.I.), illustrating that the range of uncertainty is greatest during the middle stages of testing. This increased uncertainty poses a challenge for the manager; however, by anticipating a wide range of possible outcomes, the manager can make more informed decisions and preparations.

V. CONCLUSION

The primary objective of software reliability research is to develop strategies that effectively identify and address software bugs, thereby improving overall system reliability. While traditional methods for estimating model parameters often depend on extensive historical data, this approach becomes impractical when such data is unavailable. In such cases, alternative approaches like Bayesian analysis offer a powerful and reliable solution. This research introduces a software reliability growth model for imperfect debugging, which accounts for both learning effects and negligence factors. By incorporating these elements, the model simplifies the estimation process for domain experts while improving its usability and relevance. The application of Bayesian analysis further enhances the model's practicality, especially in real-world situations where data is scarce or incomplete. Our findings underscore the importance of the parameters α and β, which indicate the effectiveness of the software testing process. Enhancing these values through additional training or by hiring experienced personnel can improve testing efficiency; however, it also increases costs. Striking a balance between these competing factors is vital for developers. Moreover, uncertainties in cost estimates are primarily influenced by the error-detection process, making it imperative for managers to consider this uncertainty in order to effectively mitigate risks.

Additionally, during the testing process, discrepancies may arise between expected and actual results. In such cases, conducting a post-analysis and adjusting the testing plan accordingly ensures alignment with real-world conditions, thereby enhancing both reliability and cost accuracy.

Future research could benefit from three promising directions. First, integrating a Non-Homogeneous Poisson Process (NHPP) with covariates would enable more accurate reliability predictions by considering additional factors such as testing effort and resource allocation. Second, addressing the time delays in error correction within reliability models would provide a more realistic representation of testing conditions. Lastly, incorporating change-point models to account for shifts in debugging conditions would enhance adaptability, particularly in dynamic testing environments, thereby making the model more flexible and effective across various scenarios.

REFERENCES

[1] H. Pham, L. Nordmann, and Z. Zhang, "A general imperfect-software-debugging model with S-shaped fault-detection rate," *IEEE Trans. Rel.*, vol. 48, no. 2, pp. 169–175, Jun. 1999.

[2] N. D. Singpurwalla and S. P. Wilson, "Statistical analysis of software failure data," in *Statistical Methods in Software Engineering*. New York, NY, USA: Springer, 1999, pp. 101–167.

[3] X. Zhang and H. Pham, "Software field failure rate prediction before software deployment," *J. Syst. Softw.*, vol. 79, no. 3, pp. 291–300, Mar. 2006.

[4] P. K. Kapur, D. Gupta, A. Gupta, and P. C. Jha, "Effect of introduction of faults and imperfect debugging on release time," *Ratio Math.*, vol. 18, pp. 62–90, Dec. 2008.

[5] R. Peng, Y. F. Li, W. J. Zhang, and Q. P. Hu, "Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction," *Rel. Eng. Syst. Saf.*, vol. 126, pp. 37–43, Jun. 2014.

[6] J. Wang, Z. Wu, Y. Shu, and Z. Zhang, "An imperfect software debugging model considering log-logistic distribution fault content function," *J. Syst. Softw.*, vol. 100, pp. 167–181, Feb. 2015.

[7] J. Wang and Z. Wu, "Study of the nonlinear imperfect software debugging model," *Rel. Eng. Syst. Saf.*, vol. 153, pp. 180–192, Sep. 2016.

[8] I. Saraf and J. Iqbal, "Generalized multi-release modelling of software reliability growth models from the perspective of two types of imperfect debugging and change point," *Qual. Rel. Eng. Int.*, vol. 35, no. 7, pp. 2358–2370, Nov. 2019.

[9] V. Verma, S. Anand, and A. G. Aggarwal, "Software warranty cost optimization under imperfect debugging," *Int. J. Quality Rel. Manage.*, vol. 37, no. 9/10, pp. 1233–1257, Dec. 2020.

[10] T. Li, X. Si, Z. Yang, H. Pei, and Y. Ma, "NHPP testability growth model considering testability growth effort, rectifying delay, and imperfect correction," *IEEE Access*, vol. 8, pp. 9072–9083, 2020.

[11] R. Bibyan, S. Anand, A. G. Aggarwal, and A. Tandon, "Multi-release testing coverage-based SRGM considering error generation and change-point incorporating the random effect," *Int. J. Syst. Assurance Eng. Manage.*, vol. 14, no. 5, pp. 1877–1887, Oct. 2023.

[12] C.-W. Yeh and C.-C. Fang, ''Software testing and release decision at different statistical confidence levels with consideration of debuggers' learning and negligent factors,'' *Int. J. Ind. Eng. Comput.*, vol. 15, no. 1, pp. 105–126, 2024.

[13] C. G. Bai, Q. P. Hu, M. Xie, and S. H. Ng, ''Software failure prediction based on a Markov Bayesian network model,'' *J. Syst. Softw.*, vol. 74, no. 3, pp. 275–282, Feb. 2005.

[14] A. Pievatolo, F. Ruggeri, and R. Soyer, ''A Bayesian hidden Markov model for imperfect debugging,'' *Rel. Eng. Syst. Saf.*, vol. 103, pp. 11–21, Jul. 2012.

[15] T. Aktekin and T. Caglar, ''Imperfect debugging in software reliability: A Bayesian approach,'' *Eur. J. Oper. Res.*, vol. 227, no. 1, pp. 112–121, May 2013.

[16] X. Zhao, B. Littlewood, A. Povyakalo, L. Strigini, and D. Wright, ''Conservative claims for the probability of perfection of a software-based system using operational experience of previous similar systems,'' *Rel. Eng. Syst. Saf.*, vol. 175, pp. 265–282, Jul. 2018.

[17] H. Wang, H. Fei, Q. Yu, W. Zhao, J. Yan, and T. Hong, ''A motifs-based maximum entropy Markov model for realtime reliability prediction in system of systems,'' *J. Syst. Softw.*, vol. 151, pp. 180–193, May 2019.

[18] D. R. Insua, F. Ruggeri, R. Soyer, and S. Wilson, ''Advances in Bayesian decision making in reliability,'' *Eur. J. Oper. Res.*, vol. 282, no. 1, pp. 1–18, Apr. 2020.

[19] C. A. Furia, R. Torkar, and R. Feldt, ''Applying Bayesian analysis guidelines to empirical software engineering data: The case of programming languages and code quality,'' *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 3, pp. 1–38, Jul. 2022.

[20] Q. Tian, C.-W. Yeh, and C.-C. Fang, ''Bayesian decision making of an imperfect debugging software reliability growth model with consideration of Debuggers' learning and negligence factors,'' *Mathematics*, vol. 10, no. 10, p. 1689, May 2022.

[21] S. Oveisi, A. Moeini, S. Mirzaei, and M. A. Farsi, ''Software reliability prediction: A machine learning and approximation Bayesian inference approach,'' *Qual. Rel. Eng. Int.*, vol. 40, no. 7, pp. 4004–4037, Nov. 2024, doi: 10.1002/qre.3616.

[22] W. R. Schucany, W. C. Parr, and J. E. Boyer, ''Correlation structure in Farlie-Gumbel-Morgenstern distributions,'' *Biometrika*, vol. 65, no. 3, p. 650, Dec. 1978.

[23] P. A. P. Moran, ''Statistical inference with bivariate gamma distributions,'' *Biometrika*, vol. 56, no. 3, p. 627, Dec. 1969.