Article citation info:
PENG R, ZHAI Q. Modeling of software fault detection and correction processes with fault dependency. Eksploatacja i Niezawodnosc – Maintenance and Reliability 2017; 19 (3): 467–475, http://dx.doi.org/10.17531/ein.2017.3.18.

Rui PENG
Qingqing ZHAI

MODELING OF SOFTWARE FAULT DETECTION AND CORRECTION PROCESSES WITH FAULT DEPENDENCY

MODELOWANIE PROCESÓW WYKRYWANIA I KOREKCJI BŁĘDÓW OPROGRAMOWANIA Z ZAŁOŻENIEM WZAJEMNEJ ZALEŻNOŚCI BŁĘDÓW

Software reliability modeling has undergone a continuous evolution over the past three decades to adapt to various and ever-changing software testing environments. In existing models, immediate fault removal and fault independency are two basic and commonly used assumptions. Recently, models combining fault detection process (FDP) and fault correction process (FCP) were proposed to alleviate the immediate fault removal assumption. In this paper, we extend such a methodology by proposing a modeling framework for the FDP and FCP incorporating fault dependency. Faults are classified as leading faults and dependent faults and the FCPs for both types of faults are explicitly modeled. Several paired models considering different assumptions for debugging lags are proposed for the combined FDP and FCP. The applicability of the proposed models are illustrated using real testing data. In addition, the optimal software release policy under this framework is studied.

Keywords: *fault dependency, non-homogeneous Poisson process (NHPP), software reliability growth model (SRGM), software fault detection and correction processes.*

Modelowanie niezawodności oprogramowania w ciągu ostatnich trzech dekad ulegało ciągłej ewolucji, pozwalającej dostosować je do różnych, stale zmieniających się środowisk testowych. W przypadku istniejących modeli, dwoma podstawowymi i powszechnie stosowanymi założeniami jest natychmiastowe usunięcie błędu oraz brak zależności między błędami. Ostatnio, badacze zaproponowali modele, które łagodzą pierwsze z tych założeń, łącząc proces wykrywania błędów (FDP) z procesem ich korekcji (FCP). W niniejszym artykule, rozszerzono tę metodologię, proponując paradygmat modelowania dla zintegrowanych procesów FDP i FCP uwzględniający zależności między błędami. W paradygmacie tym, błędy klasyfikuje się jako błędy nadrzędne i błędy zależne, a procesy FCP dla obu typów błędów są modelowane oddzielnie. Zaproponowano kilka połączonych w pary modeli rozważających różne założenia dotyczące opóźnień debugowania w procesach łączących detekcję i korekcję błędów. Możliwość zastosowania proponowanych modeli przedstawiono na przykładzie rzeczywistych danych testowych. Dodatkowo badano optymalną politykę aktualizacji oprogramowania, jaką można prowadzić w ramach proponowanego paradygmatu.

Słowa kluczowe: *zależność błędów, niejednorodny proces Poissona, model wzrostu niezawodności oprogramowania, procesy detekcji i korekcji błędów oprogramowania*

1. Introduction

Software today plays important roles in almost every section of our society, and the software reliability has been a major concern in many integrated systems [3]. With continuous debugging, analysis and correction, the software reliability will grow gradually with testing [33]. During the past three decades, numerous software reliability growth models (SRGMs) have been proposed [2, 7, 24, 26, 35, 40, 41]. Among these models, Non-homogeneous Poisson Process (NHPP) models are the most commonly accepted [20, 30, 36, 39, 50]. Although NHPP models are mathematically tractable, they are developed under some strong assumptions on the software testing process. Specifically, NHPP models assume immediate fault removal and fault independency. To adapt to different practical software testing environments, generalizations of traditional models by relaxing the assumptions have been proposed [5, 9, 17, 23, 28, 29].

In practical software testing, each detected fault has to be reported, diagnosed, removed and verified before it can be noted as corrected. Consequently, the time spent for fault correction activity is not negligible. In fact, this debugging lag can be an important element in making decisions [16, 49]. Therefore, it is necessary to incorporate the debugging lag into the modeling framework, *i.e.*, to model both the fault detection process (FDP) and fault correction process (FCP). The idea of modeling FCP was first proposed in Schneidewind [34], where a constant lag was used to model the FCP after fault detection. Clearly, the constant correction time assumption is restrictive for various types of faults and different correction profiles. For instance, data collected from practical testing projects show that the correction time can be fitted by the exponential and log-normal distributions [27]. In addition, the correction time may show a growing trend during the whole testing cycle, as later detected faults can be more difficult to correct. Some extensions were made in Lo and Huang [25] and Xie, et al. [44] by incorporating other assumptions of debugging delay. Hu, et al. [8] studied a data-driven artificial neural network model for the prediction of FDP and FCP. [37] used the fault detection/correction profile to quantify the maintainability of software. Some paired FDP and FCP models were proposed in Peng, et al. [31], where testing effort function and fault introduction were included.

Traditional NHPP models assume the statistical independency between successive software failures. Actually, it can hardly be true in practice, as some faults are not detectable until some other fault has been corrected because of logical dependency. Moreover, the common practice of mixing testing strategies can lead to the dependency of failures [6]. With a failure detected, there is a higher chance for another related failure or a cluster of failures to occur in the near future. From this point of view, faults can be classified into mutually independent and dependent types with respect to path-based testing approach. This

issue was addressed in [18], where an extended NHPP SRGM was proposed. Huang and Lin [11] studied the fault detection & correction process considering both fault dependency and debugging lags. Yang, et al. [46] discussed the statistical inference of the software reliability model with fault dependency. However, most of the studies only focus on the FDP, and only the FDP data are used for model parameters estimation. As a result, the collected information from FCP is neglected, which can lead to deficiency in model estimation.

To remedy the problem, we incorporate the fault dependency into the paired FDP and FCP model. Instead of assuming a single type of fault, this study classifies the faults in the testing process into leading faults and dependent faults.The leading faults occurs independently following an NHPP, while the dependent faults are only detectable after the related leading faults being corrected. Different from Huang and Lin [11] which modeled the FDP and the FCP as a single, synthesized fault detection & correction process, we model the FDP and FCP for the leading faults and the dependent faults separately. Subsequently, the FDP&FCP model for the aggregated, observable faults can be readily obtained. With different formulation of debug delays, we can derive various FDP&FCP models. Hence, the proposed models admit a wide applicability that can account for different software reliability growth schemes.

The rest of this paper is organized as follows. Section 2 formulates the general modeling framework of paired FDP and FCP with the incorporation of fault dependency. In Section 3, special paired FDP and FCP models are derived based on different assumptions for debugging lags. In Section 4, the proposed faults are fitted to two real datasets to illustrate the application.Section 5 derives the optimal software release policy under the proposed framework. The conclusion is given in Section 6.

Notation

a	The total number of faults in the software
a_1	The number of leading faults in the software
a_2	The number of dependent faults in the software
p	The ratio of the number of leading faults to the total number of faults
$b(t)$	Fault detection rate function at time t
b	Constant fault detection rate
c	Constant fault correction rate
$\delta(t)$	The time required to correct a fault which is finally corrected at time t
$m_d(t)$	Expected number of faults detected up to time t
$m_r(t)$	Expected number of faults removed up to time t
$m_{d1}(t)$	Expected number of leading faults detected up to time t
$m_{r1}(t)$	Expected number of leading faults removed up to time t
$m_{d2}(t)$	Expected number of dependent faults detected up to time t
$m_{r2}(t)$	Expected number of dependent faults removed up to time t
$\lambda_d(t)$	The intensity function of fault detection process
$\lambda_r(t)$	The intensity function of fault correction process
$\lambda_{d1}(t)$	The intensity function offault detection process for leading faults
$\lambda_{r1}(t)$	The intensity function of fault correction process for leading faults
$\lambda_{d2}(t)$	The intensity function offault detection process for dependent faults
$\lambda_{r2}(t)$	The intensity function of fault correction process for dependent faults

2. The general framework

In this study, we formulate the fault-oriented software testing process as a paired fault detection and correction process. During the test, a fault can only be corrected after being detected. For the faults embedded in the software system, they can be categorized into leading faults and dependent ones. The faults that can be detected and corrected independently are defined as leading faults or independent faults. Other faults that remain undetectable until the corresponding leading faults are removed are defined as dependent faults. Fig.1 illustrates the relationship between leading faults and dependent faults.

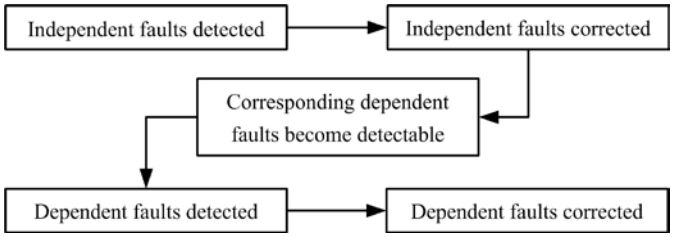

Fig. 1. Relationship of leading faults and dependent faults

Suppose the leading faults are detected and corrected independently. Then, for the leading faults, their detection (FDP_L) and correction process (FCP_L) can be modeled by NHPP models, as in Xie, et al. [44]. For the dependent faults, their detection process (FDP_D) can be modeled as a delayed process of FCP_L, considering that they are only detectable after the corresponding leading faults are corrected. Consequently, the correction process for dependent faults (FCP_D) can be modeled as a delayed process of FDP_D. The modeling framework is characterized by the mean value function for each sub-process.

2.1. Modeling FDP_L

We assume that FDP_L follows a NHPP, and the expected number of leading fault detected during $(t, t+\Delta t]$ is proportional to the number of undetected leading faults at time t. Thus we have:

$$\frac{dm_{d1}(t)}{dt} = b(t)\left(a_1 - m_{d1}(t)\right), \tag{1}$$

where $b(t)$ is the fault detection rate at time t and a_1 is number of leading faults at the beginning. With the initial condition $m_{d1}(t)=0$, it can be derived from (1) that:

$$m_{d1}(t) = a_1\left(1 - \exp\left\{-\int_0^t b(s)\,ds\right\}\right). \tag{2}$$

Different $m_{d1}(t)$ can be obtained based on different $b(t)$. Specially, when $b(t)$ is a constant, we have:

$$m_{d1}(t) = a_1\left(1 - e^{-bt}\right), \tag{3}$$

which is the G-O model [4]. When $b(t) = \dfrac{b^2 t}{1+bt}$, we have:

$$m_{d1}(t) = a_1\left(1 - (1+bt)e^{-bt}\right), \tag{4}$$

which has the same form as the Yamada delayed-S-shaped model [45].

2.2. Modeling FCP_L

FCP_L can be regarded as a delayed process of FDP_L and different models can be used to accommodate the debugging delay. Xie, et al. [44] pointed out that debugging lags could be assumed constant, time dependent or random. If the debugging lag is not random, the FCP_L can be derived from FDP_L as $m_{r1}(t)=m_{d1}(t-\delta(t))$. Otherwise, we have $m_{r1}(t)=E_{\delta(t)}\left[m_{d1}(t-\delta(t))\right]$ if $\delta(t)$ is a random variable. In particular, if the debugging lag is assumed to be an exponentially distributed random variable, i.e., $\delta(t)\sim Exp(c)$, we have:

$$m_{r1}(t)=c\int_0^t m_{d1}(t-s)\exp\{-cs\}\,ds. \tag{5}$$

Taking the derivatives of both sides with respect to t, we can obtain that:

$$\lambda_{r1}(t)=c\left(m_{d1}(t)-m_{r1}(t)\right). \tag{6}$$

This implies that the expected number of faults corrected during $(t,t+\Delta t]$ is proportional to the number of detected but uncorrected faults at time t. We call c the fault correction rate.

2.3. Modeling FDP_D and FCP_D

For these dependent faults, they can only be detected after the corresponding leading faults are removed. Hence, the proportion of the detectable dependent faults in the dependent faults is equal to the proportion of the corrected leading faults in the leading faults. Suppose the number of dependent faults is a_2. Then, the expected number of detectable dependent faults is $a_2 m_{r1}(t)/a_1$ up to time t. Furthermore, because leading faults and dependent faults are detected under the same testing environment, it is reasonable to assume that the fault detection rate for dependent faults is the same as the fault detection rate for leading faults. Therefore:

$$\frac{dm_{d2}(t)}{dt}=b(t)\left(\frac{a_2 m_{r1}(t)}{a_1}-m_{d2}(t)\right). \tag{7}$$

With the initial condition $m_{d2}(0)=0$, we can derive from (7) based on $m_{r1}(t)$ and $b(t)$ that:

$$m_{d2}(t)=\frac{a_2}{a_1}m_{r1}(t)-\frac{a_2}{a_1}\exp\left\{-\int_0^t b(s)\,ds\right\}\int_0^t \lambda_{r1}(s)\exp\left\{\int_0^s b(u)\,du\right\}ds. \tag{8}$$

Particularly, when $b(t)=b$, we have:

$$m_{d2}(t)=\frac{a_2}{a_1}m_{r1}(t)-\frac{a_2}{a_1}\exp\{-bt\}\int_0^t \lambda_{r1}(s)\exp\{bs\}\,ds. \tag{9}$$

Based on the detection process of dependent faults, the corresponding correction process can be obtained as a delayed process as for leading faults. Thus, with different assumptions for the debugging delay, $m_{r2}(t)$ of FCP_D can be derived accordingly.

2.4. Combined models

With the FDP and FCP models for both kinds of faults, the aggregated model for the paired FDP&FCP can be readily obtained:

$$a=a_1+a_2, \tag{10}$$

$$m_d(t)=m_{d1}(t)+m_{d2}(t), \tag{11}$$

$$m_r(t)=m_{r1}(t)+m_{r2}(t). \tag{12}$$

3. Specific models for dependent FDP and FCP

In this section, we consider the widely-used constant fault detection rate function $b(t)$, i.e., $b(t)=b$ [10, 22]. In this case, we have $m_{d1}(t)=ap\left(1-e^{-bt}\right)$ from (3), where $p=a_1/a$ is the proportion of leading faults. As stated, different $m_{r1}(t)$ can be derived based on different assumptions on the debugging lag. Moreover, as long as $m_{r1}(t)$ being specified, $m_{d2}(t)$ can be obtained according to (9). In the following, we consider three different types of debugging lags, which have been observed from practical testing processes. Correspondingly, specific paired PDF&FCP models are derived.

3.1. Constant debugging lag

We first consider the case where the correction of each fault takes the same time, i.e., $\delta(t)=\delta$. Then, the FCP model of leading faults is:

$$m_{r1}(t)=\begin{cases}0, t<\delta\\ ap\left(1-e^{-b(t-\delta)}\right), t\ge\delta\end{cases}. \tag{13}$$

Consequently, the FDP model for dependent faults can be derived according to (9):

$$m_{d2}(t)=\begin{cases}0, t<\delta\\ a(1-p)\left(1-\left(1+b(t-\delta)\right)e^{-b(t-\delta)}\right), t\ge\delta\end{cases}. \tag{14}$$

Based on the FDP models of the leading faults and the dependent faults, $m_d(t)$ for the aggregated FDP is obtained as:

$$m_d(t)=\begin{cases}ap\left(1-e^{-bt}\right), t<\delta\\ ap\left(1-e^{-bt}\right)+a(1-p)\left(1-\left(1+b(t-\delta)\right)e^{-b(t-\delta)}\right), t\ge\delta\end{cases}. \tag{15}$$

Because the FCP models for both kinds of faults are modeled as delayed FDP, the aggregated FCP model is:

$$m_r(t)=\begin{cases}0, t<\delta\\ ap\left(1-e^{-b(t-\delta)}\right), \delta\le t<2\delta\\ ap\left(1-e^{-b(t-\delta)}\right)+a(1-p)\left(1-\left(1+b(t-2\delta)\right)e^{-b(t-2\delta)}\right), t\ge 2\delta\end{cases} \tag{16}$$

3.2. Time-dependent Debugging Lag

In practice, the faults discovered in the later phase of the testing process may be more difficult to correct. To model such a phenomenon, we assume the debugging lag is dependent on the testing time, $\delta(t)=\frac{\ln(1+\gamma t)}{b}$, where $0<\gamma<b$. Accordingly, the FCP models for the

two kinds of faults are $m_{ri}(t)=m_{di}\left(t-\frac{\ln(1+\gamma t)}{b}\right), i=1,2$. Under this assumption, we have:

$$m_{r1}(t)=m_{d1}\left(t-\frac{\ln(1+\gamma t)}{b}\right)=ap\left(1-(1+\gamma t)e^{-bt}\right), \quad (17)$$

which is a general form of the delayed NHPP model [45].

Based on (9) and (17), $m_{d2}(t)$ can be derived. Then, $m_d(t)$ for the aggregated FDP is obtained as:

$$m_d(t)=a\left(1-e^{-bt}\right)-a(1-p)\left(bt+\frac{b\gamma t^2}{2}\right)e^{-bt}. \quad (18)$$

Because $m_r(t)=m_d\left(t-\frac{\ln(1+\gamma t)}{b}\right)$, the model for the aggregated FCP can be derived as follows:

$$m_r(t)=a\left(1-(1+\gamma t)e^{-bt}\right) -a(1-p)(1+\gamma t)\left(bt-(1+\gamma t)\ln(1+\gamma t)+\frac{b\gamma t^2}{2}+\frac{\gamma\ln^2(1+\gamma t)}{2b}\right)e^{-bt}. \quad (19)$$

3.3. Exponentially distributed random debugging lag

As obtained in Section 2.2, the number of faults corrected during time interval $(t,t+\Delta t]$ in this case is proportional to the number of detected but uncorrected faults at time t. Based on (5), $m_{r1}(t)$ can be obtained as:

$$m_{r1}(t)=\begin{cases} ap\left(1-(1+bt)e^{-bt}\right), c=b \\ ap\left(1+\frac{be^{-ct}-ce^{-bt}}{c-b}\right), c\neq b \end{cases}. \quad (20)$$

Then, $m_{d2}(t)$ can be derived based on $m_{r1}(t)$ according to (9). As the summation of $m_{d1}(t)$ and $m_{d2}(t)$, $m_d(t)$ for the aggregated FDP is thus obtained:

$$m_d(t)=\begin{cases} a\left(1-e^{-bt}\right)-a(1-p)\left(bt+\frac{b^2t^2}{2}\right)e^{-bt}, c=b \\ a\left(1-e^{-bt}\right)-a(1-p)\left(\frac{bcte^{-bt}}{c-b}+\frac{b^2\left(e^{-ct}-e^{-bt}\right)}{(c-b)^2}\right), c\neq b \end{cases} \quad (21)$$

From $m_{ri}(t)=c\int_0^t m_{di}(t-s)\exp\{-cs\}ds$, we note that $m_r(t)=c\int_0^t m_d(t-s)\exp\{-cs\}ds$ also holds. Therefore, $m_r(t)$ is readily obtained as:

$$m_r(t)=\begin{cases} a\left(1-(1+bt)e^{-bt}\right)-a(1-p)\left(\frac{b^2t^2}{2}+\frac{b^3t^3}{6}\right)e^{-bt}, c=b \\ a\left(1+\frac{be^{-ct}-ce^{-bt}}{c-b}\right)-\frac{abc(1-p)}{(c-b)^2}\left(\frac{(b+c)\left(e^{-ct}-e^{-bt}\right)}{c-b}+bte^{-ct}+cte^{-bt}\right), c\neq b \end{cases} \quad (22)$$

4. Numerical example

In this section, we illustrate the application of the proposed models to two real software testing datasets.

4.1. Description of the Datasets

The first dataset is from the System T1 data of the Rome Air Development Center (RADC) [27]. This dataset is widely used and it contains both fault detection data and fault correction data. The cumulative numbers of detected faults and corrected faults during the first 21 weeks are shown in Table 1. During the debugging, 300.1 hours of computer time were consumed and 136 faults were removed. The computer time spent in the testing process is used the time scale for the FDP and FCP.

Table 1. The dataset from System T1.

Weeks	Computer time (CPU hours)	Cumulative number of detected faults (m_d)	Cumulative number of corrected faults (m_r)
1	4	2	1
2	8.3	2	2
3	10.3	2	2
4	10.9	3	3
5	13.2	4	4
6	14.8	6	4
7	16.6	7	5
8	31.3	16	7
9	56.4	29	13
10	60.9	31	17
11	70.4	42	18
12	78.9	44	32
13	108.4	55	37
14	130.4	69	56
15	169.9	87	75
16	195.9	99	85
17	220.9	111	97
18	252.3	126	117
19	282.3	132	129
20	295.1	135	131
21	300.1	136	136

The second dataset is from the testing process of a middle-size software project [42, 44]. The cumulative numbers of detected faults and corrected faults during the first 17 weeks are listed in Table 2.

Table 2. The dataset from a middle-size software project.

Weeks	Cumulative number of detected faults (m_d)	Cumulative number of corrected faults (m_r)
1	12	3
2	23	3
3	43	12
4	64	32
5	84	53
6	97	78
7	109	89
8	111	98
9	112	107
10	114	109
11	116	113
12	123	120
13	126	125
14	128	127
15	132	127
16	141	135
17	144	143

4.2. Performance analysis

To illustrate our models, we consider the following three paired FDP&FCP models: (1) model with constant debugging lag (abbreviated as M1); (2) model with $\delta(t)=\frac{\ln(1+\gamma t)}{b}$ (abbreviated as M2); and (3) model with exponentially distributed debugging lag (abbreviated M3).

We note that the models proposed in Xie, et al. [44] are special cases of the proposed FCP&FDP models without considering the dependent faults. For comparison purpose, we also fit the data by the three simplified models of M1-M3 with p=1, which are abbreviated as M1', M2' and M3', respectively.

The six models are fitted to the two datasets by the least squares method. The least squares method minimizes the mean squared error (MSE) between the estimated cumulative numbers of detected and corrected faults and the actual cumulative numbers of detected and corrected faults. It is calculated as:

$$\mathrm{MSE}=\frac{1}{2}\left(\mathrm{MSE}_d+\mathrm{MSE}_r\right)=\frac{1}{2n}\sum_{i=1}^{n}\left[\left(m_d(t_i)-m_{d,i}\right)^2+\left(m_r(t_i)-m_{r,i}\right)^2\right], \quad (23)$$

where $m_{d,i}$, $m_{r,i}$ are the observed cumulative numbers of detected faults and corrected faults at time t_i,i=1,…,n. The estimated model parameters for dataset 1 is given in Table 3.

Table 3. The estimated model parameters for dataset 1.

Model	a	b	Remark	MSE
M1	199.27	0.00717	δ=24.78, p=0.3820	9.0114
M1'	507.47	0.00110	δ=25.71	10.8924
M2	182.23	0.00955	γ=0.1599, p=0.1374	15.5697
M2'	1737.64	0.000288	γ=0.0930	26.1383
M3	185.15	0.008456	c=0.03833, p=0.3265	7.8881
M3'	477.75	0.001177	c=0.03786	10.0985

As can be noticed from Table 1, the estimated parameter a (the total number of faults) in the three proposed models M1-M3 are close to each other. They are all close to 188, which is the number of detected faults after three years' testing, as reported in Kapur and Younes [18]. On the contrary, the models M1'-M3', which assume no dependent faults exist, produce quite large a. Therefore, ignoring the dependent faults in the model would result in incorrect total number of faults.

According to the MSE values and the point-wise squared error $\mathrm{MSE}_{d,i}$+$\mathrm{MSE}_{r,i}$ in Fig. 2, it shows that the paired FDP&FCP model with exponentially distributed debugging lag fits the dataset best. On the other hand, the model M1, which assumes constant debugging lag, also provides a competitive fit. The model assuming time-dependent debugging lags provides the least favorable fit. In fact, according to the estimated model M3, we can derive that the expected length of the debugging lag is $\frac{1}{c}=26.08$. This is close to the estimated debugging lag in M1. Thus, we can infer that there are significant debugging lags in the software testing process, and it takes about 25 hours for a detected fault to be corrected.

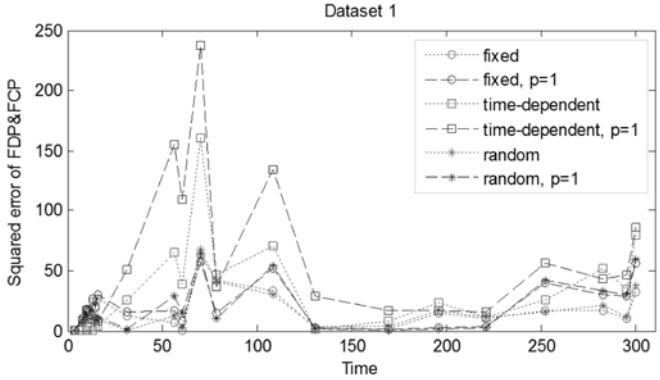

Fig. 2. Point-wise squared errors of the six fitted models for dataset 1.

The estimation results by the six models for dataset 2 are presented in Table 4. Analogous to the dataset 1, the proposed models considering both leading and dependent faults are superior to those

Table 4. The estimated model parameters for dataset 2.

Model	a	b	Remark	MSE
M1	144	0.3058	δ=1.51,p=0.474	39.5732
M1'	153.01	0.1487	δ=1.94	41.0015
M2	144	0.3938	γ=0.3112,p=0.0448	49.9352
M2'	168.36	0.1193	γ=0.2339	104.8889
M3	144	0.3354	c=0.7281,p=0.3551	47.0471
M3'	156.35	0.1404	c=0.5811	55.1920

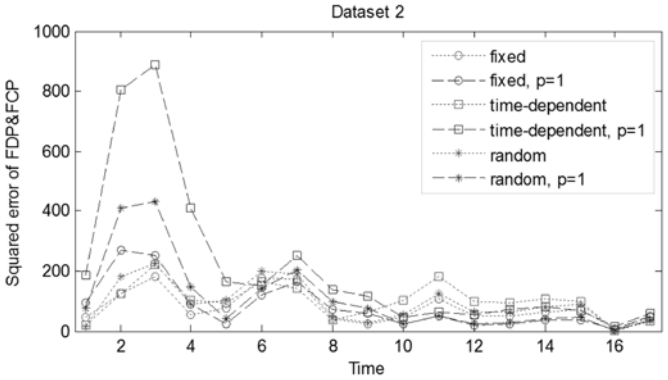

Fig. 3. Point-wise squared errors of the six fitted models for dataset 2

only considering leading faults. In the fitting procedure, we restrict the total number of faults a to be no smaller than the faults in the data. Therefore, we see that the estimated a are all equal to 144, which is the number of the total faults in dataset 2. Among the three models M1-M3, the constant debugging lag model provides the best fit. This can also be noted from the point-wise squared error in Fig. 3. This indicates the debugging lag is almost constant in the software testing process.

5. Software release policy

Based on a SRGM, useful information can be inferred to guide decision-making. For software projects, one critical decision is to determine the optimal release time [12]. Many studies have dealt with this problem [13, 19, 21, 32]; see Jain and Priya [14] and Boland and Chuív [1] for an overview. As cost and reliability requirements are of great concern, they are often used as objectives for optimizing the testing time and release policy [15, 38, 47, 48]. In this section, we study the optimal release policies based on the proposed models from the cost and reliability perspectives.

5.1. Software release policy based on reliability criterion

Software is usually released when a reliability target is achieved. It is reasonable to stop testing when a pre-specified proportion of faults are removed. We use T to denote the length of testing and consider the ratio of cumulative removed faults to the initial faults in the software system as the reliability criterion [11]:

$$R_1(T)=\frac{m_r(T)}{a}. \tag{24}$$

With a given reliability target R_1, the time to reach this reliability target is

$$T_1=m_r^{-1}(a\cdot R_1). \tag{25}$$

Another criterion is the software reliability, which is defined as the probability that no failure occurs during time interval $(T,T+\Delta T]$ given that the software is released at time T. Considering that the reliability status of software generally does not change in operational phase, the reliability function is:

$$R_2(\Delta T|T)=\exp\left[-\lambda_d(T)\Delta T\right], \tag{26}$$

where $\lambda_d(T)$ is the instantaneous failure intensity at time T. For a given target R_2 for $R_2(\Delta T \mid T)$, the time for the software to reach R_2 can be solved as $\min_T\{T:R_2(\Delta T \mid T)\geq R_2\}$.

5.2. Software release policy based on cost criterion

For a basic FDP model with mean value function $m(t)$, the following cost model is frequently used [43]:

$$C(T)=c_1m(T)+c_2\left(m(\infty)-m(T)\right)+c_3T, \tag{27}$$

where c_1 is the expected cost of removing a fault during testing, c_2 is the expected cost of removing a fault in the field and c_3 is the expected cost per unit time of testing.In practice, the cost of removing a fault in field is generally greater than that during testing, thus we assume $c_2>c_1$.

When the correction process is incorporated, the following cost model can be constructed:

$$C(T)=c_1m_r(T)+c_2\left(m_d(\infty)-m_r(T)\right)+c_3T, \tag{28}$$

where $m_r(T)$ is the total number of corrected faults at the time of release T, and $m_d(\infty)-m_r(T)$ is the number of uncorrected faults that includes both the undetected faults $m_d(\infty)-m_d(T)$ and the detected-but-not-corrected faults $m_d(T)-m_r(T)$. By minimizing the cost model with respect to T, the optimal release time T_c under the proposed framework can be obtained.

Theorem 1: Under the proposed models in Section 3, the time T_c which minimizes $C(T)$ exists. Specifically, there exist $2k(k\geq 0)$ non-negative numbers $0<z_1\leq z_2\leq\ldots\leq z_{2k}<+\infty$ which satisfy that $C(T)$ increases on $\left[z_{2j},z_{2j+1}\right)$ and decreases on $\left[z_{2j+1},z_{2j+2}\right)$ with $j=0,\ldots,k$, $z_0=0$ and $z_{2k+1}=+\infty$. The optimal T_c is determined as $T_c=\arg\min_{T\in\{0,z_2,\ldots,z_{2k}\}}C(T)$.

Proof: We just need to prove that there exists a T_s such that $C'(T)$ is positive for $T>T_s$. Since $C(T)=c_1m_r(T)+c_2\left(m_d(\infty)-m_r(T)\right)+c_3T$, we have:

$$C'(T)=c_3-(c_2-c_1)\lambda_r(T). \tag{29}$$

Clearly, $C'(0)=c_3>0$, indicating that $C(T)$ is increasing when T is close to zero. We shall prove that $\lambda_r(T)$ approaches 0 (or $C'(T)$ approaches c_3) when T approaches $+\infty$. If so, $C(T)$ is increasing when T is close to 0 or approaches $+\infty$. Consequently, if $C(T)$ has any stationary point, it must have even number of stationary points $0<z_1\leq z_2\leq\cdots\leq z_{2k}<+\infty$ such that $C(T)$ increases on on $\left[z_{2j},z_{2j+1}\right)$ and decreases on $\left[z_{2j+1},z_{2j+2}\right)$ for $j=0,\ldots,k$, $z_0=0$ and $z_{2k+1}=+\infty$. In the following, we shall show that $\lambda_r(T)$ approaches 0 when T approaches $+\infty$ under the three proposed models.

If the paired model under constant debugging lag assumption is used, from (16) we have:

$$\lambda_r(T)=pabe^{-b(T-\delta)}+ab(1-p)(bT-2b\delta)e^{-b(T-2\delta)},T\geq 2\delta. \tag{30}$$

When T approaches $+\infty$, $\lambda_r(T)$ approaches 0.

For the paired model with time-dependent debugging lags, according to (18) we have:

$$\lambda_d(T)=abe^{(-bT)}-a(1-p)(b+bcT-b^2T-(b^2cT^2)/2)e^{(-bT)}. \tag{31}$$

It can be seen that $\lambda_d(T)$ approaches 0 when T approaches $+\infty$. Moreover, we have:

$$\lambda_r(T)=\lambda_d\left(T-\frac{1}{b}\ln(1+\gamma T)\right)\left(1-\frac{\gamma}{b(1+\gamma T)}\right).$$

As $T-b^{-1}\ln(1+\gamma T)$ approaches $+\infty$ when T approaches $+\infty$ for $b>c$, we can see that $\lambda_r(T)$ approaches 0 for $T\rightarrow+\infty$.

For the paired model under exponentially distributed random debugging lags, we have:

$$\lambda_r(T)=c\left(m_d(T)-m_r(T)\right). \tag{32}$$

Both $m_d(T)$ and $m_r(T)$ approach a as T approaches $+\infty$. Thus $\lambda_r(T)$approaches 0 when T approaches $+\infty$.

5.3. Software release policy based on mixed criterion

When both reliability requirements and the total cost are considered, we determine the optimal release time T^* that minimizes the total cost under the reliability constraint. Accordingly, the problem can be formulated as:

$$\text{Minimize } C(T) = c_1 m_r(T) + c_2\left(m_d(\infty) - m_r(T)\right) + c_3 T$$

$$\text{Subject to } R_1(T) = \frac{m_r(T)}{a} \geq R_1 \text{ (or } R_2(\Delta T|T) = \exp\left[-\lambda_d(T)\Delta T\right] \geq R_2).$$

When the reliability constraint $R_1(T)$ is used, we can divide the time axis $[0,\infty)$ into two types of intervals such that $C(T)$ increases on type 1 intervals and decreases on type 2 intervals. The candidates for T^* comprise of the minimum T on each type 1 interval that satisfies $R_1(T)\geq R_1$. Then, T^* is the one among all the candidates that leads to the lowest cost.

When the reliability constraint $R_2(\Delta T \mid T)$ is used, we can split the time axis $[0,\infty)$ into four types of intervals such that both $R_2(\Delta T \mid T)$ and $C(T)$ increase on type 1 intervals, both $R_2(\Delta T \mid T)$ and $C(T)$ decrease on type 2 intervals, $R_2(\Delta T \mid T)$ increases while $C(T)$ decreases on type 3 intervals, and $R_2(\Delta T \mid T)$ decreases while $C(T)$ increases on type 4 intervals. The candidates for T^* comprise of the minimum T in each type 1 interval that satisfies $R_2(\Delta T \mid T)\geq R_2$, the maximum T in each type 2 interval that satisfies $R_2(\Delta T \mid T)\geq R_2$, the end points of type 3 intervals which satisfy $R_2(\Delta T \mid T)\geq R_2$, and the initial points of type 4 intervals which satisfy $R_2(\Delta T \mid T)\geq R_2$. The optimal release time T^* is the one corresponding to the lowest cost.

5.4. Numerical examples

For illustration, we consider the paired FDP&FCP model with constant debugging lag that fits the dataset 1 in Section 4. The model parameters are a=199.27, b=0.00717, δ=24.78 and p=0.382. In addition, we assume c_1=\$300, c_2=\$2000, c_3=\$10, ΔT=12, R_1=0.95 and R_2=0.95. In the following, we present the optimal release time that minimizes the cost with specific reliability constraints.

1) Considering cost criterion and reliability target R_1.

From (28), the testing cost under our parameter settings is:

$$C(T) = 372500 - 1700 m_r(T) + 10T. \tag{33}$$

On the other hand, the correction process model with given parameters is:

$$m_r(T) = \begin{cases} 0, T < 24.78 \\ 76.12 - 90.92\mathrm{e}^{-0.00717T}, 24.78 \leq T < 49.56 \\ 199.27 - (204.18 + 1.26T)\mathrm{e}^{-0.00717T}, T \geq 49.56 \end{cases}.$$

By substituting $m_r(T)$ into (33), it can be derived that $C(T)$ increases on [0,24.78], decreases on (24.78,1030.45) and increases on [1030.45,∞). As can be verified, $R_1(0)$<0.95, R_1(1030.45)>0.95. According to the analysis in the preceding section, the optimal release time is $T_1^* = 1030.45$. Correspondingly, the optimal software testing costis $C\left(T_1^*\right) = 45624.87$.

2) Considering cost criterion and reliability target R_2.

When $R_2(\Delta T \mid T)$ is used as the reliability constraint, we can derive the following detection rate according to the specified model parameters:

$$\lambda_d(T) = \begin{cases} 0.5458\mathrm{e}^{-0.00717T}, T < 24.78 \\ (0.3584 + 0.0076T)\mathrm{e}^{-0.00717T}, T \geq 24.78 \end{cases}.$$

It can be verified that $\lambda_d(T)$ decreases on [0,24.78), increases on [24.78,92.07), and decreases on [92.07,∞). Accordingly, $R_2(\Delta T \mid T)$ increases on [0,24.78), decreases on [24.78,92.07), and increases on [92.07,∞). Referring to the analysis in Section 5.3, the axis $[0,\infty)$ can be divided into a type 1 interval [0,24.78), a type 2 interval [24.78,92.07), a type 3 interval [92.07,1030.45) and a type 1 interval [1030.45,$+\infty$). Because $R_2(\Delta T \mid T)$<0.95 for all $T\in$[0,92.07), there is no permissible T in this interval. Therefore, the candidates for the optimal T^* are the right endpoint 1030.45 of the type 3 interval and the minimum T in [1030.45,$+\infty$) that satisfies $R_2(\Delta T \mid T)\geq$0.95. Because $R_2(\Delta T \mid 1030.45)$=0.93<0.95, the optimal T_2^* is found as $\arg\min_T \{T \geq 1030.45, R(\Delta T|T) \geq 0.95\} = 1056.81$. The optimal software testing cost is $C\left(T_2^*\right) = 45645.38$, which is slightly larger than that in the last case. An illustration of the optimal release policies under two scenarios is given in Fig. 4.

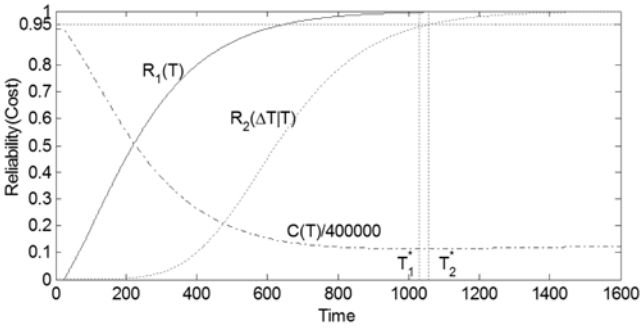

Fig. 4. Variation of normalized total cost function and software reliability functions with testing time

6. Conclusion

In this paper, we proposed a framework for the software reliability growth modeling. The software testing process was considered as a paired fault detection and correction process, and the faults during the testing were classified into leading and dependent faults according to their detectability. The leading faults can be detected and corrected directly, whereas the dependent faults can only be detected until the corresponding leading faults are corrected. For both types of faults, the FCP was modeled as a delayed FDP. In addition, the FDP of dependent faults depended on the FCP of leading faults. Special paired FDP&FCP models were derived under the proposed framework with different assumptions on the debugging lag. The application to two real software testing datasets revealed the effectiveness and the superiority of the proposed models over existing ones. Under this framework, the optimal software release policy was investigated considering cost and reliability requirements.

As a direction for future studies, the proposed modeling framework can be extended to incorporate other information or adapt to other testing environments. For instance, Bayesian technique can be used to incorporate prior information and update model parameters when more information is available. In addition, the imperfect fault correction or the fault introduction phenomenon can be incorporated, as it is common for debuggers to make mistakes with fault correction.

Acknowledgements

The research is supported by the NSFC under grant number 71671016 and 71231001 and 71420107023,and by the Fundamental Research Funds for the Central Universities of China FRF-BR-15-001B.

References

1. Boland P J and Chuív N N. Optimal times for software release when repair is imperfect. Statistics & Probability Letters 2007; 77(12): 1176-1184, https://doi.org/10.1016/j.spl.2007.03.004.
2. Chang Y-C and Liu C-T. A generalized JM model with applications to imperfect debugging in software reliability. Applied Mathematical Modelling 2009; 33(9): 3578-3588, https://doi.org/10.1016/j.apm.2008.11.018.
3. Febrero F, Calero C, and Ángeles Moraga M. Software reliability modeling based on ISO/IEC SQuaRE. Information and Software Technology 2016; 70: 18-29, https://doi.org/10.1016/j.infsof.2015.09.006.
4. Goel A L and Okumoto K. Time-dependent error-detection rate model for software reliability and other performance measures. IEEE Transactions on Reliability 1979; 28(3): 206-211, https://doi.org/10.1109/TR.1979.5220566.
5. Gokhale S S, Lyu M R, and Trivedi K S. Analysis of software fault removal policies using a non-homogeneous continuous time Markov chain. Software Quality Journal 2004;12(3): 211-230, https://doi.org/10.1023/B:SQJO.0000034709.63615.8b.
6. Goseva-Popstojanova K and Trivedi K S. Failure correlation in software reliability models. IEEE Transactions on Reliability 2000; 49(1): 37-48, https://doi.org/10.1109/24.855535.
7. Gutjahr W J. A reliability model for nonhomogeneous redundant software versions with correlated failures. Computer Systems Science and Engineering 2001;16(6): 361-370.
8. Hu Q, Xie M, Ng S H, and Levitin G. Robust recurrent neural network modeling for software fault detection and correction prediction. Reliability Engineering & System Safety 2007;92(3): 332-340. https://doi.org/10.1016/j.ress.2006.04.007
9. Huang C-Y and Huang W-C, Software reliability analysis and measurement using finite and infinite server queueing models. IEEE Transactions on Reliability 2008; 57(1): 192-203, https://doi.org/10.1109/TR.2007.909777.
10. Huang C-Y, Kuo S-Y, and Lyu M R. An assessment of testing-effort dependent software reliability growth models. IEEE Transactions on Reliability 2007; 56(2): 198-211, https://doi.org/10.1109/TR.2007.895301.
11. Huang C-Y and Lin C-T. Software reliability analysis by considering fault dependency and debugging time lag. IEEE Transactions on Reliability 2006;55(3): 436-450. https://doi.org/10.1109/TR.2006.879607
12. Huang C-Y and Lyu M R, Optimal release time for software systems considering cost, testing-effort, and test efficiency. IEEE Transactions on Reliability 2005; 54(4): 583-591, https://doi.org/10.1109/TR.2005.859230.
13. Inoue S and Yamada S. Generalized discrete software reliability modeling with effect of program size. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 2007; 37(2): 170-179, https://doi.org/10.1109/TSMCA.2006.889475.
14. Jain M and Priya K. Software reliability issues under operational and testing constraints. Asia-Pacific Journal of Operational Research 2005; 22(01): 33-49, https://doi.org/10.1142/S021759590500042X.
15. Jha P, Gupta D, Yang B, and Kapur P. Optimal testing resource allocation during module testing considering cost, testing effort and reliability. Computers & Industrial Engineering 2009; 57(3): 1122-1130, https://doi.org/10.1016/j.cie.2009.05.001.
16. Jia L, Yang B, Guo S, and Park D H. Software reliability modeling considering fault correction process. IEICE Transactions on Information and Systems 2010; 93(1): 185-188, https://doi.org/10.1587/transinf.E93.D.185.
17. Kapur P, Goswami D, Bardhan A, and Singh O. Flexible software reliability growth model with testing effort dependent learning process. Applied Mathematical Modelling 2008; 32(7): 1298-1307, https://doi.org/10.1016/j.apm.2007.04.002.
18. Kapur P and Younes S. Software reliability growth model with error dependency. Microelectronics Reliability 1995; 35(2): 273-278, https://doi.org/10.1016/0026-2714(94)00054-R.
19. Kim H S, Park D H, and Yamada S. Bayesian optimal release time based on inflection S-shaped software reliability growth model. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 2009; 92(6): 1485-1493, https://doi.org/10.1587/transfun.E92.A.1485.
20. Lee C H, Kim Y T, and Park D H. S-shaped software reliability growth models derived from stochastic differential equations. IIE transactions 2004; 36(12): 1193-1199, https://doi.org/10.1080/07408170490507792.
21. Li X, Xie M, and Ng S H. Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points. Applied Mathematical Modelling 2010; 34(11): 3560-3570, https://doi.org/10.1016/j.apm.2010.03.006.
22. Lin C-T and Huang C-Y., Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models. Journal of Systems and Software 2008; 81(6): 1025-1038, https://doi.org/10.1016/j.jss.2007.10.002.
23. Lin C-T and Huang C-Y. Staffing level and cost analyses for software debugging activities through rate-based simulation approaches. IEEE Transactions on Reliability 2009; 58(4): 711-724, https://doi.org/10.1109/TR.2009.2019669.
24. Lin C-T and Li Y-F. Rate-based queueing simulation model of open source software debugging activities. IEEE Transactions on Software Engineering 2014; 40(11): 1075-1099, https://doi.org/10.1109/TSE.2014.2354032.
25. Lo J-H and Huang C-Y. An integration of fault detection and correction processes in software reliability analysis. Journal of Systems and Software 2006; 79(9): 1312-1323, https://doi.org/10.1016/j.jss.2005.12.006.
26. Lyu M R. Handbook of Software Reliability Engineering. New York: McGraw-Hill, Inc., 1996.
27. Musa J D, Iannino A, and Okumono K. Software Reliability, Measurement, Prediction and Application. New York: McGraw-Hill, Inc, 1987.
28. Okamura H and Dohi T. Unification of software reliability models using Markovian arrival processes, in Proceedings of 17th Pacific Rim International Symposium on Dependable Computing (PRDC) 2011: 20-27, https://doi.org/10.1109/prdc.2011.12.

29. Okamura H, Dohi T, and Osaki S. Software reliability growth models with normal failure time distributions. Reliability Engineering & System Safety 2013; 116: 135-141, https://doi.org/10.1016/j.ress.2012.02.002.
30. Peng R, Li Y-F, Zhang J-G, and Li X. A risk-reduction approach for optimal software release time determination with the delay incurred cost. International Journal of Systems Science 2015; 46(9): 1628-1637, https://doi.org/10.1080/00207721.2013.827261.
31. Peng R, Li Y-F, Zhang W, and Hu Q. Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction. Reliability Engineering & System Safety 2014;126: 37-43, https://doi.org/10.1016/j.ress.2014.01.004.
32. Pietrantuono R, Russo S, and Trivedi K S. Software reliability and testing time allocation: An architecture-based approach. IEEE Transactions on Software Engineering 2010; 36(3): 323-337, https://doi.org/10.1109/TSE.2010.6.
33. Rana R, Staron M, Berger C, Hansson J, Nilsson M, Törner F, et al., Selecting software reliability growth models and improving their predictive accuracy using historical projects data. Journal of Systems and Software 2014; 98: 59-78, https://doi.org/10.1016/j.jss.2014.08.033.
34. Schneidewind N F. Analysis of error processes in computer software, in Proceedings of 1975 International Conference on Reliable Software 1975: 337-346, https://doi.org/10.1145/800027.808456.
35. Shatnawi O. Discrere time modelling in software reliability engineering-A unified approach. Computer Systems Science and Engineering 2009;24(6): 391.
36. Shatnawi O. Measuring commercial software operational reliability: an interdisciplinary modelling approach. Eksploatacja i Niezawodnosc - Maintenance and Reliability 2014;16(4): 585-594.
37. Shibata K, Rinsaka K, Dohi T, and Okamura H. Quantifying software maintainability based on a fault-detection/correction model, in Proceedings of 13th Pacific Rim International Symposium on Dependable Computing (PRDC2007) 2007: 35-42, https://doi.org/10.1109/PRDC.2007.46.
38. Shinohara Y, Nishio Y, Dohi T, and Osaki S. An optimal software release problem under cost rate criterion: artificial neural network approach. Journal of Quality in Maintenance Engineering 1998; 4(4): 236-247, https://doi.org/10.1108/13552519810233967.
39. Tamura Y and Yamada S. A flexible stochastic differential equation model in distributed development environment. European Journal of Operational Research 2006; 168(1): 143-152, https://doi.org/10.1016/j.ejor.2004.04.034.
40. Ullah N, Morisio M, and Vetro A. A comparative analysis of software reliability growth models using defects data of closed and open source software, in Proceedings of 35th Annual IEEE Software Engineering Workshop 2012: 187-192, https://doi.org/10.1109/sew.2012.26.
41. Wang L, Hu Q, and Liu J. Software reliability growth modeling and analysis with dual fault detection and correction processes. IIE Transactions 2016; 48(4): 359-370, https://doi.org/10.1080/0740817X.2015.1096432.
42. Wu Y, Hu Q, Xie M, and Ng S H. Modeling and analysis of software fault detection and correction process by considering time dependency. IEEE Transactions on Reliability 2007; 56(4): 629-642, https://doi.org/10.1109/TR.2007.909760.
43. Xie M. Software reliability modelling. Singapore: World Scientific, 1991, https://doi.org/10.1142/1390.
44. Xie M, Hu Q, Wu Y, and Ng S H. A study of the modeling and analysis of software fault-detection and fault-correction processes. Quality and Reliability Engineering International 2007; 23(4): 459-470, https://doi.org/10.1002/qre.827.
45. Yamada S, Ohba M, and Osaki S. S-shaped software reliability growth models and their applications. IEEE Transactions on Reliability 1984; 33(4): 289-292, https://doi.org/10.1109/TR.1984.5221826.
46. Yang B, Guo S, Ning N, and Huang H-Z. Parameter estimation for software reliability models considering failure correlation, in Proceedings of Annual Reliability and Maintainability Symposium (RAMS 2008) 2008: 405-410, https://doi.org/10.1109/rams.2008.4925830.
47. Zhang X and Pham H. Comparisons of nonhomogeneous Poisson process software reliability models and its applications. International Journal of Systems Science 2000; 31(9): 1115-1123, https://doi.org/10.1080/002077200418397.
48. Zhang X and Pham H. Predicting operational software availability and its applications to telecommunication systems. International Journal of Systems Science 2002; 33(11): 923-930, https://doi.org/10.1080/0020772021000023022.
49. Zhang X, Teng X, and Pham H. Considering fault removal efficiency in software reliability assessment. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 2003; 33(1): 114-120, https://doi.org/10.1109/TSMCA.2003.812597.
50. Zhao J, Liu H-W, Cui G, and Yang X-Z. Software reliability growth model with change-point and environmental function. Journal of Systems and Software 2006; 79(11): 1578-1587, https://doi.org/10.1016/j.jss.2006.02.030.

Rui PENG
Donlinks School of Economics & Management
University of Science & Technology Beijing, China

Qingqing ZHAI
Department of Industrial and Systems Engineering
National University of Singapore, Singapore

E-mails: pengrui1988@ustb.edu.cn, isezq@nus.edu.sg