Reliability Engineering and System Safety 126 (2014) 37–43

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction ☆

R. Peng [a,*], Y.F. Li [b], W.J. Zhang [c], Q.P. Hu [d]

[a] Dongling School of Economics & Management, University of Science & Technology Beijing, China
[b] Ecole Centrale Paris-SUPELEC, Paris, France
[c] Warwick Business School, The University of Warwick, Coventry CV4 7AL, UK
[d] Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China

ARTICLE INFO

Article history:
Received 2 May 2012
Received in revised form
3 January 2014
Accepted 5 January 2014
Available online 11 January 2014

Keywords:
Software reliability
Fault detection process
Fault correction process
Testing effort function
Imperfect debugging

ABSTRACT

This paper studies the fault detection process (FDP) and fault correction process (FCP) with the incorporation of testing effort function and imperfect debugging. In order to ensure high reliability, it is essential for software to undergo a testing phase, during which faults can be detected and corrected by debuggers. The testing resource allocation during this phase, which is usually depicted by the testing effort function, considerably influences not only the fault detection rate but also the time to correct a detected fault. In addition, testing is usually far from perfect such that new faults may be introduced. In this paper, we first show how to incorporate testing effort function and fault introduction into FDP and then develop FCP as delayed FDP with a correction effort. Various specific paired FDP and FCP models are obtained based on different assumptions of fault introduction and correction effort. An illustrative example is presented. The optimal release policy under different criteria is also discussed.

1. Introduction

As software is becoming more and more widely used, both the functionality and the correctness of software are of great concern. In order to ensure high reliability, testing is usually conducted, during which faults in software manifest by causing failures and can be detected and removed by debuggers [9,4,15]. On the other hand, it is almost impossible to make bug-free software even though scientific and disciplined development practices are followed. During the last 30 years, many software reliability growth models (SRGMs) have been proposed as a tool to track the reliability growth trend of the software testing process [16,3,35,23]. SRGMs are very useful in the sense that they can help management making critical decisions, such as testing resource allocation and the determination of software release time [19,13,12].

Testing consumes a large amount of resources, such as manpower and CPU hours, which are usually not constantly allocated during testing phase. The function that describes how testing resources are distributed is usually referred to as testing effort function (TEF) and it has been incorporated into software reliability studies by some researchers [30,16,17]. Yamada et al. [34] pointed out that the TEF could be described by a Weibull-type distribution, which actually includes Exponential curve, Rayleigh curve and Weibull curve. Weibull-type curve can well fit most data and is often used in the field of software reliability modeling [7]. Logistic TEF is used instead of Weibull-type TEF by some researchers and appeared to be fairly accurate in describing the consumption of testing effort [24,5,8].

Generally a detected fault cannot be corrected immediately and the time required to correct a detected fault is usually called debugging lag/delay. The idea of modeling the fault correction process (FCP) was first proposed in Schneidewind [28], in which it was modeled as a separate process following the fault detection process (FDP) with a constant time lag. Based on this framework, Xie et al. [33] and Wu et al. [31] proposed several paired FDP and FCP models through incorporating other variants of debugging delay. Later, Hwang and Pham [10] developed a generalized NHPP model considering quasi-renewal time-delay fault removal. Jia et al. [14] proposed a Markovian software reliability model considering the fault correction process. However, the influence of testing effort on debugging lag is not considered in these papers. Intuitively, the time needed to correct a detected fault, or the debugging lag, tends to be shorter if more testing effort is allocated during the period between detection and correction of the fault. Thus it is more reasonable to incorporate testing effort function into the modeling framework on both FDP and FCP.

* Corresponding author. Tel.: +86 13051540519.
E-mail address: pengrui1988@gmail.com (R. Peng).

0951-8320/$ - see front matter © 2014 The Authors. Published by Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.ress.2014.01.004

Moreover, the debugging process is usually far from perfect and actually many faults encountered by customers are those introduced during debugging [36,29,6,26]. It is essential to incorporate imperfect debugging into FDP and FCP models [32,2,18].

In this paper, a framework is proposed to develop testing effort dependent FDP and FCP models with the consideration of imperfect debugging. The rest of this paper is organized as follows. In Section 2, a framework is proposed to obtain testing effort dependent paired FDP and FCP models with the consideration of fault introduction. In Section 3, several specific models are derived based on different assumptions of fault introduction and the correction effort. In Section 4, several commonly used testing effort functions are reviewed. In Section 5, an illustrative example is presented. The optimal release policy under different criteria is studied in Section 6. Conclusions and discussions are presented in Section 7.

2. Testing effort dependent FDP and FCP models with fault introduction

The expected total number of faults at time t is denoted by the fault content rate function $a(t)$, which is the sum of the number of initial faults in the software a $(=a(0))$ and the number of faults introduced during time interval $[0, t)$. We use $w(t)$ to denote the time dependent testing effort rate and $W(t)$ to denote the cumulative testing effort consumed till time t.

2.1. FDP model

Mean value function $m_d(t)$ is used to depict the expected number of faults detected till time t and $\lambda_d(t)=dm_d(t)/dt$ is used to denote the fault intensity function. The number of faults detected during time interval $[t, t+\Delta t)$ by current testing effort expenditure is usually assumed to be proportional to the number of remaining faults at time t [21]. Hence we have

$$\lambda_d(t)=\frac{dm_d(t)}{dt}=b(t)w(t)(a(t)-m_d(t)) \tag{1}$$

where $b(t)$ is the current fault detection rate per unit of testing effort at time t, and $w(t)$ is the current testing effort expenditure at time t. Substituting the marginal condition $m_d(0)=0$ into (1) gives

$$m_d(t)=a(t)-a\exp\left\{-\int_0^t b(x)w(x)dx\right\} -\exp\left\{-\int_0^t b(x)w(x)dx\right\}\int_0^t a'(x)\exp\left\{\int_0^x b(y)w(y)dy\right\}dx \tag{2}$$

where $a'(x)=da(x)/dx$. Various $m_d(t)$ can be derived based on different assumptions of $a(t)$, $b(t)$ and $w(t)$. $\lambda_d(t)$ can be obtained by substituting (2) into the right hand side of (1) as

$$\lambda_d(t)=\frac{dm_d(t)}{dt}=ab(t)w(t)\exp\left\{-\int_0^t b(x)w(x)dx\right\}\left(1+\int_0^t\frac{a'(x)}{a}\exp\left\{\int_0^x b(y)w(y)dy\right\}dx\right) \tag{3}$$

2.2. FCP model

Mean value function $m_r(t)$ is used to denote the expected number of faults removed till time t and $\lambda_r(t)=dm_r(t)/dt$ is used to denote the fault removal intensity function. Since a removed fault must first be detected, FCP can be modeled as a separate process following FDP with a debugging delay. For convenience of discussion, the testing effort consumed during the period from detection of a fault to the final removal of the fault is termed as correction effort of the fault. Generally correcting different faults requires different amounts of testing resources, hence correction effort can be modeled as a random variable with probability density function (pdf) and the cumulative distribution function (cdf) denoted as $f(x)$ and $F(x)$.

Thus it can be obtained that

$$m_r(t)=\int_0^t\lambda_d(y)F(W(t)-W(y))dy \tag{4}$$

where $F(W(t)-W(y))$ is the probability that the fault detected at y is corrected before t.

Different $m_r(t)$ can be derived based on $m_d(t)$ and different $f(x)$. Furthermore, we have

$$\begin{aligned}\lambda_r(t)&=\int_0^t\lambda_d(y)f(W(t)-W(y))w(t)dy\\&=\int_0^{W*(t)}\lambda_d(W^{-1}(W(t)-x))f(x)w(t)d(W^{-1}(W(t)-x))\\&=\int_0^{W*(t)}\lambda_d(W^{-1}(W(t)-x))f(x)w(t)\frac{dx}{w(W^{-1}(W(t)-x))}\end{aligned} \tag{5}$$

Different $m_r(t)$ can be derived based on $m_d(t)$ and different $f(x)$.

3. Some specific models

Fault detection rate function $b(t)$ is usually assumed to be constant and it is denoted as b here [21]. From (2) we have

$$m_d(t)=a(t)-a\exp\{-bW^*(t)\}-\exp\{-bW^*(t)\}\int_0^t a'(x)\exp\{bW^*(x)\}dx \tag{6}$$

The total number of faults $a(t)$ was usually assumed to be an exponential or linear function of time in the literature. Yamada et al. [34] proposed two FDP models with consideration of imperfect debugging, by assuming that the expected total number of faults increases exponentially and linearly with the testing time, respectively. An S-shaped concave FDP model was proposed in Pham et al. [25] assuming that the total number of faults is a linear function of the testing time. In the following subsections various TEF dependent FDP and FCP models are derived based on different assumptions on $a(t)$ and $f(x)$.

3.1. Paired model 1

We assume that the total number of faults increases exponentially with the total testing effort consumed and the correction effort required is an exponential variable as

$$a(t)=a\exp\{\alpha W^*(t)\},\quad \alpha\ge 0 \tag{7}$$

$$f(x)=c\exp\{-cx\} \tag{8}$$

In this case we have

$$m_d(t)=\frac{ab}{b+\alpha}(\exp\{\alpha W^*(t)\}-\exp\{-bW^*(t)\}) \tag{9}$$

$$m_r(t)=\begin{cases}\frac{ab}{(b+\alpha)^2}(b\exp\{\alpha W^*(t)\}+\alpha\exp\{-bW^*(t)\})-\frac{ab}{(b+\alpha)}(1+bW^*(t))\exp\{-bW^*(t)\}, & c=b\\ \frac{a}{(1+\alpha/b)}\left(\frac{c\exp\{\alpha W^*(t)\}+\alpha\exp\{-cW^*(t)\}}{c+\alpha}+\frac{c\exp\{-bW^*(t)\}-b\exp\{-cW^*(t)\}}{b-c}\right), & c\ne b\end{cases} \tag{10}$$

Actually (9) can be obtained by combining (6) and (7). (10) can be obtained by substituting (9) into (3), and (4). When $W^*(t)=t$, (9) is the same as the FDP model obtained in Yamada et al. [34] for the case when the total number of faults is an exponential function of testing time. When $\alpha=0$ and $W^*(t)=t$, (9) and (10) are the same as the paired model obtained in Wu et al. [31] for the case of exponential debugging delay.

3.2. Paired model 2

We assume that the total number of faults increases exponentially with the total testing effort consumed as given in (7) and the correction effort required is a gamma variable as

$$f(x)=\frac{\mu \exp\{-\mu x\}(\mu x)^{c-1}}{\Gamma(c)}, \quad c,\ \mu>0 \tag{11}$$

where $\Gamma(c)=\int_0^\infty \exp\{-y\}y^{c-1}\,dy$ is the Euler gamma function.

Similarly we have

$$m_d(t)=\frac{ab}{b+\alpha}(\exp\{\alpha W^*(t)\}-\exp\{-bW^*(t)\}) \tag{12}$$

$$m_r(t)=\begin{cases} \frac{a\ \exp\{\alpha W^*(t)\}\Gamma(c,0,(b+\alpha)W^*(t))}{(1+\frac{\alpha}{b})^{c+1}\Gamma(c)}-\frac{a\Gamma(c,0,bW^*(t))}{(1+\frac{\alpha}{b})\Gamma(c)}+\frac{a\Gamma(c+1,0,bW^*(t))}{(1+\frac{\alpha}{b})c\Gamma(c)}), & \mu=b \\ \frac{a\ \exp\{\alpha W^*(t)\}\Gamma(c,0,(\mu+\alpha)W^*(t))}{(1+\frac{\alpha}{b})\left(1+\frac{\alpha}{\mu}\right)^c\Gamma(c)}-\frac{a\ \exp\{-bW^*(t)\}\Gamma(c,0,(\mu-b)W^*(t))}{(1+\frac{\alpha}{b})\left(1-\frac{b}{\mu}\right)^c\Gamma(c)}, & \mu\neq b \end{cases} \tag{13}$$

where $\Gamma(\varepsilon_1,\varepsilon_2,\varepsilon_3)=\int_{\varepsilon_2}^{\varepsilon_3} e^{-y}y^{\varepsilon_1-1}\,dy$ is a generalized incomplete gamma function. When $\alpha=0$ and $W^*(t)=t$, (12) and (13) are the same as the paired model obtained in Wu et al. [31] for the case of gamma debugging delay.

3.3. Paired model 3

We assume that the total number of faults increases linearly with the total testing effort consumed and the correction effort required is an exponential variable as

$$a(t)=a+sW^*(t), \quad s\geq 0 \tag{14}$$

$$f(x)=c\exp\{-cx\} \tag{15}$$

In this case we have

$$m_d(t)=\left(a-\frac{s}{b}\right)(1-\exp\{-bW^*(t)\})+sW^*(t) \tag{16}$$

$$m_r(t)=\begin{cases} (a-\frac{2s}{b})(1-(1+bW^*(t))\exp\{-bW^*(t)\}+sW^*(t)(1-\exp\{-bW^*(t)\}), & c=b \\ (a-\frac{s}{b})\left(1+\frac{b\ \exp\{-cW^*(t)\}-c\ \exp\{-bW^*(t)\}}{c-b}\right)+sW^*(t)-\frac{s}{c}(1-\exp\left\{-cW^*(t)\right.\}), & c\neq b \end{cases} \tag{17}$$

Actually (16) can be obtained by combining (6) and (14). (17) can be obtained by substituting (16) into (3) and (4). When $W^*(t)=t$, (16) is the same as the FDP model obtained in Yamada et al. [34] for the case when the total number of faults is a linear function of testing time. When $s=0$ and $W^*(t)=t$, (16) and (17) are the same as the paired model obtained in Wu et al. [31] for the case of exponential debugging delay.

3.4. Paired model 4

We assume that the total number of faults increases linearly with the total testing effort consumed as given in (14) and the correction effort required is a gamma variable as given in (11).

Similarly we have

$$m_d(t)=\left(a-\frac{s}{b}\right)(1-\exp\{-bW^*(t)\})+sW^*(t) \tag{18}$$

$$m_r(t)=\begin{cases} \frac{sw(t)}{b\Gamma(c)}(bW^*(t)\Gamma(c,0,bW^*(t)-\Gamma(c+1,0,bW^*(t))+\frac{(a-s/b)}{c\Gamma(c)}\Gamma(c+1,0,bW^*(t)), & \mu=b \\ \frac{s}{\mu\Gamma(c)}(\mu W^*(t)\Gamma(c,0,\mu W^*(t)-\Gamma(c+1,0,\mu W^*(t)) \\ +\frac{(a-s/b)}{\Gamma(c)}\Gamma(c,0,\mu W^*(t))-\frac{(a-s/b)\exp\{-bW^*(t)\}}{\Gamma(c)(1-b/\mu)^c}\Gamma(c,0,(\mu-b)W^*(t)), & c\neq b \end{cases} \tag{19}$$

when $s=0$ and $W^*(t)=t$, (18) and (19) are the same as the paired model obtained in Wu et al. [31] for the case of gamma debugging delay.

4. A summary of various testing effort functions

Testing effort functions that have been commonly used include Constant, Exponential, Rayleigh, Weibull and Logistic curves. Exponential curve and Rayleigh curve can be regarded as special cases of Weibull curve. The details are shown below.

4.1. Constant TEF

We assume that $w(t)$ is a constant. It can be expressed as

$$w(t)=w \tag{20}$$

Thus the cumulative testing effort $W(t)$ can be obtained as

$$W(t)=wt \tag{21}$$

It can be seen that the total testing effort consumed tends to positive infinity, when t approaches positive infinity. In the case that TEF is not considered, it can be regarded as considering $w(t)=1$.

4.2. Weibull TEF

Weibull TEF is very flexible and it can well fit most data that are often used in the study of SRGM. The cumulative TEF $W(t)$ is given by

$$W(t)=N(1-\exp\{-\beta t^m\}) \tag{22}$$

where N is the expected total amount of testing effort that is required by software testing. β and m are the scale parameter and shape parameter, respectively. It should also be noted that the cumulative testing effort consumed is finite and tends to N when t approaches positive infinity.

Differentiating (22) gives

$$w(t)=N\beta m t^{m-1}\exp\{-\beta t^m\} \tag{23}$$

The exponential TEF is a special case of Weibull TEF when $m=1$. Exponential curve is suitable to describe the testing environment which has a monotonically declining testing effort rate.

The Rayleigh TEF is a special case of Weibull TEF when $m=2$. Rayleigh testing effort rate first increases to its peak, then decreases with a decelerating speed to zero asymptotically without reaching zero.

4.3. Logistic TEF

Logistic curve was first proposed in Parr [24] as an alternative of Rayleigh curve. It exhibits similar behavior as Rayleigh curve, except during the initial stage of the project. The logistic cumulative TEF $W(t)$ is given by

$$W(t)=\frac{N}{1+A\exp\{-\eta t\}} \tag{24}$$

where A is a constant parameter and η is the consumption rate of testing effort expenditure. Similar to the Weibull case, the cumulative testing effort consumed is finite and tends to N when t approaches positive infinity.

Taking derivatives on both sides of (24) gives

$$w(t)=\frac{NA\eta}{\left(\exp\{\frac{\eta t}{2}\}+A\exp\{-\frac{\eta t}{2}\}\right)^2} \tag{25}$$

$w(t)$ reaches its maximum value when $t=\ln A/\eta$.

5. Illustrative example

5.1. Dataset description

The dataset we use is from the System T1 data of the Rome Air Development Center (RADC) [22]. Although this is quite an old dataset, it is widely used and it contains both fault detection data and fault correction data. Additionally, it contains information of testing effort, which is characterized by computer time (CPU hours) consumed in each week. Hence this familiar dataset is used for illustration.

The cumulative numbers of detected faults and corrected faults during the first 21 weeks are shown in Table 1. During the time span, totally 300.1 CPU hours were consumed. Till the end of the testing, 136 faults were detected and all of them were corrected.

5.2. Select the most suitable TEF for this dataset

Parameters in the different types of TEF are estimated by Least Square Error (LSE). In order to select a TEF that best fits this dataset, some criteria are used to compare the performances of different TEFs.

Table 1
The dataset – System T1.

Weeks	Computer time (CPU hours)	Cumulative number of detected faults	Cumulative number of corrected faults
1	4	2	1
2	4.3	2	2
3	2	2	2
4	0.6	3	3
5	2.3	4	4
6	1.6	6	4
7	1.8	7	5
8	14.7	16	7
9	25.1	29	13
10	4.5	31	17
11	9.5	42	18
12	8.5	44	32
13	29.5	55	37
14	22	69	56
15	39.5	87	75
16	26	99	85
17	25	111	97
18	31.4	126	117
19	30	132	129
20	12.8	135	131
21	5	136	136

Table 2
Estimated parameters and comparison results for different TEF.

TEF	Estimated parameters	RMSE	Bias	Variance	RMSPE
Constant	$w=14.29$	12.11	−0.00048	12.11	12.11
Weibull	$N=407.0830$ $\beta=2.064\text{E}-4$ $m=2.923$	7.86	1.2615	7.76	7.86
Logistic	$N=321.482$ $\eta=0.3826$ $A=423.788$	6.7828	−0.5778	6.7570	6.7818

(1) RMSE
The Root Mean Square Error (RMSE) is defined as

$$RMSE=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(w(t_i)-w_i)^2} \tag{26}$$

A smaller RMSE indicates a smaller fitting error and better performance.

(2) Bias
The bias is defined as the sum of the deviation of the estimated testing curve from the actual data, as shown below:

$$Bias=\frac{1}{n}\sum_{i=1}^{n}(w(t_i)-w_i) \tag{27}$$

(3) Variance
The variance is defined as [8]

$$Variance=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(w(t_i)-w_i-Bias)^2} \tag{28}$$

(4) RMSPE
The Root Mean Square Prediction Error (RMSPE) is defined as [8]

$$RMSPE=\sqrt{Variance+Bias^2} \tag{29}$$

RMSPE is also a measure to depict how close the model predicts the observation.

Estimated parameters and comparison results for different TEFs are shown in Table 2. Fig. 1 is plotted for graphical illustration.

It can be seen that logistic TEF has the smallest RMSE, Variance, and RMSPE and also has a smaller Bias than Weibull TEF. Fig. 1 also shows that logistic TEF fits best. Thus logistic TEF is adopted for further analysis.

5.3. Performance analysis

The paired model (9) and (10) is used for illustration. After substituting the cumulative logistic testing effort function (24) with the estimated parameters $N=321.482$, $\eta=0.3826$, and $A=423.788$ into (9) and (10), the paired model is applied to fit against the real dataset. The LSE estimation of the parameters are obtained as $a=100.97$, $b=0.0094$, $\alpha=0.0021$ and $c=0.0418$. According to the estimated parameters, there are about 100.97 faults at the beginning of testing. The total number of faults when t approaches infinity is expected to be $\lim_{t\to\infty}a(t)=198.01$. Fig. 2 is plotted for graphical illustration.

6. Software release policies

Determination of the optimal release time is a critical decision for software projects and has been studied in many papers [1,11,20]. As cost and reliability requirements are of great concern, they are often used to determine the time to stop the testing and release the software [21,27].

6.1. Software release policy based on reliability criterion

Software reliability 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 software normally does not change in operational phase, the reliability function is

$$R(\Delta T|T) = \exp\{-\lambda_d(T)\Delta T\} \tag{30}$$

If R_1 is the reliability target and T_{LC} is the length of the software life cycle, the time when the reliability of the software reaches R_1 can be obtained as $T_1 = \inf\{\lambda_d(T) \le \ln(1/R_1)/\Delta T : T \in [0, T_{LC}]\}$.

6.2. Software release policy based on cost criterion

Besides the reliability requirement, we can also discuss the optimal release time based on the total cost during the software testing phase and the operational phase. With the incorporation of FCP $m_r(t)$, the cost model can be expressed as

$$C(T) = c_1 m_r(T) + c_2(m_d(T_{LC}) - m_r(T)) + c_3 W^*(T) \tag{31}$$

where c_1 is the cost of fixing a fault during the testing phase, c_2 is the cost of fixing a fault during the operational phase $(c_2 > c_1 > 0)$, c_3 is the unit cost for testing effort consumed during testing. By minimizing the cost model with respect to T, the optimal release time T_c can be obtained.

Differentiating both sides of (31), we have

$$C'(T) = c_3 w(T) - (c_2 - c_1)\lambda_r(T) \tag{32}$$

Furthermore we have $C'(0) = c_3 w(0) > 0$ Let $z_1 \le z_2 \le \ldots \le z_n$ be all the solutions to $\lambda_r(T)/w(T) = c_3/(c_2 - c_1)$ during $(0, T_{LC})$. If $n=2k$ $(k \ge 0)$, T_c can be determined as $T_c = \arg\min_{T=0,z_2,\ldots,z_{2k}} C(T)$. Otherwise $n=2k+1$ and T_c can be determined as $T_c = \arg\min_{T=0,z_2,\ldots,z_{2k},T_{LC}} C(T)$.

6.3. Software release policy based on mixed criterion

When both reliability requirements and the total cost are considered, our goal is to determine the optimal release time T^* which minimizes the total cost without compromising the reliability requirements. Thus the problem can be formulated as

Minimize $C(T) = c_1 m_r(T) + c_2(m_d(T_{LC}) - m_r(T)) + c_3 W^*(T)$

Subject to $R(\Delta T|T) = \exp[-\lambda_d(T)\Delta T] \ge R_1$

The time axis $[T_1, T_{LC})$ can be divided into four types of intervals such that both $R(\Delta T|T)$ and $C(T)$ increase on type 1 intervals, both $R(\Delta T|T)$ and $C(T)$ decrease on type 2 intervals, $R(\Delta T|T)$ increases while $C(T)$ decreases on type 3 intervals, and $R(\Delta T|T)$ 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 $R(\Delta T|T) \ge R_1$, the maximum T in each type 2 interval that satisfies $R(\Delta T|T) \ge R_1$, the end points of type 3 intervals which satisfy $R(\Delta T|T)$

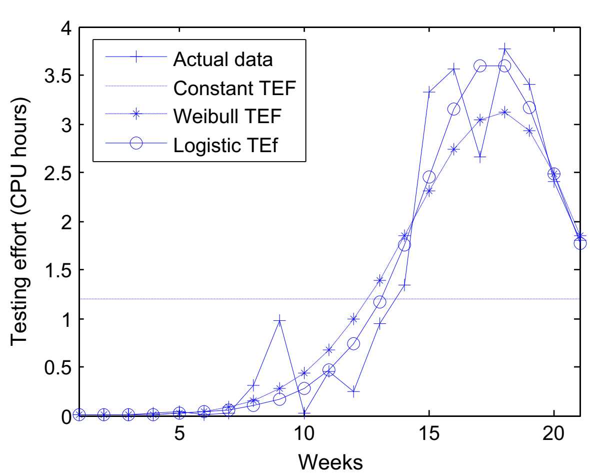

Fig. 1. Observed/estimated TEF for the real dataset.

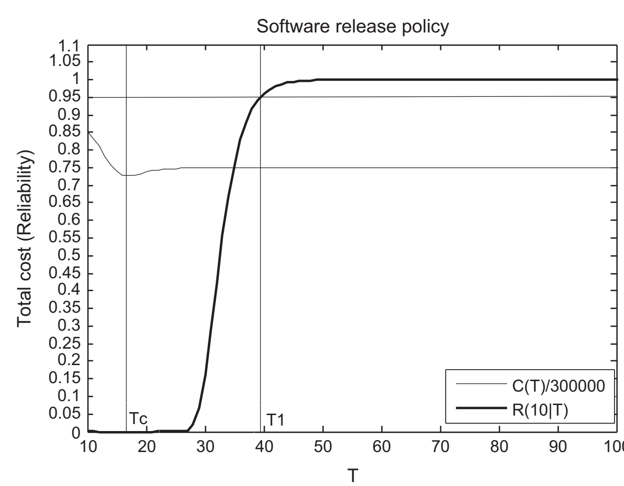

Fig. 3. Plot of total cost function, and software reliability function.

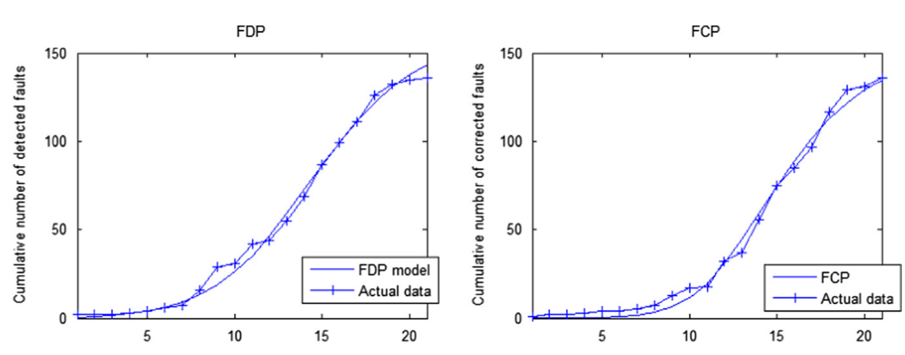

Fig. 2. The paired model fitted against the real dataset.

$\geq R_1$, and the initial points of type 4 intervals which satisfy $R(\Delta T|T) \geq R_1$. T^* equals to the candidate which incurs the lowest cost.

6.4. Numerical examples for software release policy

For illustration, we consider the first paired model (9) and (10) with parameters estimated as $a=100.97$, $b=0.0094$, $\alpha=0.0021$ and $c=0.0418$ and logistic TEF with parameters estimated as $N=321.482$, $\eta=0.3826$, and $A=423.788$. We also assume $T_{LC}=300$, $c_1=\$300$, $c_2=\$2000$, $c_3=\$700$,,$\Delta T=10$, and $R_1=0.95$.

From (9) we have

$$\lambda_d(T)=bw(T)\left(\frac{a\alpha}{b+\alpha}\exp\{\alpha W^*(T)\}+\frac{ab}{b+\alpha}\exp\{-bW^*(T)\}\right)$$
$$=\frac{9033.4\exp\{0.0021W^*(T)\}+40439\exp\{-0.0094W^*(T)\}}{(\exp\{0.1913T\}+423.788\exp\{-0.1913T\})^2} \tag{33}$$

It can be seen that $\lambda_d(T)$ increases from on $[0, 14.112]$ and decreases on $(14.112, 300)$. Solving $\lambda_d(T)=\ln(1/R_1)/\Delta T=0.0051$ gives $T_1=39.626$. The reliability requirement is satisfied if the software is released after 39.626 weeks of testing.

From (46) and (47) we have

$$C(T)=c_1m_r(T)+c_2(m_d(T_{LC})-m_r(T))+c_3W^*(T)$$
$$=253590-1700m_r(T)+700W^*(T) \tag{34}$$

$$C'(T)=700w(T)-1700\lambda_r(T) \tag{34}$$

In addition we have

$$\frac{\lambda_r(T)}{w(T)}=\frac{a\alpha bc}{(b+\alpha)(c+\alpha)}(\exp\{\alpha W^*(T)\}-\exp\{-cW^*(T)\})$$
$$+\frac{ab^2c(\exp\{-cW^*(T)\}-\exp\{-bW^*(T)\})}{(b+\alpha)(b-c)}$$
$$=0.165\exp\{0.0021W^*(T)\}-1.1659\exp\{-0.0418W^*(T)\}$$
$$+1.0009\exp\{-0.0094W^*(T)\} \tag{35}$$

Solving $\lambda_r(T)/w(T)=700/1700$ gives $T=8.013$ and 16.848. Thus $C(T)$ increases on $[0, 8.013]$, decreases on $(8.013, 16.848)$ and increases on $[16.848, 300]$. The optimal release time which minimizes the total cost is $T_c=16.848$. The corresponding total cost is $C(T)=\$217820$.

As both $R(\Delta T|T)$ and $C(T)$ increase on $[T_1, 300]$, the optimal software release time $T^*=T_1=39.626$. Fig. 3 is plotted for graphical illustration.

7. Conclusions

This paper studies testing effort function dependent software FDP and FCP with incorporation of imperfect debugging. Testing resource is usually not constantly allocated during software testing phase, which can largely influence the fault detection rate and the time needed to correct the detected faults. For example, the debugger may spend a week without doing any testing work and work very hard in the following few days. In addition, it is natural for debuggers to make mistakes and introduce new faults during testing. The debuggers tend to introduce more faults if more testing effort is consumed since the code has experienced more changes. In order to capture the influences of testing resource allocation and fault introduction on both FDP and FCP, we first derive FDP incorporating testing effort function and the fault introduction effect, and then obtain FCP as delayed FDP with a correction effort. Various paired FDP and FCP models are obtained based on different assumptions on fault introduction and correction effort. It can be seen that our model is quite general and flexible. Some simpler models are the special cases of our models. An example is presented to illustrate the application of the paired models. The optimal release policy under different criteria is also studied.

Acknowledgments

The research report here was partially supported by the NSFC under Grant nos. 71231001 and 71301009, China Postdoctoral Science Foundation funded project under Grant no. 2013M530531, and by the MOE Ph.D. supervisor fund, 20120006110025.

References

[1] Boland PJ, Chuí NN. Optimal times for software release when repair is imperfect. Stat Probab Lett 2007;77:1176–84.
[2] Cai KY, Cao P, Dong Z, Liu K. Mathematical modeling of software reliability testing with imperfect debugging. Comput Math Appl 2010;59(10):3245–85.
[3] Chang YC, Liu CT. A generalized JM model with applications to imperfect debugging in software reliability. Appl Math Model 2009;33(9):3578–88.
[4] Chiu KC, Huang YS, Lee TZ. A study of software reliability growth from the perspective of learning effects. Reliab Eng Syst Saf 2008;93(10)1410–21.
[5] Demarko T. Controlling software projects: management, measurement and estimation. Englewood Cliffs, NJ: Prentice-Hall; 1982.
[6] Gokhale SS, Lyu MR, Trivedi KS. Incorporating fault debugging activities into software reliability models: a simulation approach. IEEE Trans Reliab 2006;55(2):281–92.
[7] Huang CY. Performance analysis of software reliability growth models with testing-effort and change-point. J Syst Softw 2005;76(2):181–94.
[8] Huang CY, Kuo SY. Analysis and assessment of incorporating logistic testing effort function into software reliability modeling. IEEE Trans Reliab 2002;51(3):261–70.
[9] Hu QP, Xie M, Ng SH, Levitin G. Robust recurrent neural network modeling for software fault detection and correction prediction. Reliab Eng Syst Saf 2007;92(3):332–40.
[10] Hwang S, Pham H. Quasi-renewal time-delay fault-removal consideration in software reliability modeling. IEEE Trans Syst Man Cybern Part A–Syst Hum 2009;39(1):200–9.
[11] Inoue S, Yamada S. Generalized discrete software reliability modeling with effect of program size. IEEE Trans Syst Man Cybern Part A–Syst Hum 2007;37(2):170–9.
[12] Jain M, Gupta R. Optimal release policy of module-based software. Qual Technol Quant Manag 2011;8(2):147–65.
[13] Jha PC, Gupta D, Yang B, Kapur PK. Optimal testing resource allocation during module testing considering cost, testing effort and reliability. Comput Ind Eng 2009;57(3):1122–30.
[14] Jia LX, Yang B, Guo SC, Park DH. Software reliability modeling considering fault correction process. IEICE Trans Inf Syst 2010;E93D(1):185–8.
[15] Kang HG, Lim HG, Lee HJ, Kim MC, Jang SC. Input-profile-based software failure probability quantification for safety signal generation systems. Reliab Eng Syst Saf 2009;94(10):1542–6.
[16] Kapur PK, Goswami DN, Bardhan A, Singh O. Flexible software reliability growth model with testing effort dependent learning process. Appl Math Model 2008;32:1298–307.
[17] Kapur PK, Shatnawi O, Aggarwal A, Kumar R. Unified framework for developing testing effort dependent software reliability growth models. WSEAS Trans Syst 2009;8(4):521–31.
[18] Kapur PK, Pham H, Anand S, Yadav K. A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. IEEE Trans Reliab 2011;60(1):331–40.
[19] Kim HS, Park DH, Yamada S. Bayesian optimal release time based on inflection S-shaped software reliability growth model. IEICE Trans Fundam Electron Commun Comput Sci 2009;E92A(6):1485–93.
[20] Li X, Xie M, Ng SH. Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points. Appl Math Model 2010;34(11):3560–70.
[21] Lin CT, Huang CY. Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models. J Syst Softw 2008;81:1025–38.
[22] Musa JD, Iannino A, Okumono K. Software reliability, measurement, prediction and application. New York: McGraw Hill; 1987.
[23] Okamura H, Dohi T, Osaki S. Software reliability growth models with normal failure time distributions. Reliab Eng Syst Saf 2013;116:135–41.
[24] Parr FN. An alternative to the Rayleigh curve for software development effort. IEEE Trans Softw Eng 1980;6(3):291–6.
[25] Pham H, Nordmann L, Zhang X. A general imperfect software debugging model with s-shaped fault detection rate. IEEE Trans Reliab 1999;48:169–75.
[26] Pievatolo A, Ruggeri F, Soyer R. A Bayesian hidden Markov model for imperfect debugging. Reliab Eng Syst Saf 2012;103:11–21.
[27] Peng R, Li YF, Zhang JG, Li X. A risk-reduction approach for optimal software release time determination with the delay incurred cost. Int J Syst Sci; 2014 http://dx.doi.org/10.1080/00207721.2013.827261.

[28] Schneidewind NF. Analysis of error processes in computer software. Proc Int Conf Reliab Softw. Los Alamitos, CA: IEEE Computer Society Press; 1975; 337–346.
[29] Shyur HJ. A stochastic software reliability model with imperfect-debugging and change-point. J Syst Softw 2003;66:135–41.
[30] Stikkel G. Dynamic model for the system testing process. Inf Softw Technol 2006;48:578–85.
[31] Wu YP, Hu QP, Xie M, Ng SH. Modeling and analysis of software fault detection and correction process by considering time dependency. IEEE Trans Reliab 2007;56(4):629–42.
[32] Xie M, Yang B. A study of the effect of imperfect debugging on software development cost. IEEE Trans Softw Eng 2003;29(5):471–3.
[33] Xie M, Hu QP, Wu YP, Ng SH. A study of the modeling and analysis of software fault-detection and fault-correction processes. Qual Reliab Eng Int 2007;23:459–70.
[34] Yamada S, Tokuno K, Osaki S. Imperfect debugging models with fault introduction rate for software reliability assessment. Int J Syst Sci 1992;23(12):2241–52.
[35] Yang B, Li X, Xie M, Tan F. A generic data-driven software reliability model with model mining technique. Reliab Eng Syst Saf 2010;95(6):671–8.
[36] Zhang XM, Teng XL, Pham H. Considering fault removal efficiency in software reliability assessment. IEEE Trans Syst Man Cybern Part A–Syst Hum 2003;33:114–20.