IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 17, 2021, accepted March 1, 2021, date of publication March 8, 2021, date of current version March 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3064296

Simulated Software Testing Process and Its Optimization Considering Heterogeneous Debuggers and Release Time

KAIYE GAO, (Member, IEEE)
School of Economics and Management, Beijing Information Science & Technology University, Beijing 100192, China
Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China

e-mail: kygao@foxmail.com

This work was supported in part by the National Natural Science Foundation of China under Grant 72001027, and in part by the Beijing Municipal Commission of Education under Grant KM202111232007.

ABSTRACT Most traditional software reliability growth models (SRGMs) assume immediate fault correction upon detection and therefore only consider fault detection process (FDP). In order to be more realistic, some researchers have tried to incorporate fault correction process (FCP) and fault introduction process (FIP) into the software reliability models. However, it is still difficult to incorporate into the analytical software reliability models some other factors, such as the different fault detection and correction capabilities of debuggers. In this paper, a simulation approach is proposed to model FDP, FIP, and FCP together considering debuggers with different contributions to fault detection rate, different fault correction rate and different fault introduction rate. Besides, this paper also constructed a cost calculation method to optimize the testing design including debuggers assignment and software release time. Some numerical examples are provided to illustrate the proposed model. The results show that the trends of FDP, FCP and FIP are consistent with the intuition to the practice of software testing, and the optimal testing resources allocation and the optimal release time can be obtained according to the proposed model.

INDEX TERMS Fault detection, optimization, reliability, simulation, software debugging.

I. INTRODUCTION

Since the computer era, software has been playing an essentially important role in many complex, sophisticated and safety-critical systems of a variety of significant and multi-faceted areas, such as air traffic analysis and control, nuclear reactor testing and management, military and defense services, etc. [1]–[7]. The failure of software may lead to significant losses. For example, 228 casualties that resulted from the crash of the AF-447 in the Atlantic on June 1, 2009 are one of the tragic consequences of aircraft accidents in which software was involved [8].

As a result, the reliability of the software is a common concern for its developers and users. In order to ensure high reliability, developers usually need to conduct testing before they launch the software on public market. Related studies of software testing and reliability have been on-going for thirty years and the increasing dependability of various fields on software creates a need to make the software field more accurate and reliable [9]–[14]. Besides, in order to track the growth of software reliability during testing and make some important decisions, such as the optimal software release time and the staffing level, numerous software reliability growth models (SRGMs) have been proposed during the past four decades.

Most analytical models of SRGM only studied the fault detection process, and ignored the time needed from detection of a fault to the correction of a fault. In practice, a detected fault needs to be reported, confirmed and finally corrected, which may need a few days or even a few weeks. Thus fault correction process (FCP) can be regarded as a delayed process after fault detection process (FDP) and can be modeled together with FDP using a queuing system [15]–[19]. In [15] and [17], some software reliability models were proposed considering different types of debugging delay, which is the time between fault detection and fault correction. In [20],

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

some paired models of fault detection and fault correction are proposed with consideration of the uneven testing resource allocation during software testing phase. Moreover, the fault introduction process (FIP) is also incorporated. In [19], the reliability growth of open sourced software is studied considering both fault detection and fault corrections.

Though the analytical models of SRGM have been incorporating more factors in order to be more realistic, it is still difficult to incorporate into the analytical models some important information, such as the different debugging capabilities of different debuggers. In [21], the number of debuggers is incorporated into FDP and FCP through a rate-based simulation approach. However, it is assumed that all the debuggers share the same fault correction ability. In [22], the influence of the different debuggers on the fault correction process is simulated. However, it is assumed that the no new faults are introduced during fault correction. Besides, it is assumed that the fault detection rate does not depend on the number of debuggers, which is not realistic.

In practice, the debuggers need to edit the code during the fault correction process, thus it is possible to introduce new faults into the code [23], [24]. Also, the fault detection rate tends to increase if there are more debuggers available. The two factors will influence the software reliability so that the decision making on the basis of software reliability will also be altered. Thus, considering the two points, the framework of SRGM needs to be reconstructed, that is, heterogeneous debuggers with different capabilities and the FIP from these debuggers should be introduced into the SRGM.

In this paper, a framework of SRGM is proposed to simulate the software testing process considering heterogeneous debuggers with different contributions to the fault detection rate, different fault correction rate and fault introduction rate. Testing also consumes a large amount of resources, such as human power and CPU hours, which are usually not constantly allocated during testing phase. Besides, uncorrected faults in the software of released version may result in economic loss. Herein, in order to minimize the total cost during the testing phase and the operational phase, testing design including the personnel assignment of debuggers and the release time of software are also of great concern. Thus, a cost calculation method is proposed to optimize the testing design in this paper. Specifically, all the processes are simulated with a queuing system adopting the Simulink toolbox of Matlab and all the calculations are done with Matlab. The numbers of detected faults, corrected faults, and introduced faults in different time intervals can be collected and the total cost can be calculated accordingly.

The contribution of this study is threefold. Firstly, this study uses a new framework of SRGM to simulate the software testing process in order to improve the prediction accuracy of the detected faults and corrected faults in each period. Secondly, the heterogeneous debuggers with different capabilities is considered in the proposed model where debuggers are not only an important factor to software reliability but also a testing resource to be allocated optimally. Lastly, this study incorporates FIP which is related with heterogeneous debuggers into our proposed model. As demonstrated by the case study using real data, the prediction results of our proposed model can help the decision maker in reliability assessment, cost prediction and determination of the optimal release time.

The remaining of this paper is arranged as follows. The general framework of the queuing model and cost calculation is proposed in Section II, with description of the simulation framework. Specifically, all the boxes used in the Simulink are explained. Numerical examples are shown in Section III to illustrate the simulation approach. Section IV illustrates the testing design optimizing. Section V concludes and points out some future directions.

The abbreviations, acronyms and nomenclature used in this paper are shown in Table 1.

II. MODELLING FRAMEWORK

In this section, the modelling framework of the software reliability is proposed.

A. ASSUMPTION

In order to provide mathematical tractability, SRGMs are often derived under some restrictive assumptions [25]–[27]. The fault detection, correction and introduction processes can be described as a whole using a queuing system, with FDP being the arrival process, FCP being the departure process and FIP being the feedback. Then, to briefly depict the problem, this study uses the following assumptions:

(1) FDP corresponds to the arrival process
(2) FCP corresponds to the departure process
(3) FIP corresponds to the feedback
(4) the FDP is described as a NHPP characterized with intensity function $\lambda d(t)$ and mean value function $md(t)$
(5) the FCP is described as a NHPP characterized with intensity function $\lambda c(t)$ and mean value function $mc(t)$
(6) the FCP can be simulated based on the simulated FDP and the fault correction time
(7) there are totally k types of debuggers and the number of the i-th type debuggers is $\mathrm{SN}(i)$
(8) the fault detection rate is assumed to be constant and it is a function of $\mathrm{SN} = [\mathrm{SN}(1), \ldots, \mathrm{SN}(k)]$, denoted as $b = f(\mathrm{SN}(1), \ldots, \mathrm{SN}(k))$
(9) the form of $b = f(\mathrm{SN}(1), \ldots, \mathrm{SN}(k))$ can be estimated from similar projects or by experts
(10) the time for a type-i debugger to correct a fault is random, and it is allowed to observe arbitrary distribution
(11) The cost of software testing includes the fixed cost, labor cost, bonus for fault correction and the economic loss of uncorrected faults
(12) the fixed cost increases in a constant speed with the time of software testing process
(13) the labor cost comes from the wages of debuggers, which is paid based on the working hours

TABLE 1. Abbreviations, acronyms and nomenclature.

Symbol	Quantity
SRGMs	software reliability growth models
FDP	fault detection process
FCP	fault correction process
FIP	fault introduction process
NHPP	nonhomogeneous Poisson process
GO	Goel–Okumoto
$\lambda_d(t)$	the NHPP intensity function of FDP
$m_d(t)$	the NHPP mean value function of FDP
$\lambda_c(t)$	the NHPP intensity function of FCP
$m_c(t)$	the NHPP mean value function of FCP
k	the number of different types of debuggers
SN(i)	the number of the i-th type debuggers
SN	[SN(1), …, SN(k)]
a	the number of faults in the software before testing
b	fault detection rate
$\{N(t), t \geq 0\}$	a counting process representing the cumulative number of faults by time t that happens in the initial version of software
$m(t)$	a bounded, non-decreasing function of t
$\Lambda(x)$	a monotone non-decreasing right-continuous function and it is bounded in the fixed interval (0, T]
$F(t)$	the fixed cumulative density function of FDP
d_i^j	the collected fault detection number
c_i^j	the collected fault correction number
e_i^j	the collected fault introduction number
$\overline{d}_i$	the average cumulative number of detected faults
$\overline{c}_i$	the average cumulative number of corrected faults
$\overline{e}_i$	the average cumulative number of introduced faults
C	the cost of software testing
C_f	the fixed cost
C_l	the labor cost
C_b	the bonus for fault correction
C_e	the economic loss due to uncorrected faults
Co_f	the fixed cost per unit time
Co_{l1}	labor cost for debuggers of type 1 per capita per unit time
Co_{l2}	labor costs for debuggers of 2 per capita per unit time
Co_b	the bonus for a corrected fault
Co_e	the economic loss of punishment due to an uncorrected fault

(14) for the debugger with stronger capability, the wage per unit time is higher
(15) every time a debugger corrects a fault, the debugger is awarded a fixed bonus
(16) the economic loss is great if the fault is uncorrected at the end of software testing process
(17) the software is immediately released and operated when the testing is finished

Assumptions (1)-(3) connect FDP, FCP and FIP with queuing system. When a fault is detected, it arrives debuggers for correction; once this fault is corrected, it departs the debuggers with a probability of introducing new faults as a kind of feedback. Implied by this discussion, assumptions (1)-(3) are proposed.

On basis of assumptions (1)-(3), it is needed to determine the processes for the detection and correction of the faults initially embedded in the software. In previous studies, there are mainly four types of models that used to model this process, which are Markov models, data-driven models, simulation models, and non-homogeneous Poisson process (NHPP) models.

In Markov-based SRGMs, the cumulative number of failures is described as a pure birth process with state-dependent transition rates when one counts the number of software failures experienced during the testing phase [28], [29]. For data-driven SRGMs, the software process is viewed as a time series and it is assumed that a software failure is strongly correlated with the most recent failures [30]–[32]. In reality, this assumption may not be valid and hence the model performance would be affected [33]. In contrast, simulation approaches can relax certain impractical assumptions that are common in model-based approaches [34], [35]. However, most simulation approaches assume that each time a failure occurs, the corresponding fault will be removed with absolute certainty, which simplifies the simulation process significantly [36], [37].

Among all the models, the NHPP models can be regarded as the most effective [38], [20], [39]–[41]. It is proposed by Goel and Okumoto in 1979, and their framework of NHPP models is called Goel–Okumoto (GO) model [42]. Thus, this study also uses NHPP to describe the processes of the detection and correction of faults, which forms assumptions (4) and (5). Since the fault correction time is considered in this study, assumption (6) is also proposed after assumptions (4) and (5).

Assumption (7) gives status of debuggers including the number of debugger types and the number of debuggers of each type. Assumption (8) defines the relation between the fault detection rate and the status of debuggers, while assumption (9) provides an approach to estimate this relation. Assumption (10) defines the correction time for different types of debugger, which is the main characteristic of debuggers.

The statement related to cost are given in assumptions (11)-(13). Assumption (11) defines the area of the cost concerned in this study. Assumptions (12)-(16) describe the fixed cost, the labor cost, the fixed bonus and the economic loss respectively.

Finally, the last assumption determines the release time of the software. In practice, the software may not be released immediately after the testing is finished. However, there are too many factors that may influence the decision of release time. Considering these factors will make the model much more complex and difficult to analyze. Moreover, once the software testing is finished, it should be ready to be used at any following time. If there are some faults in the software at that time, the software will still bring economic loss to its developer, operator and user. Thus, to facilitate the modelling, assumption (17) states that the software is immediately released and operated when the testing is finished.

B. SOFTWARE TESTING PROCESS FORMULATION

First, a NHPP process is introduced for the detection of the faults initially embedded in the software [43]. This study uses a counting process $\{N(t),\ t \geq 0\}$, to represent the cumulative number of detected original faults by time t. Since there are no faults detected at beginning, we have $N(0) = 0$.

By using the NHPP failure process, it is assumed that the number of faults detected during non-overlapped time intervals are independent. In other words, the n random variables $N(t_1),\ N(t_2)-N(t_1),\ \ldots,\ N(t_n)-N(t_{n-1})$ are s-independent increments for any finite collection of times $t_1 < t_2 < \ldots < t_n$.

Let $m(t)$ represent the s-expected detected number of software faults by time t. If the fault introduction is not considered, then the s-expected number of faults remaining in the system at any time is finite. Thus, $m(t)$ is a bounded, non-decreasing function of t. If we use a to denote the expected number of faults initially embedded in the software to be eventually detected, $m(t)$ follows the boundary condition:

$$m(t) = \begin{cases} 0, & t = 0 \\ a, & t \to \infty. \end{cases} \tag{1}$$

Assume that the s-expected number of software faults during $(t,\ t + \Delta t)$ is essentially in proportion to the s-expected number of undetected original faults at t, that is

$$m\left(t,\ t + \Delta t\right) - m\left(t\right) = b\left\{a - m\left(t\right)\right\}\Delta t + o\left(\Delta t\right), \tag{2}$$

where b is the fault detection rate, and $o\left(\Delta t\right)/\Delta t$ infinitely approaches 0 as $\Delta t \to 0$. Using Equation (2), by letting $\Delta t \to 0$, we have:

$$\lambda(t) = m'\left(t\right) = ab - bm\left(t\right), \tag{3}$$

where $\lambda(t)$ is the fault intensity function.

Solving Equation (3) under boundary conditions shown in Equation (1) yields

$$m(t) = a(1 - e^{-bt}). \tag{4}$$

As discussed above, the FDP of the original faults needs to be simulated from work space, instead of in Simulink. Then for the FDP, the mean value function can be given as

$$m_d(t) = a(1 - e^{-bt}). \tag{5}$$

and the intensity function can be calculated to be

$$\lambda_d(t) = m_d'(\mathrm{t}) = abe^{-bt}. \tag{6}$$

The FDP is simulated in the time interval of $(0, T]$. Since the general order statistics (GOS) models are closely related to NHPP models [44], simulation of FDP can be implemented through generation of a Poisson number of order statistics from a fixed cumulative density function [45]. Specifically, if $X_1, X_2, \ldots, X_n$ are the points of a NHPP in a fixed interval $(0, T]$, and if $N(T) = n$, then conditional on having observed $n(>0)$ points in $(0,\ T]$, then X_i are distributed as the order statistics from a sample of size n from the distribution function

$$\frac{\Lambda\left(t\right) - \Lambda\left(0\right)}{\Lambda\left(T\right) - \Lambda\left(0\right)}, \quad 0 < t \leq T, \tag{7}$$

where $\Lambda\left(x\right)$ is a monotone non-decreasing right-continuous function and it is bounded in the fixed interval $(0,\ T]$ [45].

Then, since $m_d\left(x\right)$ is non-decreasing and right-continuous, we let $m_d\left(x\right) = \Lambda\left(x\right)$. Finally, from formula (7), we can simulate FDP by the fixed cumulative density function shown as below

$$F\left(t\right) = \frac{m_d\left(t\right)}{m_d\left(T\right)}, \quad 0 < t \leq T. \tag{8}$$

The simulation of the fault correction time is straightforward, with traditional inverse transformation approach. The FCP can be obtained based on the simulated FDP and the fault correction time. For the case where there are infinite number of homogeneous debuggers, the mean value function of FCP can be formulated as shown below.

The correction time of software is often assumed to observe exponential distribution. Here, consider a special case where there are infinite debuggers and the correction time of each debugger observes the same exponential distribution as $\Delta\left(t\right) = \exp\left(\mu\right)$. Then using the intensity function of fault detection, $\lambda_d(t)$, the correction intensity function can be calculated as the expectation of $\lambda_d(t - \Delta(t))$, that is

$$\begin{aligned} \lambda_c(t) &= E\left[\lambda_d(t - \Delta(t))\right] = \int_0^\infty \lambda_d(t - x)\mu e^{-\mu x}dx \\ &= \int_0^\infty abe^{-b(t-x)}\mu e^{-\mu x}dx. \end{aligned} \tag{9}$$

Then we can have the mean value function of FCP as

$$m_c(t) = \int_0^t \lambda_c(t)dt. \tag{10}$$

Specifically, if the FDP is described by the GO model, the mean value function for the fault correction process is:

$$m_c(t) = \begin{cases} a\left[1 - (1 + bt)e^{-bt}\right], & \mu = b \\ a\left[1 - \dfrac{\mu}{\mu - b}e^{-bt} + \dfrac{b}{\mu - b}e^{-\mu t}\right], & \mu \neq b \end{cases} \tag{11}$$

where the $m_c(t)$ has the same form as the $m_d(t)$ for an s-shaped NHPP model with $\mu = b$.

For the debugger, during the correction of each fault, it is unavoidable that some codes are changed. If the code is changed, it is likely to introduce new faults. This paper assumes that the correction of each fault has a probability of introducing a new fault. With the FCP and the fault introduction rate of different types of debuggers, the FIP can be obtained through the bottom path in Figure 1. The fault introduction rate is set to be constant for each type of debuggers.

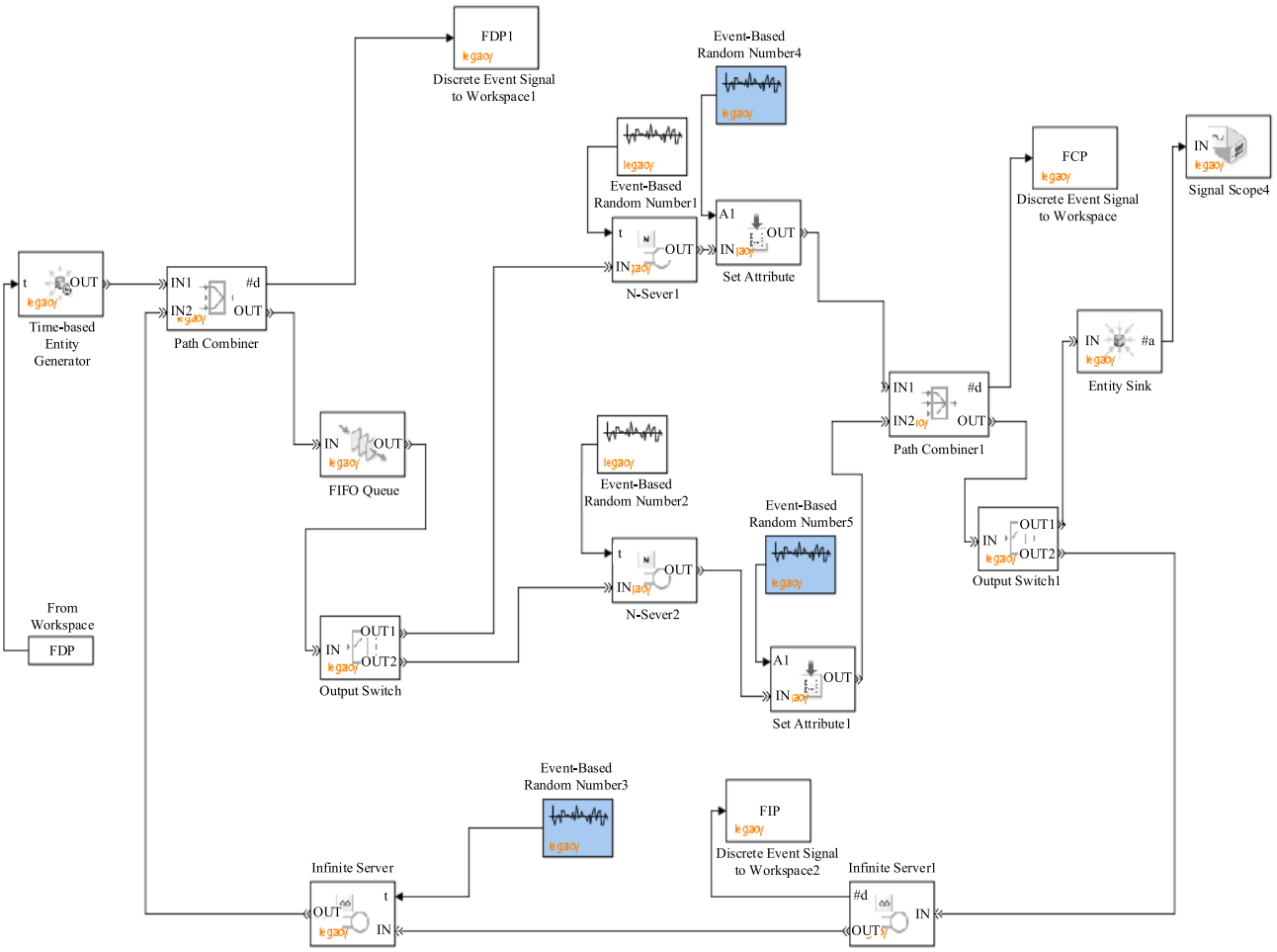

FIGURE 1. Software FDP, FCP and FIP simulated with Simulink.

C. COST CALCULATION

According to assumptions listed above, the cost of software testing includes four parts: fixed cost, labor cost, bonus for fault correction and economic loss due to uncorrected faults, as shown below

$$C = C_f + C_l + C_b + C_e \tag{12}$$

where C_f indicates the fixed cost, C_l indicates the labor cost, C_b indicates the bonus for fault correction and C_e indicates the economic loss due to uncorrected faults.

The fixed cost is mainly concerned with the unchangeable expenditure for the testing, such as the rental for the office, the cost of testing equipment or materials, and so on. Assuming that the fixed cost is monotonously increasing with testing time in a constant speed, we can have the fixed cost C_f as shown below

$$C_f = Co_f \times t \tag{13}$$

where Co_f is the fixed cost per unit time.

Labor cost is mainly from the wages of debuggers. Employing debuggers with stronger capability usually needs to pay higher wages, so the labor costs for different types of debuggers are different. Besides, since the wages are generally calculated by working hours, labor cost also increases with the testing hours. In this case, the labor cost C_l can be expressed as below.

$$C_l = Co_{l1} \times \mathrm{SN}(1) \times t + Co_{l2} \times \mathrm{SN}(2) \times t \tag{14}$$

where Co_{l1} is the labor cost per debuggers of type 1 per unit time Co_{l2} is the labor cost per debuggers of type 2 per unit time.

Except the wages, debuggers can also receive bonus if they successfully correct faults. The bonus of debuggers is accumulated according to the number of faults they correct, which is shown as below.

$$C_b = Co_b \times m_c(t) \tag{15}$$

where Co_b is the bonus for a fault corrected.

The last item is a punitive expenditure. If the faults are still uncorrected by the end of the testing process, they will result in some economic loss. Because once the software is launched on the market, the faults in it will cause tremendous

adverse influence. Thus, this economic loss is usually huge in practice and should be taken into the cost account, as shown below.

$$C_e = Co_e \times (a + m_i(t) - m_c(t)) \tag{16}$$

where Co_e is the economic loss of punishment due to an uncorrected fault. It can be seen from equation (16) that the number of uncorrected faults equals the number of the faults in the beginning plus the number of faults introduced and minus the number of faults corrected.

Finally, by substituting equations (13) - (16) into (12), we can calculate the total cost, as shown in equation (17).

$$\begin{aligned} C = {} & Co_f \times t + Co_{l1} \times \mathrm{SN}(1) \times t + Co_{l2} \times \mathrm{SN}(2) \times t \\ & + Co_b \times m_c(t) + Co_e \times (a + m_i(t) - m_c(t)) \end{aligned} \tag{17}$$

By minimizing the cost calculated in equation (17), we can make the optimal decisions about the software testing process, for example, to optimize the debugger numbers of different types or the software release time.

D. SIMULATION

In order to simulate the fault detection and correction processes, the Simulink toolbox in Matlab is used. It is a graphical programming environment for modelling simulating and analyzing multi-domain dynamical systems. Simulink is widely used in automatic control and digital signal processing for multi-domain simulation and model-based design [26], [27]. The whole process can be simulated using the queuing entities provided by the Simulink toolbox of Matlab.

In this study, assuming that there are 2 types of debuggers, Figure 1 illustrates the fault detection, correction, and introduction processes. The left-most entity ''From Workspace'' in Figure 1 corresponds to the fault detection process generated without considering the newly introduced faults. The ''Path Combiner'' entity in the left part of Figure 1 combines the fault detection process of the original faults in the software and the fault detection process of the newly introduced faults. The ''Output Switch'' entity chooses the first unblocked channel, which means the detected fault will be handled by the first idle debugger. The ''N-Server1'' and ''N-Server2'' correspond to the two types of debuggers. The ''Path Combiner1'' at the right part of Figure 1 combines the corrected faults from two types of debuggers to get the fault correction process. The ''Output Switch1'' leads to the fault introduction process through port 2, and the probability of choosing this port depends on the attribute of the corrected fault. Note that, the attribute of the corrected fault depends on it is corrected by which type of debuggers. Therefore, entities ''Set Attribute'' and ''Set Attribute1'' serve to assign the fault introduction rate to a fault by debuggers of type 1 and debuggers of type 2.

As for the setting of specifically these two types of debuggers considered, there are three four reasons. First, there may be many types of debugger in practice, but in general they can be classified into good debuggers and not good debuggers. Second, the figure of simulation is too big to present in the paper in terms of the layout, if more types are considered. Third, the optimal resource allocation is convenient to be presented in a figure with 3 dimensions. Last, the model for the case with more types of debuggers can be easily developed based on the model in this study.

III. TESTING PROCESS SIMULATION

In this section, to present the testing process simulation a specific model configuration of the testing process and the simulation results are provided.

A. CALCULATION

A specific model configuration is shown for illustrating the testing process here. It is assumed that there are totally two different kinds of debuggers. Parameters used in this model can be obtained according to experts' estimation through Delphi method. Here these parameters are directly given for the convenience of calculation. The simulation process is briefly described in the following flow chart.

The parameters for the NHPP model of FDP are configured as $a = 100$ and $b = 0.02\mathrm{SN}(1) + 0.01\mathrm{SN}(2)$. Then the $m_d(t)$ can be expressed as

$$m_d(t) = 100(1 - e^{-(0.02\mathrm{SN}(1)+0.01\mathrm{SN}(2))t}), \tag{18}$$

and the $\lambda_d(t)$ can be calculated as

$$\begin{aligned} \lambda_d(t) = {} & 100 \times (0.02\mathrm{SN}(1) + 0.01\mathrm{SN}(2)) \\ & \times e^{-(0.02\mathrm{SN}(1)+0.01\mathrm{SN}(2))t}. \end{aligned} \tag{19}$$

In this study, the exponential distribution is used for the correction and the introduction rate. In many areas, the exponential distribution is the widely used probability distribution that describes the time between events in a Poisson point process, i.e., a process in which events occur continuously and independently at a constant average rate [46], [47].

Herein, we assume that the correction time for the two types of debuggers both observe exponential distribution with parameters $\mu(1) = 0.2$ and $\mu(2) = 0.1$ respectively. The introduction rates for the two types of debuggers both observe exponential distribution with parameters $\gamma(1) = 0.2$ and $\gamma(2) = 0.1$ respectively.

For illustration, we investigate the cases of SN = [5, 3] and SN = [3, 5]. The fault detection rates for both cases are respectively $b = 1.3$ and $b = 1.1$. The mean value functions of the fault detection processes for the two cases are respectively

$$m_d(t|\ SN = [5, 3]) = 100(1 - e^{-1.5t}), \tag{20}$$

and

$$m_d(t|\ SN = [3, 5]) = 100(1 - e^{-1.3t}). \tag{21}$$

As there are finite and heterogeneous debuggers, the mean value function for the fault correction process and the fault

introduction process is not mathematically tractable. However, both of them can be obtained through simulation. The simulation replicates 100 times in the time interval of (0, 100]. Then the collected fault detection, correction, and introduction time data are grouped into the cumulative fault detection number d_i^j, fault correction number c_i^j and fault introduction number e_i^j, $i = 1, 2, \ldots, 100, j = 1, \ldots, 100$. Here j denotes the number of a simulation run, and i denotes the testing time till which the fault detection number, fault correction number, and fault introduction number are cumulated. Finally, we can have the average cumulative number of detected faults, corrected faults, and introduced faults as

$$\bar{d}_i = \frac{1}{100}\sum_{j=1}^{100} d_i^j, \tag{22}$$

$$\bar{c}_i = \frac{1}{100}\sum_{j=1}^{100} c_i^j, \tag{23}$$

$$\bar{e}_i = \frac{1}{100}\sum_{j=1}^{100} e_i^j, \tag{24}$$

where $i = 1, 2, \ldots, 100$ and $j = 1, 2, \ldots, 100$.

From equations (22)-(24), it can be seen that with the number of simulation times replicated, the variances of $\overline{d_i}$, $\overline{c_i}$ and $\overline{e_i}$ will decrease, which illustrates the performance of the simulation method [48]–[50].

B. SIMULATION RESULTS AND DISCUSSIONS

The simulation results are shown in Figure 2. In both Figure 2 (a) and Figure 2 (b), the horizontal axis represents the time and the vertical axis represents the number of faults; three lines representing the numbers of faults for FDP, FCP and FIP are plotted in each figure.

It can be seen clearly that when there are more debuggers of the first type, the mean value functions of FDP, FCP and FIP are higher, which indicate that more debuggers not only contribute more to fault detection rate but also correct faults faster and introduce more faults. This is consistent with the analytical mean value function of fault detection process, as given by (18) and (19).

Specifically, in both (a) and (b), the FDP first increases rapidly to about 100 before a time between 20 and 30. The reason may be that in the beginning of the test, the fault detection is easy due to the comparatively great number of faults in the software. After that, the growth rate suddenly slows down to the end. This means that as the number of faults decreases, detecting faults becomes increasingly harder.

In contrast, the slopes of FCP and FIP curves are more stable in both (a) and (b) in Figure 2. This is mainly because the capacity of correcting is limited. Though a lot of faults may be detected at beginning, some may need to wait in the queue. At the end of the testing process, though the number of detected faults are smaller, the debuggers are more available and thus facilitate the fault correction.

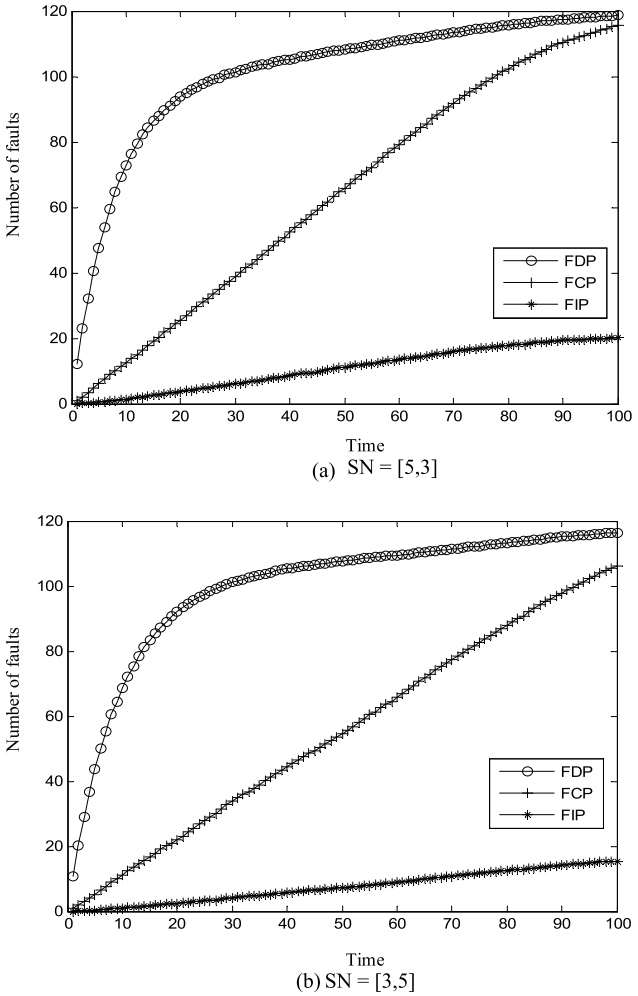

FIGURE 2. Simulated FDP, FCP and FIP.

According to the simulation results, it can be seen that the trends of FDP, FCP and FIP are consistent with the analysis in Section II as well as the intuition to problem description and practice. This proves the effectiveness of the method proposed in this study.

IV. TESTING DESIGN OPTIMIZATION

In general, the system reliability and cost can be obtained by balanced by optimizing the system resources [51]–[53]. Thus, besides the reliability requirement, the testing design including optimal personnel assignment of debuggers and optimal software release time should also be discussed. The target of testing design optimization is to minimize the total cost during the software testing phase and the operational phase. In the testing process, there are mainly two factors, debuggers and release time, which can be adjusted [54].

On the one hand, if debuggers have different capabilities, they may ask for different wages. As both debuggers' capabilities and wages can influence the cost of software testing, then how many debuggers of type 1 and type 2 should be employed is an important question for the software testing design. On the other hand, how long the testing should be

conducted (software release time), can also influence the cost. Most faults may be corrected in a comparatively short time, so if the test lasts too long, many resources may be wasted; on the contrary, if the testing duration is short, more faults will be uncorrected and lead to a loss when the software is launched on the market. Thus, the design optimization simulated here mainly consider the personnel assignment of debuggers and the software release time. All the following simulations replicate 100 times. Similar as Section III A, Parameters used in this section can be obtained according to experts' estimation through Delphi method. Here these parameters are directly given for the convenience of calculation.

A. THE OPTIMIZATION OF DEBUGGERS ASSIGNMENT

First the optimizing for debuggers assignment is discussed. Generally, there are two situations that need to be optimized. The first is that debuggers of one type performs absolutely better than debuggers of the other type. In this case, the optimization problem mainly comes from the high labor cost of the better debuggers. The second situation is that different debugger types have different strengths respectively. Then, the main job of optimization is to make the best use of their abilities. The two situations are shown as follows.

1) SITUATION 1

In this situation, debuggers of one type have better abilities both in detection and correction than the other type

In this Situation, the parameters for the testing process are set as $a = 100$, $b = 0.02\mathrm{SN}(1) + 0.01\mathrm{SN}(2)$, $\mu(1) = 0.2$, $\mu(2) = 0.1$, $\gamma(1) = 0.1$ and $\gamma(2) = 0.15$. From the parameter setting we can see that debuggers of type 1 performs better than of type 2, so the labor cost for debuggers of type 1 should be higher than labor cost for type 2. The parameters for the cost calculation are configured as $Co_f = 15$, $Co_{l1} = 4$, $Co_{l2} = 2$, $Co_b = 2$ and $Co_e = 50$. Specifically, $Co_{l1} = 4$ and $Co_{l2} = 2$ indicate that the labor cost for a type-1 debugger is twice the labor cost for a type-2 debugger. According to Equation (17), the total cost can be expressed as below.

$$C = 15t + 4 \times \mathrm{SN}(1) \times t + 2 \times \mathrm{SN}(2) \times t + 2 \times m_c(t) + 50 \times (100 + m_d(t) - m_c(t)) \quad (25)$$

The testing process is simulated in the time interval of (0, 100]. From the simulation results, we can have the average cumulative number of faults corrected and introduced, as illustrated in Section III. Then, according to equation (25), we can finally calculate the testing cost for the software. For this example, we vary the debugger numbers of two different types respectively from 1 to 10 and compare the corresponding costs, as shown in Figure 3.

According to our results, the cost reaches its minimum, 4245.68, when there are 5 debuggers of type 1 and 2 debuggers of type 2 for the software testing in this case, as shown in Figure 3. From the shape of the figure, it can also be seen that the cost generally first decreases and then increases with the number of debuggers of one type when the number of the other type is fixed.

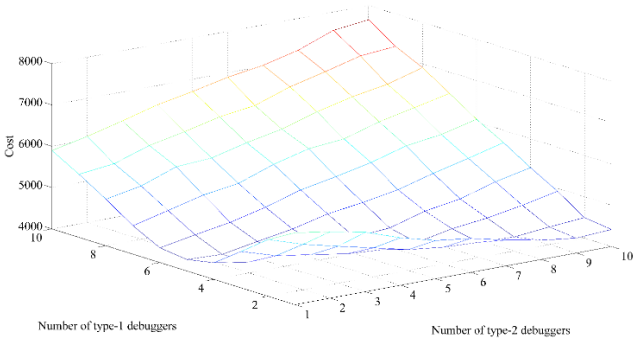

FIGURE 3. Costs for different numbers of debuggers of type 1 and type 2 in case 1.

2) SITUATION 2

In this situation, debuggers of one type have better ability in detection but weaker ability in correction than the other type.

For this situation, the parameters for the testing process are set as $a = 100$, $b = 0.05\mathrm{SN}(1) + 0.01\mathrm{SN}(2)$, $\mu(1) = 0.15$, $\mu(2) = 0.2$, $\gamma(1) = 0.1$ and $\gamma(2) = 0.15$. From the parameter setting we can see that debuggers of type 1 performs better in faults detection while debuggers of type 2 performs better in faults correction. The parameters for the cost calculation are configured as $Co_f = 15$, $Co_{l1} = 2$, $Co_{l2} = 3$, $Co_b = 2$ and $Co_e = 110$. It should be noted that compared with the cost parameters in case 1, we reduced the labor cost gap between the two types because each type has its own advantage. According to Equation (17), the total cost can be expressed as below.

$$C = 15t + 2 \times \mathrm{SN}(1) \times t + 3 \times \mathrm{SN}(2) \times t + 2 \times m_c(t) + 110 \times (100 + m_d(t) - m_c(t)) \quad (26)$$

The testing process is also simulated in the time interval of (0, 100]. As in case 1, we can finally have the testing cost for the software according to equation (26). Figure 4 shows the results for costs of the different debugger numbers of two different types respectively from 1 to 10.

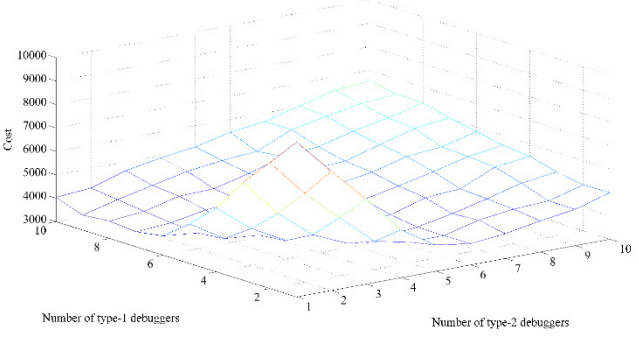

FIGURE 4. Costs for different numbers of debuggers of type 1 and type 2 in case 2.

According to our results, the cost reaches its minimum, 3553.44, when there are 6 debuggers of type 1 and 3 debuggers of type 2 for the software testing in this case, as shown

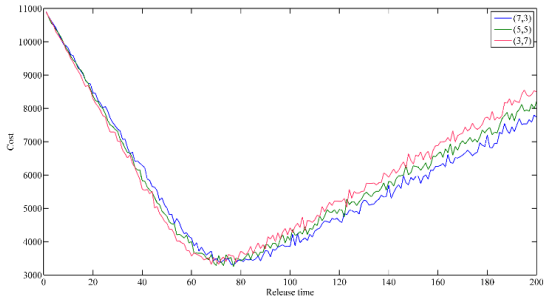

FIGURE 5. Cost for three different pairs of debuggers in the release time interval of [1, 200].

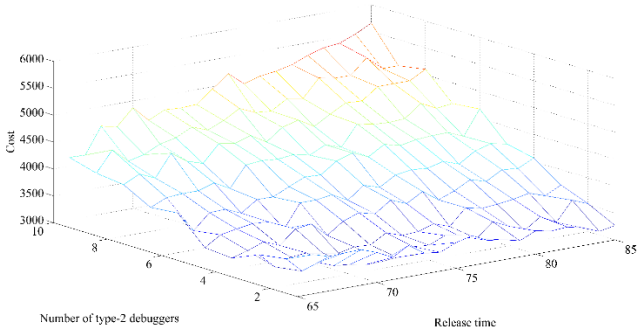

FIGURE 6. Costs for different number of type-2 debuggers and software release time with 9 debuggers of type 1.

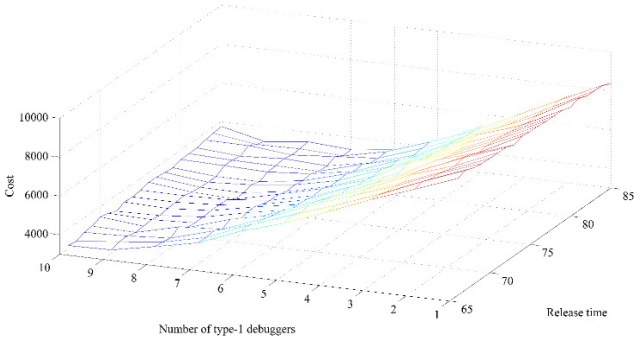

FIGURE 7. Costs for different number of type-1 debuggers and software release time with 2 debuggers of type 2.

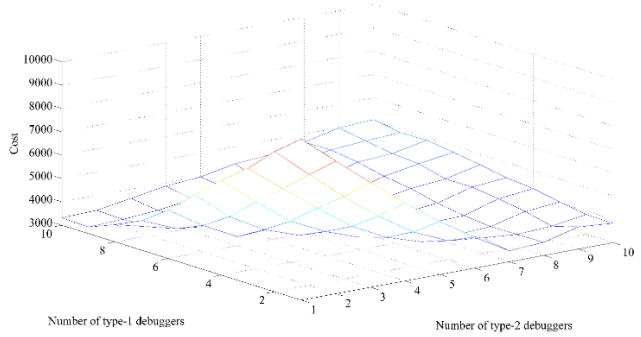

FIGURE 8. Costs for different numbers of debuggers of type 1 and type 2 with the release time of 77.

in Figure 4. From the shape of the figure, it also can be seen that the cost generally first decreases and then increases with the number of debuggers of one type when the number of debuggers of the other type is fixed.

B. THE OPTIMIZATION OF RELEASE TIME

In this example, we consider the cost model with the same parameters used in case 2 in section IV A. For illustration, three pairs of debuggers were selected for the software testing and the software release time was varied from 1 to 200 unit. The results of costs for different pairs of debuggers and different release time are shown in Figure 5, where (x, y) in the legend indicates that employing x debuggers of type 1 and y debuggers of type 2.

It can be seen from Figure 5 that all the costs for three pairs of debuggers first decrease before 75 or thereabout and then turn to increase. Specifically, the cost for debuggers of (7, 3) is the highest during the decreasing phase while the cost for debuggers of (3, 7) is the highest during the increasing phase. As known in case 2 in section IV A, debuggers of type 1 performs better in fault detection and debuggers of type 2 performs better in fault correction. Thus, we can guess that the fault detection is more important in the beginning of the test but the fault correction is becoming increasingly important as there will be fewer faults to be detected with the time going.

C. THE OPTIMIZATION OF DEBUGGERS ASSIGNMENT AND RELEASE TIME

In this example, we simultaneously optimize the debugger assignment and release time. In order to compare conveniently, the parameters used here are the same with the example in section IV B. According to our results, the optimal testing scheme should last 77 units of time and employ 9 debuggers of type 1 and 2 debuggers of type 2. In this case, the cost of testing is least, which is only 3335.82. To illustrate this situation, we give three separate figures where the numbers of type-1 and type-2 debuggers as well as the software release time are fixed respectively, as shown in Figures 6-8.

As shown in Figure 4 and 5, the cost usually reaches its minimum when the debugger numbers of two types lie between 1 and 10, and the release time lies between 70 and 80. Thus, in this example, to present more details, the debugger number of each type is varied from 1 to 10 and the release time lays within the range of (65, 85).

V. CONCLUSION

This paper simulates the software fault detection, correction and introduction processes considering the impact of debuggers' different contributions on the fault detection rate, the different fault correction rate, and the different fault introduction rate of different debuggers. The whole procedure is simulated as a queuing process adopting the Simulink toolbox of Matlab. Based on this procedure, the calculation method of total cost, the main target of the testing design optimizing, is also proposed. The application of the proposed approach is illustrative under certain assumptions of the parameters' values. The procedure simulation results show that more

debuggers not only contribute more to fault detection rate but also correct faults faster and introduce more faults. The testing design optimization results show that in some practical situations, reasonably scheduling the debugger types and software release time can help to reduce the total cost of software testing. The results implies us that companies should consider the benefits and the costs of the debuggers and the release time in order to maximize their profit.

There are several directions that can be further investigated based on this study: First, the optimal combination of different types of debuggers and the optimal software release time, considering the balance between the software reliability and the cost function, can be further studied. Second, it would be interesting to consider the fault dependency between different faults; Say, some faults may be dependent on other faults, so that they become detectable only upon correction of other faults. Last, it is also meaningful to consider that the testing process of software can be combined with the testing process of hardware that supports this software.

REFERENCES

[1] S. Martinez-Fernandez, A. M. Vollmer, A. Jedlitschka, X. Franch, L. Lopez, P. Ram, P. Rodriguez, S. Aaramaa, A. Bagnato, M. Choras, and J. Partanen, "Continuously assessing and improving software quality with software analytics tools: A case study," *IEEE Access*, vol. 7, pp. 68219–68239, 2019, doi: 10.1109/ACCESS.2019.2917403.

[2] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidovic, and R. Kroeger, "Measuring the impact of code dependencies on software architecture recovery techniques," *IEEE Trans. Softw. Eng.*, vol. 44, no. 2, pp. 159–181, Feb. 2018, doi: 10.1109/TSE.2017.2671865.

[3] J. Borstler and B. Paech, "The role of method chains and comments in software readability and comprehension—An experiment," *IEEE Trans. Softw. Eng.*, vol. 42, no. 9, pp. 886–898, Sep. 2016, doi: 10.1109/TSE.2016.2527791.

[4] E. Dilorenzo, E. Dantas, M. Perkusich, F. Ramos, A. Costa, D. Albuquerque, H. Almeida, and A. Perkusich, "Enabling the reuse of software development assets through a taxonomy for user stories," *IEEE Access*, vol. 8, pp. 107285–107300, 2020, doi: 10.1109/ACCESS.2020.2996951.

[5] A. Baabad, H. B. Zulzalil, S. Hassan, and S. B. Baharom, "Software architecture degradation in open source software: A systematic literature review," *IEEE Access*, vol. 8, pp. 173681–173709, 2020, doi: 10.1109/ACCESS.2020.3024671.

[6] S. R. Aziz, T. A. Khan, and A. Nadeem, "Efficacy of inheritance aspect in software fault prediction—A survey paper," *IEEE Access*, vol. 8, pp. 170548–170567, 2020, doi: 10.1109/ACCESS.2020.3022087.

[7] K. Gao, R. Peng, L. Qu, and S. Wu, "Jointly optimizing lot sizing and maintenance policy for a production system with two failure modes," *Rel. Eng. Syst. Saf.*, vol. 202, Oct. 2020, Art. no. 106996.

[8] F. M. Favará, D. W. Jackson, J. H. Saleh, and D. N. Mavris, "Software contributions to aircraft adverse events: Case studies and analyses of recurrent accident patterns and failure mechanisms," *Rel. Eng. Syst. Saf.*, vol. 113, pp. 131–142, May 2013.

[9] H. Pham, *System Software Reliability*. London, U.K.: Springer-Verlag, 2006.

[10] K. Song, I. Chang, and H. Pham, "A software reliability model with a weibull fault detection rate function subject to operating environments," *Appl. Sci.*, vol. 7, no. 10, p. 983, Sep. 2017.

[11] K. Gao, X. Yan, R. Peng, and L. Xing, "Economic design of a linear consecutively connected system considering cost and signal loss," *IEEE Trans. Syst., Man, Cybern. Syst.*, early access, Oct. 24, 2019, doi: 10.1109/TSMC.2019.2946195.

[12] K. Gao, X. Yan, X.-D. Liu, and R. Peng, "Object defence of a single object with preventive strike of random effect," *Rel. Eng. Syst. Saf.*, vol. 186, pp. 209–219, Jun. 2019.

[13] R. Peng, D. Wu, H. Xiao, L. Xing, and K. Gao, "Redundancy versus protection for a non-reparable phased-mission system subject to external impacts," *Rel. Eng. Syst. Saf.*, vol. 191, Nov. 2019, Art. no. 106556.

[14] K. Yi, G. Kou, K. Gao, and H. Xiao, "Redundancy versus protection for a non-reparable phased-mission system subject to external impacts," *J. Risk Reliab.*, vol. 235, no. 1, pp. 50–62, Feb. 2021.

[15] M. Xie, Q. P. Hu, Y. P. Wu, and S. H. Ng, "A study of the modeling and analysis of software fault-detection and fault-correction processes," *Qual. Rel. Eng. Int.*, vol. 23, no. 4, pp. 459–470, 2007.

[16] 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 Trans. Rel.*, vol. 56, no. 4, pp. 629–642, Dec. 2007.

[17] R. Peng, Y. F. Li, and Y. Liu, *Software Fault Detection and Correction: Modeling and Applications*. Singapore: Springer, 2018.

[18] M. Zhu and H. Pham, "A two-phase software reliability modeling involving with software fault dependency and imperfect fault removal," *Comput. Lang., Syst. Struct.*, vol. 53, pp. 27–42, Sep. 2018.

[19] J. Yang, Y. Liu, M. Xie, and M. Zhao, "Modeling and analysis of reliability of multi-release open source software incorporating both fault detection and correction processes," *J. Syst. Softw.*, vol. 115, pp. 102–110, May 2016.

[20] X. Xiao, H. Okamura, and T. Dohi, "NHPP-based software reliability models using equilibrium distribution," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. E95.A, no. 5, pp. 894–902, 2012.

[21] C.-T. Lin and C.-Y. Huang, "Staffing level and cost analyses for software debugging activities through rate-based simulation approaches," *IEEE Trans. Rel.*, vol. 58, no. 4, pp. 711–724, Dec. 2009.

[22] R. Peng and F. R. Shahrzad, "Simulation of software fault detection and correction processes considering different skill levels of debuggers," in *Proc. IEEE 20th Pacific Rim Int. Symp. Dependable Comput.*, Nov. 2014, pp. 157–158.

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

[24] Q. Li and H. Pham, "NHPP software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage," *Appl. Math. Model.*, vol. 51, pp. 68–85, Nov. 2017.

[25] K. Shibata, K. Rinsaka, and T. Dohi, "M-SRAT: Metrics-based software reliability assessment tool," *Int. J. Perform. Eng.*, vol. 11, pp. 369–379, Dec. 2015.

[26] M. Bisi and N. Goyal, "Early prediction of software fault-prone module using artificial neural network," *Int. J. Perform. Eng.*, vol. 11, pp. 43–52, Dec. 2015.

[27] X. Xiao and T. Dohi, "On the role of weibull-type distribu-tions in NHPP-based software reliability modeling," *Int. J. Perform. Eng.*, vol. 9, pp. 123–132, Mar. 2013.

[28] D. Pfeifer, "The structure of elementary pure birth processes," *J. Appl. Probab.*, vol. 19, no. 3, pp. 664–667, Sep. 1982.

[29] J. G. Shanthikumar, "A general software reliability model for performance prediction," *Microelectron. Rel.*, vol. 21, no. 5, pp. 671–682, Jan. 1981.

[30] Q. P. Hu, M. Xie, S. H. Ng, and G. Levitin, "Robust recurrent neural network modeling for software fault detection and correction prediction," *Reliab. Eng. Syst. Safe.*, vol. 92, pp. 332–340, 2007.

[31] N. Karunanithi, D. Whitley, and Y. K. Malaiya, "Prediction of software reliability using connectionist models," *IEEE Trans. Softw. Eng.* vol. 18, no. 7, pp. 563–574, Jul. 1992.

[32] R. Sitte, "Comparison of software-reliability-growth predictions: Neural net-works vs. Parametric-recalibration," *IEEE Trans. Rel.*, vol. 48, no. 3, pp. 285–291, Sep. 1999.

[33] B. Yang, X. Li, M. Xie, and F. Tan, "A generic data-driven software reliability model with model mining technique," *Rel. Eng. Syst. Saf.*, vol. 95, no. 6, pp. 671–678, Jun. 2010.

[34] M. R. Lyu, *Handbook of Software Reliability Engineering*. New York, NY, USA: McGraw-Hill, 1996.

[35] R. C. Tausworthe and M. R. Lyu, "A generalized technique for simulating software reliability," *IEEE Softw.*, vol. 13, no. 2, pp. 77–88, Mar. 1996.

[36] S. S. Gokhale and M. Rung-Tsong Lyu, "A simulation approach to structure-based software reliability analysis," *IEEE Trans. Softw. Eng.*, vol. 31, no. 8, pp. 643–656, Aug. 2005.

[37] C.-T. Lin, "Analyzing the effect of imperfect debugging on software fault detection and correction processes via a simulation framework," *Math. Comput. Model.*, vol. 54, nos. 11–12, pp. 3046–3064, Dec. 2011.

[38] H. Pham and X. Zhang, "NHPP software reliability and cost models with testing coverage," *Eur. J. Oper. Res.*, vol. 145, no. 2, pp. 443–454, Mar. 2003.

[39] 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.
[40] K. Song, I. Chang, and H. Pham, "Optimal release time and sensitivity analysis using a new NHPP software reliability model with probability of fault removal subject to operating environments," *Appl. Sci.*, vol. 8, no. 5, p. 714, May 2018.
[41] K. Song, I. Chang, and H. Pham, "An NHPP software reliability model with S-Shaped growth curve subject to random operating environments and optimal release time," *Appl. Sci.*, vol. 7, no. 12, p. 1304, Dec. 2017.
[42] J. D. Musa and K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement," in *Proc. Int. Conf. Softw. Eng.*, 1948, pp. 230–238.
[43] A. L. Goel and K. Okumoto, "Time-dependent error detection rate model for software reliability and other performance measures," *IEEE Trans. Rel.*, vol. R-28, no. 3, pp. 206–211, Aug. 1979.
[44] L. Kuo and T. Y. Yang, "Bayesian computation for nonhomogeneous Poisson processes in software reliability," *J. Amer. Stat. Assoc.*, vol. 91, no. 434, pp. 763–773, Jun. 1996.
[45] P. A. W. Lewis and G. S. Shedler, "Simulation of nonhomogeneous Poisson processes by thinning," *Nav. Res. Logistics Quart.*, vol. 26, no. 3, pp. 403–413, Sep. 1979.
[46] J. P. Klein, A. P. Kragh, and K. Niels, "Exponential distribution," *A Primer Stat. Distrib.*, vol. 2, pp. 1–13, Dec. 1996.
[47] R. D. Gupta and D. Kundu, "Theory & methods: Generalized exponential distributions," *Austral. New Zealand J. Stat.*, vol. 41, pp. 173–188, Mar. 2015.
[48] H. Xiao and S. Gao, "Simulation budget allocation for selecting the top-M designs with input uncertainty," *IEEE Trans. Autom. Control*, vol. 63, no. 9, pp. 3127–3134, Sep. 2018.
[49] H. Xiao, H. Chen, and L. H. Lee, "An efficient simulation procedure for ranking the top simulated designs in the presence of stochastic constraints," *Automatica*, vol. 103, pp. 106–115, May 2019.
[50] H. Xiao, L. H. Lee, D. Morrice, C.-H. Chen, and X. Hu, "Ranking and selection for terminating simulation under sequential sampling," *IISE Trans.*, vol. 29, pp. 1–16, Aug. 2020.
[51] J. Guo, Z. Li, and T. Keyser, "A Bayesian approach for integrating multilevel priors and data for aerospace system reliability assessment," *Chin. J. Aeronaut.*, vol. 31, no. 1, pp. 41–53, Jan. 2018.
[52] G. Wu, Z. Li, and P. Liu, "Risk-informed reliability improvement optimization for verification and validation planning based on set covering modeling," *Proc. Inst. Mech. Eng., O, J. Risk Rel.*, vol. 1, Feb. 2020, Art. no. 1748006X1989829.
[53] J. Guo, Z. Li, and J. Jin, "System reliability assessment with multilevel information using the Bayesian melding method," *Rel. Eng. Syst. Saf.*, vol. 170, pp. 146–158, Feb. 2018.
[54] H. Xiao, M. Cao, and R. Peng, "Artificial neural network based software fault detection and correction prediction models considering testing effort," *Appl. Soft Comput.*, vol. 94, Sep. 2020, Art. no. 106491.

KAIYE GAO (Member, IEEE) received the Ph.D. degree in management science and engineering from the University of Science and Technology Beijing, in 2018.

He is currently an Associate Professor with the School of Economics and Management, Beijing Information Science and Technology, China, and a Postdoctoral Researcher with the Academy of Mathematics and System Science, Chinese Academy of Sciences. From 2016 to 2017, he visited the Department of Mathematics, The University of Manchester. He has about 40 published or accepted papers in journals and conference proceedings, such as IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS: SYSTEMS, *Reliability Engineering and System Safety*, and *Proceedings of the Institution of Mechanical Engineers—Part O-Journal of Risk and Reliability*. His research interests include system reliability, risk analysis and optimization, maintenance, quality, and health management.