Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 9276093, 13 pages
http://dx.doi.org/10.1155/2016/9276093

Research Article

A New Software Reliability Growth Model: Multigeneration Faults and a Power-Law Testing-Effort Function

Fan Li[1] and Ze-Long Yi[2]

[1]*China Center for Special Economic Zone Research, Shenzhen University, Nanhai Ave 3688, Shenzhen, Guangdong 518060, China*
[2]*Department of Transportation Economics and Logistics Management, College of Economics, Shenzhen University, Nanhai Ave 3688, Shenzhen, Guangdong 518060, China*

Correspondence should be addressed to Ze-Long Yi; yizl@szu.edu.cn

Received 20 August 2016; Accepted 13 October 2016

Academic Editor: Khaled Bahlali

Software reliability growth models (SRGMs) based on a nonhomogeneous Poisson process (NHPP) are widely used to describe the stochastic failure behavior and assess the reliability of software systems. For these models, the testing-effort effect and the fault interdependency play significant roles. Considering a power-law function of testing effort and the interdependency of multigeneration faults, we propose a modified SRGM to reconsider the reliability of open source software (OSS) systems and then to validate the model's performance using several real-world data. Our empirical experiments show that the model well fits the failure data and presents a high-level prediction capability. We also formally examine the optimal policy of software release, considering both the testing cost and the reliability requirement. By conducting sensitivity analysis, we find that if the testing-effort effect or the fault interdependency was ignored, the best time to release software would be seriously delayed and more resources would be misplaced in testing the software.

1. Introduction

Software systems are thought to run successfully when they perform without any failure during a certain period of time. However, in practice, failures are frequently observed and a perfect operational process cannot always be guaranteed due to a variety of reasons. Software reliability, which is defined as the probability of failure-free operation of a software program for a specified time interval, is an essential consideration in the processes of software development and a critical factor in measurement of software quality. In fact, accurately modeling the software reliability growth process and predicting its possible trend are of high values for software developers. For these reasons, a bunch of software reliability growth models (SRGMs), most of which are based on a nonhomogeneous Poisson process (NHPP), have been constructed [1–5]. Unlike closed-source software, open source software's (OSS) knowledge base is not bounded to a particular organization. As a result, reliability measurement is important and complicated for OSS, such as Linux operating systems, Mozilla browser, and Apache web server. Due to their unique properties, a parallel of SRGMs has been purposefully developed to quantify the stochastic failure behavior of OSS [6, 7].

One key factor of software reliability growth modeling is the testing-effort effect [8–14]. Huang and Kuo [15] incorporated a logistic testing-effort function into software reliability modeling and showed that it provides powerful prediction capability based on real failure data. Huang [16] considered a generalized logistic testing-effort function and change-point parameters in software reliability modeling and some practical data were employed to fit the new models. Ahmad et al. [17] incorporated the exponentiated Weibull testing-effort function into inflection S-shaped software reliability growth models. Peng et al. [18] established a flexible and general software reliability model considering both testing-effort function and imperfect debugging.

In these models, assuming independence among successive software runs may not be appropriate [19–21]. For instance, Goševa-Popstojanova and Trivedi [22] introduced a software reliability model based on Markov renewal processes that covers interdependence among successive software runs. The Compound-Poisson software reliability model presented

by Sahinoglu [23] considered multiple failures that occur simultaneously. Singh et al. [24] proposed new SRGMs that assume the presence of two types of faults in the software: leading and dependent. Leading faults can be removed when a failure is observed, but dependent faults are masked by leading faults and can be removed only after the corresponding leading faults are removed.

We also notice that a bundle of prior papers have examined the problems of optimal release time under different criteria. One stream studies this issue by minimizing the total cost subject to a constraint on software reliability. For example, Huang and Lin [25] provided a series of theoretical findings regarding when a software version should be released and when fault dependency and debugging time lag are considered. In addition, Ahmad et al. [26] analyzed the optimal release timing in an SRGM with Burr-type X testing-effort functions. Another stream adopts the multiattribute utility theory (MAUT) to decide the release time. For instance, Singh et al. [27] built an SRGM based on the Yamada two-stage model and focused on two different attributes to investigate the release time. Pachauri et al. [28] studied an SRGM incorporating a generalized modified Weibull testing-effort function in an imperfect debugging environment and the optimal release policy is examined using both genetic algorithm (GA) and MAUT.

In this work, we propose a new model that incorporates a power-law testing-effort function and three generations of interdependent errors. Specifically, the detection process for the first-generation errors is independent, and the detection rate is proportional to the instantaneous testing-effort expenditure and the mean of remaining first-generation errors in the system. By contrast, the detection process of the second-generation errors depends on the first generation and the detection rate is proportional to not only the instantaneous testing-effort expenditure and the mean of remaining second-generation errors but also the ratio of removed first-generation errors. Similarly, the detection process of the third-generation errors relies on the second-generation errors and the detection rate is proportional to the instantaneous testing-effort expenditure, the mean of remaining third-generation errors, and the ratio of removed second-generation errors.

Model parameters are estimated by the nonlinear least square estimation method, which is realized by minimizing the sum of squares of the deviations between estimated values and true observations. Numerical experiments are carried out based on three versions of Apache. The estimation results show that the proposed model well fits the real observations and performs better than the traditional Goel-Okumoto model and the Yamada delayed S-shaped model. Comprehensive comparisons in terms of prediction capabilities among various models are conducted to reveal that our model can predict the error occurrence process more accurately than the benchmarks. We also analyze the optimal release policy and the minimal cost for the software development managers, based on which insightful suggestions are provided. Besides, we conduct sensitivity analysis and find that resources would be misplaced if either the testing-effort effect or the error interdependency is overlooked by the software testing team. Unlike the benchmarks, our results suggest that the strategy of an earlier release time can substantially reduce the cost.

2. Software Reliability Growth Modeling

2.1. Basic Assumptions. To describe the stochastic failure behavior of software systems, we define $\{N(t), t \geq 0\}$ as a counting process that represents the cumulative number of failures by time t. We use the mean value function (MVF) $m(t)$ to denote the expected cumulative number of detected failures in the time period $(0, t]$. That is,

$$m(t) = E[N(t)] = \int_0^t \lambda(t')\, dt', \tag{1}$$

where $E[\cdot]$ means the expectation operator and $\lambda(t')$ is called failure intensity function which indicates the instantaneous fault-detection rate at time t'. Specifically, for an NHPP-based SRGM, it is assumed that $N(t)$ follows a Poisson distribution with a mean value function $m(t)$. That is, the model can be mathematically characterized as

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

where $\text{Poim}(n, m(t))$ is the Poisson probability mass function (pmf) with mean $m(t)$ [29, 30]. Generally, one can obtain different NHPP models by taking different mean value functions.

In this paper, to characterize the stochastic behavior of the software failure process and obtain the expression for the mean value function, we make the following assumptions:

(i) The software system is subject to failures at random times caused by errors remaining in the system.

(ii) The error detection phenomenon in software testing is modeled by a nonhomogeneous Poisson process.

(iii) Each time a failure occurs, the error causing it is immediately removed and no new errors are introduced.

(iv) The faults existing in the software are of three generations (G1, G2, and G3) and each generation is modeled by a different growth curve.

(v) The mean number of G1 errors detected in the time interval $(t, t + \Delta t)$ is proportional to the instantaneous testing-effort expenditure and the mean number of remaining G1 errors in the system.

(vi) The mean number of G2 errors detected in the time interval $(t, t + \Delta t)$ is proportional to the instantaneous testing-effort expenditure, the mean number of remaining G2 errors in the system, and the ratio of removed G1 errors.

(vii) The mean number of G3 errors detected in the time interval $(t, t + \Delta t)$ is proportional to the instantaneous testing-effort expenditure, the mean number of remaining G3 errors in the system, and the ratio of removed G2 errors.

(viii) Fault-detection/removal rate is a power-law function of testing time for all three generations of errors.

The first four assumptions in the above list are the building blocks in most prior software reliability growth models and the others are specific in our model. To facilitate our later analysis, we summarize the notations used throughout this paper in Notations section.

2.2. Proposed Software Reliability Growth Model. In general, an implemented software system is tested to detect and correct software errors in the development process. During the testing phase, software errors remaining in the system are discovered and corrected by test personnel. Based on the aforementioned assumptions, the detection rate must be proportional to the instantaneous testing-effort expenditure. Particularly, unlike traditional closed-source software, in open source software, instructions are executed by various concurrent users and each release of open source software can attract an increasing number of volunteers in the software development and/or code modification procedure. Hence, it is reasonable to assume the detection/correction rate as a power-law function of testing time [19, 24]. That is, the instantaneous testing-effort expenditure at time t, $w(t)$, takes the following form:

$$w(t) = bt^k, \tag{3}$$

where b and k are parameters to be estimated by real-world data sets. As such, we can quickly obtain the cumulative amount of testing-effort expenditures by time t; that is,

$$W(t) = \int_0^t w(t')\,dt' = \frac{bt^{k+1}}{k+1}. \tag{4}$$

As stated above, it is essential to consider fault interdependency when modeling software reliability growth processes. Specifically, in this work we propose three generations of interdependent errors. That is, the detection of the second-generation errors relies on the first generation, and the detection of the third-generation errors depends on the second generation. Moreover, the second-generation errors are detectable only if the first-generation errors have been removed, and the third-generation errors are detectable only if the second-generation errors have been removed. The flow chart is shown in Figure 1 to illustrate the error detecting process in a software system.

Next, we sequentially examine the mean value function for each generation of errors:

(i) For the first-generation errors in the software system, the number of detected errors is proportional to the number of remaining errors and the instantaneous testing-effort expenditure. Thus, the mean value function for G1 errors satisfies the following differential equation:

$$\frac{dm_1(t)}{dt} = w(t)\left[a_1 - m_1(t)\right], \tag{5}$$

where a_1 is the initial content of G1 errors in the system and $m_1(t)$ represents the expected number of G1 errors detected in time $(0, t]$. Specifically, $m_1(t)$ is assumed to be a bounded nondecreasing function of t with the initial condition $m_1(0) = 0$. Solving the above differential equation yields

$$m_1(t) = a_1\left(1 - e^{-bt^{k+1}/(k+1)}\right). \tag{6}$$

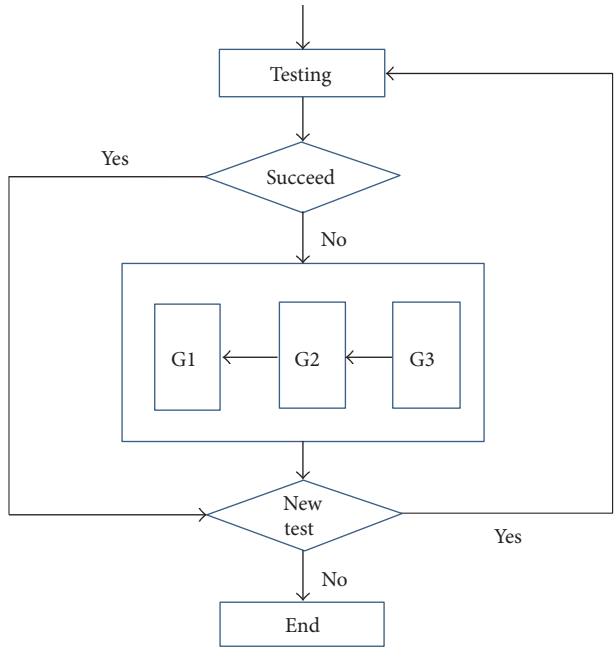

Figure 1: A flow chart for the error testing process when multigeneration faults exist in a software system.

(ii) The mean value function for G2 errors satisfies the following differential equation:

$$\frac{dm_2(t)}{dt} = w(t)\left[a_2 - m_2(t)\right]\frac{m_1(t)}{a_1}, \tag{7}$$

where a_2 is the initial content of G2 errors in the system and $m_2(t)$ stands for the expected number of G2 errors detected in time $(0, t]$. Specifically, $m_2(t)$ is also assumed to be a bounded nondecreasing function of t with the initial condition $m_2(0) = 0$. Solving the above differential equation yields

$$m_2(t) = a_2\left(1 - e^{-bt^{k+1}/(k+1) - e^{-bt^{k+1}/(k+1)} + 1}\right). \tag{8}$$

(iii) Similarly, the mean value function for G3 errors satisfies the following differential equation:

$$\frac{dm_3(t)}{dt} = w(t)\left[a_3 - m_3(t)\right]\frac{m_2(t)}{a_2}, \tag{9}$$

where a_3 is the initial content of G3 errors in the system and $m_3(t)$ means the expected number of G3 errors detected in time $(0, t]$. Specifically, $m_3(t)$ is assumed to be a bounded nondecreasing function of t with the initial condition $m_3(0) = 0$. Solving the above differential equation leads to

$$m_3(t) = a_3\left(1 - e^{-bt^{k+1}/(k+1) + e^{1 - e^{-bt^{k+1}/(k+1)}} - 1}\right). \tag{10}$$

After this, we formulate the overall mean value function $m(t)$ that consists of mean value functions for three generations of errors. Assume that $a = a_1 + a_2 + a_3$ is the expected total number of faults that will be eventually detected and p_1, p_2, and p_3 are corresponding ratios for each generation of errors at the beginning of testing. By noting that $a_1 = ap_1$, $a_2 = ap_2$, $a_3 = ap_3$, and $p_1 + p_2 + p_3 = 1$, we have

$$\begin{aligned} m(t) &= m_1(t) + m_2(t) + m_3(t) = a_1\left(1 - e^{-bt^{k+1}/(k+1)}\right) + a_2\left(1 - e^{-bt^{k+1}/(k+1) - e^{-bt^{k+1}/(k+1)} + 1}\right) \\ &\quad + a_3\left(1 - e^{-bt^{k+1}/(k+1) + e^{1-e^{-bt^{k+1}/(k+1)}} - 1}\right) = a\left(1 - p_1 e^{-bt^{k+1}/(k+1)} - p_2 e^{-bt^{k+1}/(k+1) - e^{-bt^{k+1}/(k+1)} + 1}\right. \\ &\quad \left. - p_3 e^{-bt^{k+1}/(k+1) + e^{1-e^{-bt^{k+1}/(k+1)}} - 1}\right) = a\left(1 - p_1 e^{-bt^{k+1}/(k+1)} - p_2 e^{-bt^{k+1}/(k+1) - e^{-bt^{k+1}/(k+1)} + 1}\right. \\ &\quad \left. - (1 - p_1 - p_2) e^{-bt^{k+1}/(k+1) + e^{1-e^{-bt^{k+1}/(k+1)}} - 1}\right). \end{aligned} \tag{11}$$

As such, $m(t)$ is a bounded nondecreasing function of t, with $m(0) = 0$ and $m(\infty) = a$.

Then, based on the aforesaid mean value function, the failure intensity function for the proposed model, with testing-effort, can be explicitly given as

$$\begin{aligned} \lambda(t) &= \frac{\mathrm{d}m(t)}{\mathrm{d}t} = w(t)\left[a_1 e^{-bt^{k+1}/(k+1)} + a_2 e^{-bt^{k+1}/(k+1) - e^{-bt^{k+1}/(k+1)} + 1}\left(1 - e^{-bt^{k+1}/(k+1)}\right)\right. \\ &\quad + a_3 e^{-bt^{k+1}/(k+1) + e^{1-e^{-bt^{k+1}/(k+1)}} - 1}\left(1 - e^{-bt^{k+1}/(k+1) - e^{-bt^{k+1}/(k+1)} + 1}\right)\Bigg] = abt^k\left[p_1 e^{-bt^{k+1}/(k+1)} + p_2 e^{-bt^{k+1}/(k+1) - e^{-bt^{k+1}/(k+1)} + 1}\left(1 - e^{-bt^{k+1}/(k+1)}\right) + (1 - p_1 - p_2) e^{-bt^{k+1}/(k+1) + e^{1-e^{-bt^{k+1}/(k+1)}} - 1}\left(1 - e^{-bt^{k+1}/(k+1) - e^{-bt^{k+1}/(k+1)} + 1}\right)\right]. \end{aligned} \tag{12}$$

We further note that when $k = 0$, $p_1 = 1$, and $p_2 = p_3 = 0$, the original Goel-Okumoto model can be realized [31]; that is,

$$m(t) = a\left(1 - e^{-bt}\right), \tag{13}$$

which is one of our benchmarks for the comparative analysis. In this case, errors are assumed to be independent and the testing-effort function keeps constant, that is, b. Besides, the mean value function $m(t)$ is a concave function with respect to t.

Moreover, when $k = 0$, we have $w(t) = b$; namely, the instantaneous testing-effort expenditure at time t reduces to a constant. In this case, we have

$$\begin{aligned} m(t) &= a\left(1 - p_1 e^{-bt} - p_2 e^{-bt - e^{-bt} + 1}\right. \\ &\quad \left. - (1 - p_1 - p_2) e^{-bt + e^{1-e^{-bt}} - 1}\right), \end{aligned} \tag{14}$$

and we call it No-k model for brevity.

In addition, when $p_1 = 1$ and $p_2 = p_3 = 0$, the error interdependency is ignored and only one generation of errors exists in the software system. As a result, we have

$$m(t) = a\left(1 - e^{-bt^{k+1}/(k+1)}\right). \tag{15}$$

We call it Only-G1 model for exposition.

As another benchmark, hereby we introduce an S-shaped model, which incorporates an increasing detection rate function and well explains "learning effect" of the testing team. Essentially, the mean value function is stated as follows:

$$m(t) = a\left(1 - (1 + bt) e^{-bt}\right). \tag{16}$$

It is also called Yamada delayed S-shaped model [32, 33] in literature. For this model, the detection rate first increases and then decreases with t.

3. Estimation and Analysis

3.1. Data Set. To compare the proposed model against traditional models for reliability assessment, numerical examples are provided based on real-world data. Specifically, the well-known Apache project, the most used web server software that is developed and maintained by an open community of developers, is selected in this paper for illustration. This project has large and well-organized communities: a great number of developers have the right to update and change files freely. In particular, testing data about three versions of Apache, 2.0.35/2.0.36/2.0.39, are used. The failure data is presented in Table 1 [8], including the time and the number of errors detected for each version. Specifically, testing for Apache 2.0.35 ranges from day 1 to day 43 and in total 74 errors are found; failure data for Apache 2.0.36 are collected from day 1 to day 103 and 50 errors are detected; Apache 2.0.39 is tested over a 164-day period and in the end 58 errors are discovered by the testing team.

3.2. Estimation Method. The nonlinear least square estimation (NLSE) method is a widely used estimation technique, especially for small/medium sized data sets [34]. We will use this technique to estimate parameters in models raised in this paper. More specifically, estimation is done by minimizing the sum of squared residuals, which are the squares of the differences between estimated values and true observations.

Table 1: Detected faults (days from release and number of errors) for three versions of Apache, 2.0.35/2.0.36/2.0.39. Note that some failure data is not shown due to page limit. For example, as no faults are detected on the 42nd day in Apache 2.0.35, no failure data is shown on this day.

Apache 2.0.35		Apache 2.0.36		Apache 2.0.39	
Days from release	Detected errors	Days from release	Detected errors	Days from release	Detected errors
1	5	1	2	1	1
2	5	2	5	2	2
3	4	3	1	3	2
4	1	4	1	4	3
5	2	5	1	5	3
6	4	7	2	7	2
7	6	8	1	8	1
8	2	9	1	9	1
10	1	10	3	10	1
11	8	12	2	11	1
12	5	13	1	15	3
13	2	15	2	16	2
14	2	17	1	17	3
15	1	18	2	18	1
17	2	21	1	19	1
18	3	25	1	22	3
19	4	27	1	23	1
20	1	29	2	24	1
21	4	30	2	25	2
23	2	31	3	26	1
24	1	32	1	28	1
25	1	33	3	29	1
26	2	34	1	30	2
27	1	35	3	31	1
28	2	38	3	32	1
31	1	40	1	35	3
34	1	43	1	38	1
43	1	44	1	39	1
		103	1	42	1
				43	1
				49	3
				50	1
				51	1
				57	1
				66	1
				70	1
				81	1
				164	1

The objective function for the minimization problem is stated as follows:

$$\chi^2 = \sum_{j=1}^{K} \left(m\left(t_j\right) - m_j \right)^2, \tag{17}$$

where m_j is the jth observation, K is the total number of observations, t_j is the time in which the jth error occurs, and $m(t_j)$ is the corresponding estimated/theoretical value. Specifically, in this paper, $m(t_j)$ can be from (11), (13), (14), (15), or (16). They correspond to our proposed model, the Goel-Okumoto model, the No-k model, the Only-G1 model, and the Yamada delayed S-shaped, respectively.

Differentiating χ^2 with respect to each unknown parameter and setting the partial derivatives to zero, we can obtain the estimated values [34, 35]. Specifically, for our proposed model, we have

$$\frac{\partial \chi^2}{\partial a} = \frac{\partial \chi^2}{\partial b} = \frac{\partial \chi^2}{\partial p_1} = \frac{\partial \chi^2}{\partial p_2} = \frac{\partial \chi^2}{\partial k} = 0. \tag{18}$$

Using numerical methods to solve the set of simultaneous equations above yields the estimation for unknown parameters in our proposed model.

3.3. Metrics for Model Comparison. With estimated parameters, we can conduct comparative analysis among models based on the following metrics:

(1) Goodness of Fit (R^2): the value of R^2 determines how well the curve fits the real data [19, 36]. It is defined as follows:

$$R^2 = 1 - \frac{\sum_{j=1}^{K} \left(m\left(t_j\right) - m_j\right)^2}{\sum_{j=1}^{K} \left(m_j - \sum_{l=1}^{K} m_l/K\right)^2}. \tag{19}$$

It measures the percentage of the total variance about the mean accounted for the fitted curve. It ranges from 0 to 1. The larger R^2 is, the better the model fits data. In this work, the model is considered to provide a good fit if R^2 is above 0.95.

(2) Mean of Square Error (MSE): the value of MSE measures the average of squares of the differences between the predicted values and the actual values [9]. It is defined as follows:

$$\text{MSE} = \frac{1}{K} \sum_{j=1}^{K} \left(m\left(t_j\right) - m_j\right)^2. \tag{20}$$

Generally, the closer to zero the MSE value is, the better the model fits the data.

(3) Theil Statistic (TS): the value of TS indicates the average percentage deviation with respect to the actual failure data during the selected time period [37]. It is defined as follows:

$$\text{TS} = \sqrt{\frac{\sum_{j=1}^{K} \left(m\left(t_j\right) - m_j\right)^2}{\sum_{j=1}^{K} y_j^2}}. \tag{21}$$

A smaller TS value means a smaller deviation and a better prediction capability. In this work, the model is considered to provide a high-level prediction if the TS value is below 10%.

(4) Relative Error (RE): the value of RE measures the relative difference between the predicted value and the real observations [25, 38]. The definition can be stated as follows:

$$\text{RE} = \frac{m\left(t_q\right) - m_q}{m_q}. \tag{22}$$

Specifically, assuming that we have observed m_q failures by the end of test at time t_q, the failure data up to time t_e ($t_e \leq t_q$) will be used to estimate parameters in the models. Substituting the estimated parameters in the mean value function yields the estimation of the number of failures $m(t_q)$ by t_q. The estimation is then compared with the actual number m_q. Such procedure is repeated for various values of t_e. Positive values of error indicate overestimation and negative values indicate underestimation.

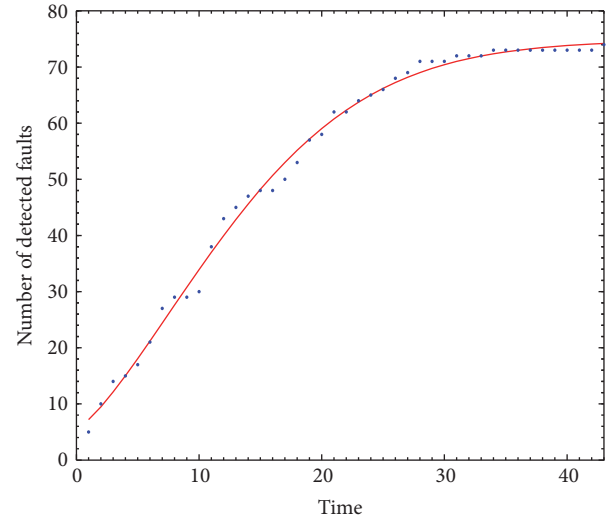

FIGURE 2: Predicted values and real-world observations for Apache 2.0.35. Solid line: predicted values; dot: observations.

3.4. Estimation Results and Model Comparison. The parameter estimation can help characterize properties of theoretical models, determine the number of errors that have already been detected/corrected in the testing process, and predict the number of errors that will be eventually encountered by users. Specifically, by minimizing the objective function in (17), we can obtain the estimated results for each model.

(i) For Apache 2.0.35, the estimated results are presented in Table 2, with $\widehat{a} = 74.8102$, $\widehat{b} = 0.0432$, $\widehat{p_1} = 0.5905$, $\widehat{p_2} = 0.2838$, and $\widehat{k} = 0.3554$ for our proposed model. Then, the ratio of G1 errors to all types of errors is approximately 59%, the ratio of G2 errors is around 28%, and the leftover are G3 errors. Note that the errors consist of not only independent errors (G1), although G2 and G3 account much less than G1. Through incorporating such interdependency among generations of errors, we are able to strengthen the understanding towards composition of software system. Besides, we find that the estimated value of k is obviously larger than 0. Hence, not considering the effect of testing effort in the analysis of the reliability growth process may lead to significantly biased results.

Illustrations for the theoretical values versus their observations are shown in Figure 2. We find that the theoretical values are highly consistent with the real-world observations. We can further examine the failure intensity function $\lambda(t)$. Figure 3 shows that the failure intensity function is a single-peaked curve; that is, the error detection rate first increases and then decreases with time. It in turn implies that our proposed model falls into the S-shaped category for the SRGMs [34].

We observe that our proposed model well fits the real-world software failure data. This can be seen from the high value of R^2, that is, 0.9954, which is very close to 1. By comparing with the Goel-Okumoto model, the Yamada delayed S-shaped model, the No-k model, and the Only-G1 model, R^2 values are the highest in our proposed model. This conclusion can also be drawn by comparing MSE values

Table 2: Optimal estimated values: Apache 2.0.35.

	$\widehat{a}$	$\widehat{b}$	$\widehat{p_1}$	$\widehat{p_2}$	$\widehat{p_3}$	$\widehat{k}$	R^2	MSE	TS
Goel-Okumoto model	84.6047	0.0564	—	—	—	—	0.9873	5.7624	0.0417
Yamada delayed S-shaped model	74.2741	0.1539	—	—	—	—	0.9830	7.7033	0.0482
No-k model	76.4154	0.1087	0.1593	0.7293	0.1114*	—	0.9947	2.3912	0.0269
Only-G1 model	77.1477	0.0438	—	—	—	0.2323	0.9925	3.4129	0.0321
Proposed model	74.8102	0.0432	0.5905	0.2838	0.1258*	0.3554	0.9954	2.0979	0.0252

*Calculated by $\widehat{p_3} = 1 - \widehat{p_1} - \widehat{p_2}$.

Table 3: Optimal estimated values: Apache 2.0.36.

	$\widehat{a}$	$\widehat{b}$	$\widehat{p_1}$	$\widehat{p_2}$	$\widehat{p_3}$	$\widehat{k}$	R^2	MSE	TS
Goel-Okumoto model	52.3158	0.0393	—	—	—	—	0.9526	8.8417	0.0707
Yamada delayed S-shaped model	49.8964	0.0896	—	—	—	—	0.9550	8.3927	0.0689
No-k model	50.4507	0.0669	0.0000	0.8065	0.1935*	—	0.9706	5.4946	0.0557
Only-G1 model	50.1765	0.0196	—	—	—	0.3207	0.9643	6.6635	0.0614
Proposed model	49.4122	0.0045	0.4031	0.2855	0.3114*	0.8967	0.9806	3.6163	0.0452

*Calculated by $\widehat{p_3} = 1 - \widehat{p_1} - \widehat{p_2}$.

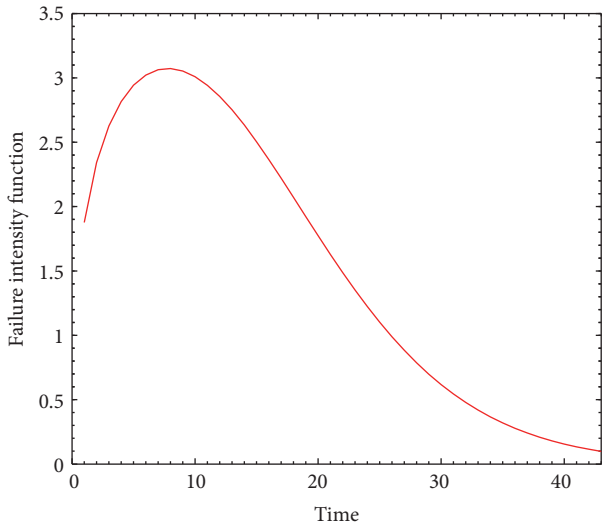

Figure 3: Failure intensity function with respect to time, with estimated values based on Apache 2.0.35.

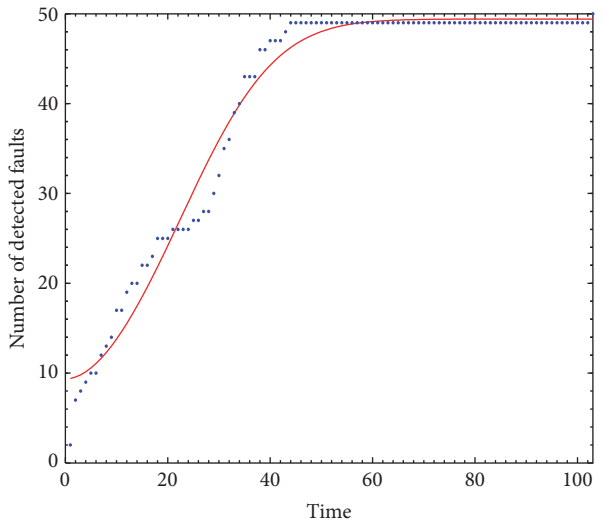

Figure 4: Predicted values and real-world observations for Apache 2.0.36. Solid line: predicted values; dot: observations.

among the three models; our proposed model provides the lowest MSE values (2.0979). From this aspect, the model in our paper best fits the observed data about Apache 2.0.35.

Furthermore, the prediction capability in our model is very high. The low TS value for Apache 2.0.35, 0.0252, is much lower than 10% and well confirms a high prediction capability for our model. Compared with the benchmarks, our model can provide significantly lower TS values and higher predication capacity to the failure of Apache system. Lastly, the relative error in prediction using this data set is computed and results are plotted in Figure 4. We observe that the relative error approaches 0 as t_e approaches t_q and the relative error curve is usually within ±5%.

(ii) For Apache 2.0.36, the estimated results are presented in Table 3, with $\widehat{a} = 49.4122$, $\widehat{b} = 0.0045$, $\widehat{p_1} = 0.4031$, $\widehat{p_2} = 0.2855$, and $\widehat{k} = 0.8967$ for our proposed model. Three generations of errors account for 40%, 29%, and 31%, respectively. Obviously, G2 and G3 significantly contribute to the error composition, which counts for 60% in total. We also find that the estimated value of k is much larger than 0 implying that the testing-effort effect is necessary.

Then, we can obtain theoretical values in our proposed model. Illustrations for the theoretical values versus their observations are presented in Figure 5, which shows that the theoretical values are consistent with the real-world data. In Figure 6, we present the failure intensity function $\lambda(t)$ based on the estimated values. Similar to Apache 2.3.35, the hump-shaped curve shows that this function possesses an increasing-then-decreasing trend.

We find that our proposed model also well fits the real-world software failure data for Apache 2.0.36. This can be seen from the high value of R^2, 0.9806. It is also much higher

Table 4: Optimal estimated values: Apache 2.0.39.

	$\widehat{a}$	$\widehat{b}$	$\widehat{p_1}$	$\widehat{p_2}$	$\widehat{p_3}$	$\widehat{k}$	R^2	MSE	TS
Goel-Okumoto model	58.3827	0.0367	—	—	—	—	0.9871	2.5670	0.0316
Yamada delayed S-shaped model	56.8974	0.0805	—	—	—	—	0.9879	2.4045	0.0306
No-k model	57.3279	0.0577	0.2414	0.6741	0.0845*	—	0.9969	0.6133	0.0154
Only-G1 model	57.3266	0.0204	—	—	—	0.2453	0.9957	0.8562	0.0182
Proposed model	57.1499	0.0300	0.5299	0.3820	0.0881*	0.1940	0.9973	0.5314	0.0144

*Calculated by $\widehat{p_3} = 1 - \widehat{p_1} - \widehat{p_2}$.

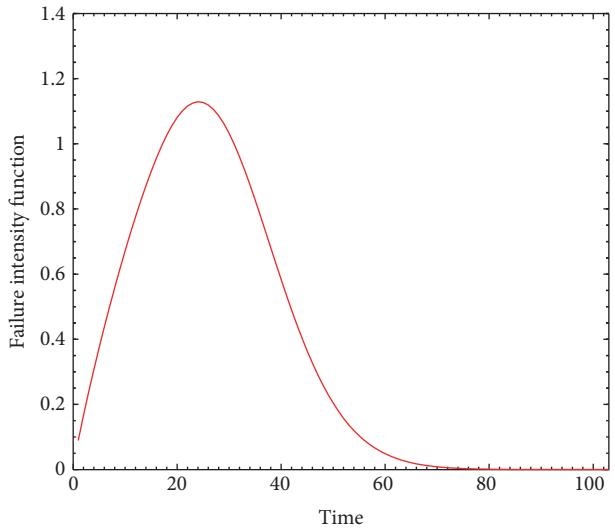

Figure 5: Failure intensity function with respect to time, with estimated values based on Apache 2.0.36.

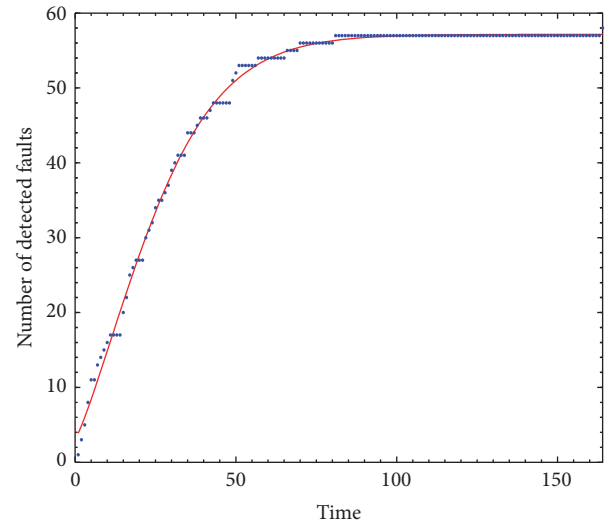

Figure 6: Predicted values and real-world observations for Apache 2.0.39. Solid line: predicted values; dot: observations.

than those in the Goel-Okumoto model, the Yamada delayed S-shaped model, the No-k model, and the Only-G1 model. By comparing MSE values among the models, our proposed model provides the lowest MSE value (3.6163). From this aspect, the proposed model best fits the observed data about Apache 2.0.36.

Furthermore, the TS value for Apache 2.0.36 is as low as 0.0452, which is much lower than 10% and well confirms a high prediction capability of our model. Compared with the benchmarks, our model shows a lower value of TS and a higher level of predication capability. Lastly, the relative error in prediction for this data set is computed, as shown in Figure 7. We observe that the relative error approaches 0 as t_e approaches t_q and the curve is usually within ±5%.

(iii) For Apache 2.0.39, the estimated results are presented in Table 4, with $\widehat{a} = 57.1499$, $\widehat{b} = 0.0300$, $\widehat{p_1} = 0.5299$, $\widehat{p_2} = 0.3820$, and $\widehat{k} = 0.1940$ for our proposed model. The ratios of three generations of errors become about 53%, 38%, and 9%, respectively. Then, the ratio for G1 errors is approximately 59%, the ratio for G2 errors is around 28%, and the leftover are G3 errors. We also find that the estimated value of k is significantly greater than 0. Thus, ignoring the effect of testing effort in the analysis of the software reliability growth curve would lead to significant bias.

Comparison between the fitted curve and the observed failure data is illustrated in Figure 8, which indicates that the theoretical values and the real-world observations are highly consistent. In Figure 9, we present the failure intensity function based on the estimated values. Obviously, it is also a single-peaked curve: the error detection rate first increases and then decreases with time.

The high value of R^2, 0.9937, suggests that our proposed model well fits the real-world software failure data for this version. By comparing with the Goel-Okumoto model, the Yamada delayed S-shaped model, the No-k model, and the Only-G1 model, R^2 values are the highest in our proposed model. This conclusion can also be drawn by comparing MSE values among the models; our proposed model provides the lowest MSE value (0.5314). Therefore, our proposed model best fits the observed data about Apache 2.0.39.

Furthermore, the TS value for Apache 2.0.39 is 0.0144. It is much lower than 10% and well confirms a high prediction capability for our model. Compared with the benchmarks, TS values in our model are significantly lower, and prediction capacity of the failure occurrence is also much greater. Lastly, the relative error in prediction for this data set is computed and results are plotted in Figure 10. We observe that the relative error approaches 0 as t_e approaches t_q and the error curve is usually within ±5%.

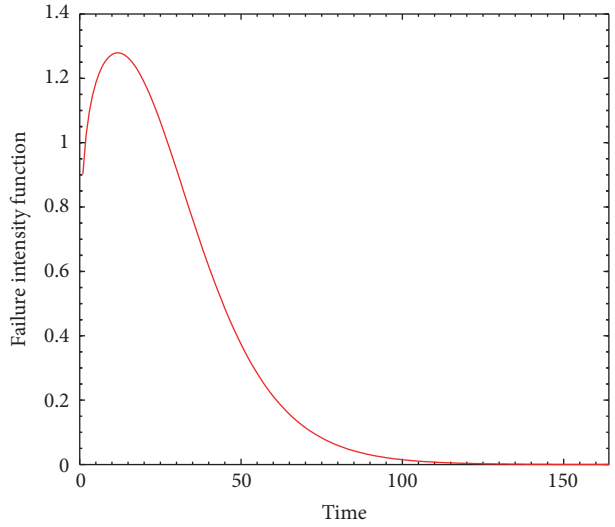

Figure 7: Failure intensity function with respect to time, with estimated values based on Apache 2.0.39.

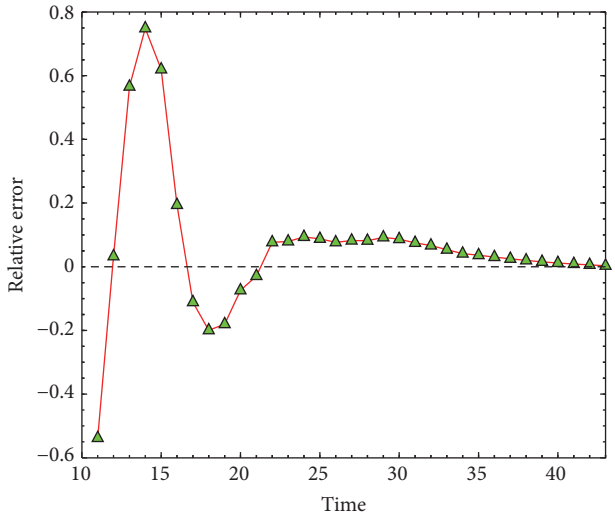

Figure 8: Relative error for Apache 2.0.35.

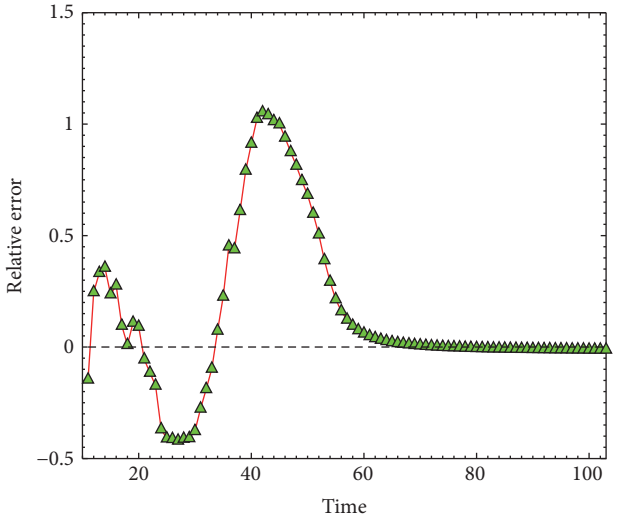

Figure 9: Relative error for Apache 2.0.36.

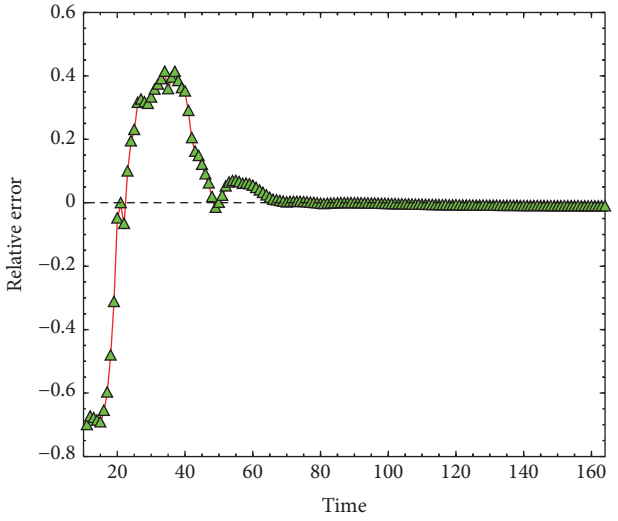

Figure 10: Relative error for Apache 2.0.39.

In summary, all of three tests using different versions of Apache failure data sets suggest that our proposed model well fits the real data and presents better performance than the benchmarks in error occurrence prediction. Besides, both the testing effort and the error interdependency effects play an important role in the parameter estimation.

4. Software Release Policy

The theoretical modeling can also determine whether the software is ready to release to users or how much more effort should be spent in further testing. In this section, we formally examine the software developer's best release time and deliver sensitivity analysis regarding how the test effort effect and error interdependency affect the optimal time and the cost.

4.1. Optimal Release Time. When the failure process is modeled by a nonhomogeneous Poisson process with a mean value function, $m(t)$, a popular cost structure [15, 25, 27, 39] is applied as

$$C(T) = c_1 m(T) + c_2 (m(\infty) - m(T)) + c_3 T, \quad (23)$$

where T is the release time of the software to be determined. In addition, c_1 is the expected cost of removing a fault during the testing phase, c_2 is the expected cost of removing a fault during the operation phase, and c_3 is the expected cost per unit time of testing. Specifically, the expected cost of removing a fault during the operation phase is higher than that during the testing phase; that is, $c_2 > c_1$. Overall, the above cost function is the summation of the testing cost, including the failure cost during testing and actual cost of testing, and the cost of failure in the field.

Proposition 1. *If only the cost is considered, the optimal release time satisfies* $m'(T^*) = c_3/(c_2 - c_1)$.

Proof of Proposition 1. Note that the objective function is as follows:

$$C(T) = c_1 m(T) + c_2 (m(\infty) - m(T)) + c_3 T. \quad (24)$$

The first-order condition yields

$$C'(T^*) = c_1 m'(T^*) - c_2 m'(T^*) + c_3 = 0, \quad (25)$$

which indicates $m'(T^*) = c_3/(c_2 - c_1)$. □

It is also reasonable to quit testing as soon as the number of remaining faults in the system is less than a prescribed portion of total faults. Hence, a simple criterion for software release is given [8, 25]:

$$R_1(t) = \frac{m(t)}{a}, \quad (26)$$

which is the type-I reliability definition.

Alternatively, given that the testing or the operation has been conducted up to time t, the probability that a software failure will not occur in the time interval $(t, t+\delta)$, with $\delta \geq 0$, can be given by the following conditional probability [3, 40]:

$$R_2(\delta \mid t) = e^{-(m(t+\delta)-m(t))}, \quad (27)$$

which is the type-II reliability definition.

In most situations, the project managers need to strike a balance between the cost and the reliability level. That is, on one hand, the reliability level should be at least as high as a desired level set beforehand; on the other hand, given the target level of software reliability α, the cost minimization problem over the release time T should be imposed. Considering this tradeoff yields

$$R_1(T) \geq \alpha \quad (28)$$

for the type-I reliability definition or

$$R_2(\delta \mid T) \geq \alpha \quad (29)$$

for the type-II reliability definition.

As an illustrative example, Figure 11 presents how the software reliability and the total cost change with time. In this example, we use the data set with Apache 2.0.36 and take $\delta = 5$, $c_1 = 20\$$, $c_2 = 150\$$, and $c_3 = 15\$$. Suppose that the reliability must be higher than 0.95; that is, $\alpha = 0.95$. Then, the reliability constraint can be satisfied only if $T \geq T_1 = 45.99$ for the type-I reliability definition (or $T \geq T_2 = 66.95$ for the type-II reliability definition). When only the total cost is considered, we find that the cost is minimized at $T_c = 54.37$. Furthermore, when both criteria (reliability and cost) are taken into account, we find that $T^* = 54.37$ for the type-I reliability definition (or $T^* = 66.95$ for the type-II reliability definition).

In addition, for Apache 2.0.35, the reliability level is above 0.95 only if $T \geq T_1 = 30.91$ if the type-I reliability definition is taken (or $T \geq T_2 = 55.15$ for the type-II reliability definition). When only the cost is considered, it is minimized at $T_c = 38.13$. Furthermore, when both criteria (reliability and cost) are taken into account, we find that $T^* = 38.13$ for the type-I reliability definition (or $T^* = 55.15$ for the type-II reliability definition).

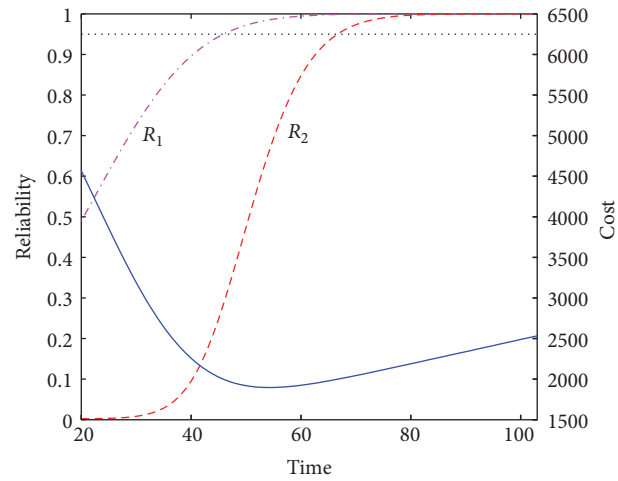

FIGURE 11: The reliability and the cost as a function of time for Apache 2.0.36. Solid line: cost; dash-dot line: type-I definition of reliability; dash line: type-II definition of reliability; dot line: target level of reliability.

Similarly, for Apache 2.0.39, the reliability level is higher than 0.95 only if $T \geq T_1 = 62.21$ if the type-I reliability definition is taken (or $T \geq T_2 = 103.20$ for the type-II reliability definition). When only the cost is considered, the minimization is attained at $T_c = 61.61$. When both reliability and cost are considered, we find that $T^* = 62.21$ for the type-I reliability definition (or $T^* = 103.20$ for the type-II reliability definition).

4.2. Sensitivity Analysis. As mentioned above, we have incorporated both the testing-effort and error interdependency effects into our proposed model. In this part, we intend to examine what if the testing effort, the error interdependency, or both of them are ignored in decision-making of software release. The first case corresponds to the No-k model (14), the second case corresponds to the Only-G1 model (15), and the third case is just the Goel-Okumoto model (13).

For each model, we shall examine the optimal release time and the corresponding minimal cost when a desired reliability level is set beforehand. The data set with Apache 2.0.36 is used to illustrate our findings. Under the type-I reliability definition for reliability measurement, the optimal release time and the minimal cost as a function of α are presented in Figures 12 and 13, respectively. Similarly, under the type-II reliability definition, the optimal release time and the minimal cost as a function of α are shown in Figures 14 and 15, respectively. Other parameters take the same values as analysis in Section 4.1; that is, $\delta = 5$, $c_1 = 20\$$, $c_2 = 150\$$, and $c_3 = 15\$$.

Under either type-I or type-II reliability definition, both the optimal release time and the total cost would be

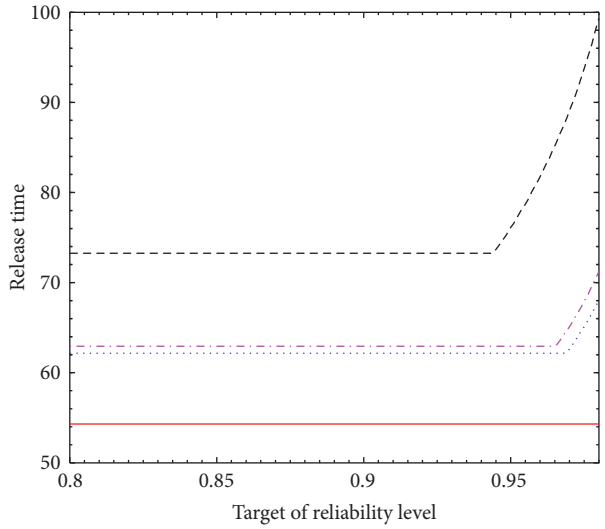

Figure 12: Optimal release time as a function of desired reliability level, with estimated values based on Apache 2.0.36. The type-I reliability definition is adopted for such analysis. Solid line: our proposed mode; dot line: the Only-G1 model; dash-dot line: the No-k model; dash line: the Goel-Okumoto model.

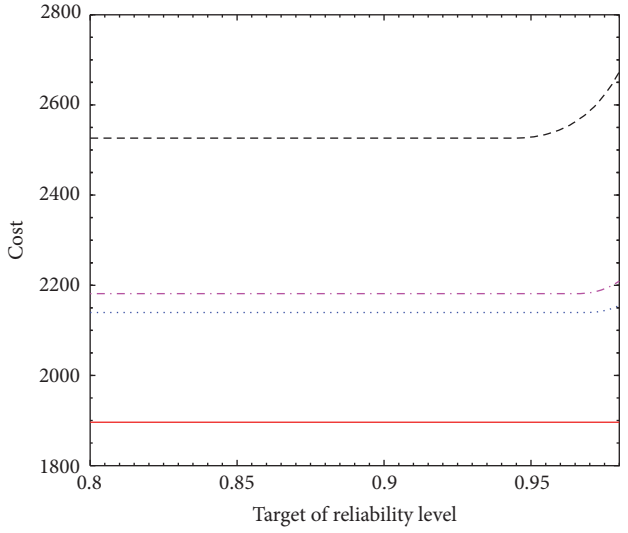

Figure 13: Total cost as a function of desired reliability level, with estimated values based on Apache 2.0.36. The type-I reliability definition is adopted for such analysis. Solid line: our proposed mode; dot line: the Only-G1 model; dash-dot line: the No-k model; dash line: the Goel-Okumoto model.

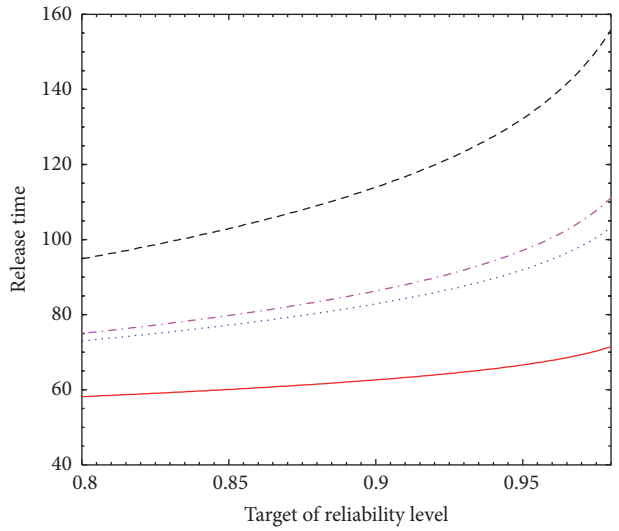

Figure 14: Optimal release time as a function of desired reliability level, with estimated values based on Apache 2.0.36. The type-II reliability definition is adopted for such analysis. Solid line: our proposed mode; dot line: the Only-G1 model; dash-dot line: the No-k model; dash line: the Goel-Okumoto model.

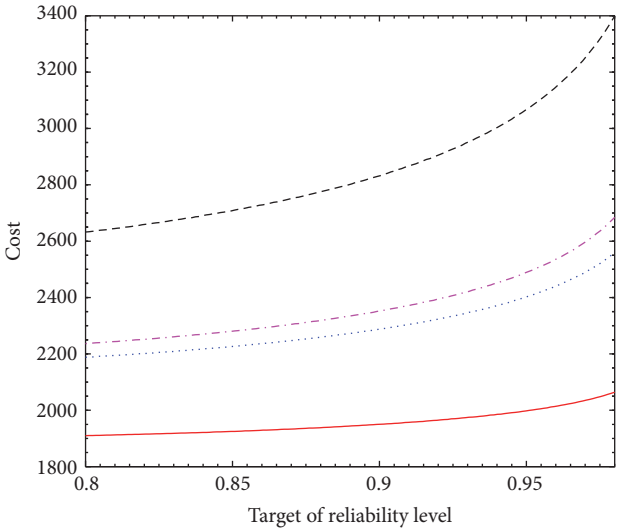

Figure 15: Total cost as a function of desired reliability level, with estimated values based on Apache 2.0.36. The type-II reliability definition is adopted for such analysis. Solid line: our proposed mode; dot line: the Only-G1 model; dash-dot line: the No-k model; dash line: the Goel-Okumoto model.

mistakenly estimated if either testing effort or fault interdependency is ignored. Unlike these two cases, our proposed model always suggests an earlier release time and a lower testing cost.

As an example, we compare the Goel-Okumoto model and our proposed model when the type-I reliability definition is adopted and α is set as 95%. The Goel-Okumoto model indicates the best release time to be 76.56 while that for our proposed model is 54.3. Hence, the testing lasts much longer if the Goel-Okumoto model is used to decide the stopping time. Similarly, using the Goel-Okumoto model yields the minimal cost as 2529$ while that for our proposed model is just 1896$, implying that a decision based on our model can save the software developers as much as 25% in cost.

Alternatively, when the type-II reliability definition is adopted, the Goel-Okumoto model predicts the best release time as 132.2 while that for our proposed model is 66.75, which suggests that the testing duration in our model is doubled compared with traditional Goel-Okumoto model. Similarly, according to the Goel-Okumoto model,

the minimal cost is 3083$ while that for our proposed model is merely 2000$. Thus, a decision based on our model can save the software developers as much as 35% in terms of the cost.

5. Concluding Remarks

In the development process, a variety of tests need to be conducted to ensure the reliability of the software system. In this work, we propose a framework for software reliability modeling that simultaneously captures effects of testing effort and interdependence between error generations. In order to evaluate our proposed model, we compare its performance with other existing models by doing experiments with actual software failure data, that is, three versions of Apache. The nonlinear least square estimation technique is applied for parameter estimation, and values of R^2, mean of square error, Theil Statistic, and relative error are used for model comparison in terms of fitting quality and prediction capability. The numerical analysis from all three Apache versions demonstrates that the proposed model fits the real-world data very well and exhibits a high level of prediction capability.

Our theoretical results can help software project managers assess the most cost-effective time to stop software testing and to release the product. Specifically, we have formally examined the release timing and provided useful managerial insights based on the Apache data sets. Moreover, our proposed model recommends an earlier release time and a reduced cost compared with the Goel-Okumoto model and the Yamada delayed S-shaped model. Therefore, our modeling essentially enhances accuracy of the understanding towards the software reliability growth process and the best time to release a software version.

Notations

i:	The ith generation of fault
t:	Testing time
$m_i(t)$:	Mean value function of the ith generation faults
$m(t)(= \sum_1^3 m_i(t))$:	Mean value function for all three generations of faults
m_j:	The jth observation
$\lambda(t)$:	Failure intensity function
$N(t)$:	Cumulative number of software errors detected by testing time t
a:	Initial content of all three generations of faults
p_i:	Proportion of generation i fault
$a_i(= ap_i)$:	Initial content of generation i fault
b:	Coefficient for the testing-effort function
k:	Power of the testing-effort function
$w(t)(= bt^k)$:	Testing-effort expenditure at time t
$W(t)$:	Cumulative testing-effort expenditure by time t
R^2:	Metric for goodness of fit
MSE:	Mean of square error
TS:	Theil Statistic
RE:	Relative error
$C(T)$:	Total testing cost
$R_1(t)$:	Type-I reliability definition of software systems
$R_2(t)$:	Type-II reliability definition of software systems
α:	Target level of reliability
c_l $(l = 1, 2, 3)$:	Testing cost coefficient.

Competing Interests

The authors declare that they have no competing interests.

References

[1] P. K. Kapur, S. Anand, S. Yamada, and V. S. S. Yadavalli, "Stochastic differential equation-based flexible software reliability growth model," *Mathematical Problems in Engineering*, vol. 2009, Article ID 581383, 15 pages, 2009.

[2] P. K. Kapur, A. Gupta, and V. S. S. Yadavalli, "Software reliability growth modeling using power function of testing time," *International Journal of Operations and Quantitative Management*, vol. 12, no. 2, pp. 127–140, 2006.

[3] C.-Y. Huang, M. R. Lyu, and S.-Y. Kuo, "A unified scheme of some nonhomogenous poisson process models for software reliability estimation," *IEEE Transactions on Software Engineering*, vol. 29, no. 3, pp. 261–269, 2003.

[4] C.-J. Hsu and C.-Y. Huang, "Optimal weighted combinational models for software reliability estimation and analysis," *IEEE Transactions on Reliability*, vol. 63, no. 3, pp. 731–749, 2014.

[5] H. Okamura, T. Dohi, and S. Osaki, "Software reliability growth models with normal failure time distributions," *Reliability Engineering and System Safety*, vol. 116, pp. 135–141, 2013.

[6] Y. Tamura and S. Yamada, "A component-oriented reliability assessment method for open source software," *International Journal of Reliability, Quality and Safety Engineering*, vol. 15, no. 1, pp. 33–53, 2008.

[7] Y. Tamura and S. Yamada, "Comparison of software reliability assessment methods for open source software," in *Proceedings of the 11th International Conference on Parallel and Distributed Systems*, vol. 2, pp. 488–492, IEEE, Fukuoka, Japan, July 2005.

[8] X. Li, Y. F. Li, M. Xie, and S. H. Ng, "Reliability analysis and optimal version-updating for open source software," *Information and Software Technology*, vol. 53, no. 9, pp. 929–936, 2011.

[9] X. Li, M. Xie, and S. H. Ng, "Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points," *Applied Mathematical Modelling*, vol. 34, no. 11, pp. 3560–3570, 2010.

[10] S. M. Rafi, K. N. Rao, and S. Akthar, "Incorporating generalized modified Weibull TEF in to software reliability growth model and analysis of optimal release policy," *Computer and Information Science*, vol. 3, no. 2, article 145, 2010.

[11] C. Jin and S.-W. Jin, "Parameter optimization of software reliability growth model with S-shaped testing-effort function using improved swarm intelligent optimization," *Applied Soft Computing*, vol. 40, pp. 283–291, 2016.

[12] J. Wang, Z. Wu, Y. Shu, and Z. Zhang, "An imperfect software debugging model considering log-logistic distribution fault content function," *Journal of Systems and Software*, vol. 100, pp. 167–181, 2015.

[13] S. Yamada, H. Ohtera, and H. Narihisa, "A testing-effort dependent software reliability model and its application," *Microelectronics Reliability*, vol. 27, no. 3, pp. 507–522, 1987.

[14] S. Yamada and H. Ohtera, "Software reliability growth models for testing-effort control," *European Journal of Operational Research*, vol. 46, no. 3, pp. 343–349, 1990.

[15] C.-Y. Huang and S.-Y. Kuo, "Analysis of incorporating logistic testing-effort function into software reliability modeling," *IEEE Transactions on Reliability*, vol. 51, no. 3, pp. 261–270, 2002.

[16] C.-Y. Huang, "Performance analysis of software reliability growth models with testing-effort and change-point," *Journal of Systems and Software*, vol. 76, no. 2, pp. 181–194, 2005.

[17] N. Ahmad, M. G. M. Khan, and L. S. Rafi, "A study of testing-effort dependent inflection S-shaped software reliability growth models with imperfect debugging," *International Journal of Quality and Reliability Management*, vol. 27, no. 1, pp. 89–110, 2010.

[18] 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," *Reliability Engineering and System Safety*, vol. 126, pp. 37–43, 2014.

[19] V. B. Singh, G. P. Singh, R. Kumar, and P. K. Kapur, "A generalized reliability growth model for open source software," in *Proceedings of the 2nd International Conference on Reliability, Safety and Hazard (ICRESH '10)*, pp. 523–528, IEEE, Mumbai, India, December 2010.

[20] A. Pievatolo, F. Ruggeri, and R. Soyer, "A Bayesian hidden Markov model for imperfect debugging," *Reliability Engineering and System Safety*, vol. 103, pp. 11–21, 2012.

[21] T. Aktekin and T. Caglar, "Imperfect debugging in software reliability: a Bayesian approach," *European Journal of Operational Research*, vol. 227, no. 1, pp. 112–121, 2013.

[22] K. Goševa-Popstojanova and K. Trivedi, "Failure correlation in software reliability models," in *Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE '99)*, pp. 232–241, November 1999.

[23] M. Sahinoglu, "Compound-Poisson software reliability model," *IEEE Transactions on Software Engineering*, vol. 18, no. 7, pp. 624–630, 1992.

[24] V. B. Singh, K. Yadav, R. Kapur, and V. S. S. Yadavalli, "Considering the fault dependency concept with debugging time lag in software reliability growth modeling using a power function of testing time," *International Journal of Automation and Computing*, vol. 4, no. 4, pp. 359–368, 2007.

[25] C.-Y. Huang and C.-T. Lin, "Software reliability analysis by considering fault dependency and debugging time lag," *IEEE Transactions on Reliability*, vol. 55, no. 3, pp. 436–450, 2006.

[26] N. Ahmad, M. G. M. Khan, S. M. K. Quadri, and M. Kumar, "Modelling and analysis of software reliability with Burr type X testing-effort and release-time determination," *Journal of Modelling in Management*, vol. 4, no. 1, pp. 28–54, 2009.

[27] O. Singh, P. K. Kapur, and A. Anand, "A multi-attribute approach for release time and reliability trend analysis of a Software," *International Journal of Systems Assurance Engineering and Management*, vol. 3, no. 3, pp. 246–254, 2012.

[28] B. Pachauri, A. Kumar, and J. Dhar, "Software reliability growth modeling with dynamic faults and release time optimization using GA and MAUT," *Applied Mathematics and Computation*, vol. 242, pp. 500–509, 2014.

[29] M. Xie, *Software Reliability Modelling*, vol. 1, World Scientific, River Edge, NJ, USA, 1991.

[30] M. R. Lyu, *Handbook of Software Reliability Engineering*, vol. 222, IEEE Computer Society Press, Los Alamitos, Calif, USA, 1996.

[31] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 206–211, 1979.

[32] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," *IEEE Transactions on Reliability*, vol. 32, no. 5, pp. 475–484, 1983.

[33] S. Yamada, M. Ohba, and S. Osaki, "S-shaped software reliability growth models and their applications," *IEEE Transactions on Reliability*, vol. 33, no. 4, pp. 289–292, 1984.

[34] A. Wood, "Predicting software reliability," *Computer*, vol. 29, no. 11, pp. 69–77, 1996.

[35] T. F. Coleman and Y. Li, "On the convergence of interior-reflective Newton methods for nonlinear minimization subject to bounds," *Mathematical Programming*, vol. 67, no. 2, pp. 189–224, 1994.

[36] K. Pillai and V. S. S. Nair, "A model for software development effort and cost estimation," *IEEE Transactions on Software Engineering*, vol. 23, no. 8, pp. 485–497, 1997.

[37] K.-C. Chiu, Y.-S. Huang, and T.-Z. Lee, "A study of software reliability growth from the perspective of learning effects," *Reliability Engineering and System Safety*, vol. 93, no. 10, pp. 1410–1421, 2008.

[38] Y. P. Wu, Q. P. Hu, M. Xie, and S. H. Ng, "Modeling and analysis of software fault detection and correction process by considering time dependency," *IEEE Transactions on Reliability*, vol. 56, no. 4, pp. 629–642, 2007.

[39] M. Xie and B. Yang, "A study of the effect of imperfect debugging on software development cost," *IEEE Transactions on Software Engineering*, vol. 29, no. 5, pp. 471–473, 2003.

[40] S. Yamada, *Software Reliability Modeling: Fundamentals and Applications*, Springer Briefs in Statistics, Springer, Tokyo, Japan, 2014.