International Journal of Hybrid Information Technology
Vol.9, No.4 (2016), pp. 407-416
http://dx.doi.org/10.14257/ijhit.2016.9.4.35

Infinite Server Queuing Models with Resource Expenditures and Change-point for Software Reliability

Zhang Nan
Harbin University of Commerce, Harbin 150028, China
zhangnan@hrbcu.edu.cn

Abstract

Software reliability growth models (SRGMs) investigating software debugging process have recently been developed by some researchers to estimate software reliability measures. However, these SRGMs either did not involve resource expenditures spent on fault remove process or assumed that the rate of resources consumed is constant. In practice, software fault intensity may be not continuous for many conditions. Thus, in order to address the problem, this paper develops models incorporating the resource expenditures and change-point, which spent on software debugging process. A real software failure project is demonstrated the effectiveness of proposed models, and numerical results prove new models can provide better fit and prediction.

Keywords: *Software reliability; Detection effort; Correction effort; Change-point.*

Notation

a the expected number of initial faults
b the rate of fault detection per unit detection effort
$m_d(t)$ the mean number of software faults detected in time $(0,t]$
$m_c(t)$ the mean number of software faults corrected in time $(0,t]$
$W_d(t)$ the cumulative detection effort consumption at time t
$W_c(t)$ the cumulative correction effort consumption at time t
$\zeta(t)$ a time-dependent scale function
ζ a constant scale parameter
$p(0,\tau_1]$ the probability that a fault detected at time x_1 is fully removed in time $(x_1,\tau_1]$
$q(\tau_1,t]$ the probability that a fault detected at time x_2 is removed in time $(x_2,t]$
T_d fault detected time
T_c fault removed time
$G_1(\tau_1-x_1)$ the probability fault removed in time $(0,\tau_1]$
$G_2(t-\tau_1)$ the probability fault removed in time $(\tau_1,t]$

1. Introduction

Reliability is one of the key factors of software quality, which is defined as the probability of no software failure occurrence in a specified environment and time [1, 2]. So far, the study on software reliability has been focus on software reliability growth models (SRGMs) [1, 2]. In developing these models, assumptions pertaining to detecting

ISSN: 1738-9968 IJHIT

and removing are made [1, 2]. Some of these assumptions seem reasonable; however, others are questionable for realistic environment [3].

Recently, some researchers studied that a queuing model to express the software debugging processes [4, 5, 6]. For example, Huang et al. [5, 7] used queuing theory to investigate software debugging process. Kapur et al. [6] defined the level complexities of software faults applying queuing models, which considered software fault remove process rate is constant. However, software fault remove is a complex process. The influence factors have the quantity of software fault, the capability of the debuggers, the factor of resource expenditures, and the software operational environment and so on [2]. Furthermore, software resources are one of the important factors during software debugging process [1]. Therefore, it is reasonable to consider the resource expenditures and software fault remove rate change when modeling software debugging process in realistic cases [8].

Based on the foregoing discussion, we present an approach of queue theory to model software debugging process. Namely, software fault detection process which corresponds to the arrival process of queuing model [5, 7, 9]. While, software fault remove process corresponds to the departure process of queuing model [5, 7, 9]. The detection effort and remove effort are defined as resource expenditures spent on the queuing model, respectively. The change-points of software fault remove are investigated. Thus, we will propose server queuing models with detection effort, correction effort and change-point in the paper. The remainder of paper is organized as follows. Section 2 gives detection effort function and correction effort function. Moreover, change-points are also presented in this section. In Section 3, we present infinite sever queuing models considering detection effort function, correction effort function and change-points. Numerical example and simulation results are discussed in Section 4. Section 5 is conclusions.

2. Debugging resource and change-point

2.1. Detection effort function and correction effort function

The distribution of resources consumed in the software debugging process is defined as a consumption curve of software testing effort [10]. Furthermore, experience has studied that fault detection and fault remove are usually performed by different curve in the software debugging phase of a project [1]. In this case, software failure identification and modifiers are comprehended as separate resources [1]. Thus, detection effort and correction effort are defined as the testing effort consumption spent on software debugging process, respectively. Namely, testing effort is composed of detection effort and correction effort. Note that testing effort expenditures are depicted by distribution curves in the real data. Similarly, we also describe both detection effort function and correction effort function by using distribution curves. Let $w_d(t)$ and $w_c(t)$ be the current detection effort consumption and correction effort consumption at time t, respectively. According to detection effort is a part of testing effort, we assume

$$w_d(t) = \zeta(t)w(t) \tag{1}$$

From the relation of $w_d(t)$ and$w_c(t)$, we obtain

$$w_c(t) = [1 - \zeta(t)]w(t) \tag{2}$$

Let us consider the simplest case. Assuming that $\zeta(t) = \zeta(\zeta > 0)$ and substituting it into Eq. (3) and Eq. (4), we have

$$w_d(t) = \zeta w(t) \tag{3}$$

$$w_c(t) = (1 - \zeta)w(t) \tag{4}$$

The ζ can be estimated by the lease squares estimate and is determined for the actual software failure data set.

2.2. Change-point

Most software reliability models considered the rate of software fault remove is a constant. However, in facts, the rate of software fault remove is affected by the difficulty of faults, the ability of the programmers, resource allocation, environments and tools of software remove, etc. Once these factors are changed, the rate of software fault remove may be neither smooth nor constants, and it may change at some moments called change-points.

Recently, some researchers have focus on the problem of change-points in the modeling of software reliability. For example, Chang [11] considered that the rate of fault detection is change in the non-homogeneous Poisson process (NHPP) model. Shyur [12] incorporated both change-point and imperfect debugging problem into the NHPP model. Zhao Jing [13] presented change-point and environmental function in software failure process.

3. Model formulation

Let $\{N_d(t), t \geq 0\}$ be counting processes of software fault detected. There are assumptions in model proposed [5, 7, 9]:

1. In$(t, t + \Delta t]$, the mean number of faults detected by the current detection effort expenditures is proportional to the mean number of remaining faults

2. Faults are mutually independent in software

3. Software fault remove process is considered, and the number of faults removed lags behind total number of faults detected

4. Software debugging process can be defined as an infinite queuing model with the NHPP arrival and general service time distribution

5. The detection effort function and correction effort function are presented by generalized modified Weibull curves

6. It is not introduce a new fault when removing faults

Based on assumptions, we can get the following differential equation [8, 10]

$$\frac{dm_d(t)}{dt} = bw_d(t)[a - m_d(t)] \tag{5}$$

Solving Eq. (5) under boundary condition $m_d(0) = 0$ yields

$$m_d(t) = a[1 - exp(-bW_d(t) + bW_d(0))] \tag{6}$$

Let $\{N_c(t), t \geq 0\}$ be counting processes of software fault removed, $N_c(t)$ be the number of fully removed in time $(0, t]$. If there was one change-point in fault remove process, we define

$$m_c(t) = m_c(0, \tau_1] + m_c(\tau_1, t] \tag{7}$$

In time $(0, \tau_1]$, We can get

$$\begin{aligned} p(0,\tau_1] &= \int_0^{\tau_1} P\{(T_d = x_1) \cap (T_c \le \tau_1 - x_1)\}\mathrm{d}x_1 \\ &= \int_0^{\tau_1} P\{T_d = x_1\} P\{T_c \le \tau_1 - x_1 | T_d = x_1\}\mathrm{d}x_1 \\ &= \int_0^{\tau_1} P\{T_d = x_1\} G_1(\tau_1 - x_1)\mathrm{d}x_1 \end{aligned} \quad (8)$$

In time $(0,\tau_1]$, software fault detected at time x_1 is given by

$$P\{T_d = x_1\} = \frac{m_d'(x_1)}{m_d(\tau_1)} \quad (9)$$

Substituting Eq. (9) into Eq. (8) yields

$$p(0,\tau_1] = \int_0^{\tau_1} \frac{m_d'(x_1)}{m_d(\tau_1)} G_1(\tau_1 - x_1)\mathrm{d}x_1 \quad (10)$$

When the fault was detected in time $(0,\tau_1]$, $m_c(0,\tau_1]$ can be written

$$\begin{aligned} m_c(0,\tau_1] &= m_d(0,\tau_1] p(0,\tau_1] \\ &= m_d(\tau_1) \int_0^{\tau_1} \frac{m_d'(x_1)}{m_d(\tau_1)} G_1(\tau_1 - x_1)\mathrm{d}x_1 \\ &= \int_0^{\tau_1} m_d'(x_1) G_1(\tau_1 - x_1)\mathrm{d}x_1 \end{aligned} \quad (11)$$

Next, $m_c(\tau_1,t]$ consist of two parts: fault detected in time $(0,\tau_1]$ was removed in time $(\tau_1,t]$; fault detected in time $(\tau_1,t]$ was removed in time $(\tau_1,t]$.

In time $(\tau_1,t]$, remain software fault removed is

$$[m_d(\tau_1) - m_c(\tau_1)] G_2(t - \tau_1) \quad (12)$$

In time $(\tau_1,t]$, we have

$$\begin{aligned} q(\tau_1,t] &= \int_{\tau_1}^{t} P\left\{(T_d = x_2) \cap (T_c \le t - x_2)\right\}\mathrm{d}x_2 \\ &= \int_{\tau_1}^{t} P\left\{T_d = x_2\right\} P\left\{T_c \le t - x_2 | T_d = x_2\right\}\mathrm{d}x_2 \\ &= \int_{\tau_1}^{t} P\left\{T_d = x_2\right\} G_2(t - x_2)\mathrm{d}x_2 \end{aligned} \quad (13)$$

In time $(\tau_1,t]$, a software fault detected at time x_2 can be written

$$P\{T_d = x_2\} = \frac{m_d'(x_2)}{m_d(t) - m_d(\tau_1)} \quad (14)$$

Substituting Eq. (14) into Eq. (13) yields

$$q(\tau_1,t] = \int_{\tau_1}^{t} \frac{m_d'(x_2)}{m_d(t) - m_d(\tau_1)} G_2(t - x_2)\mathrm{d}x_2 \quad (15)$$

If a fault was detected in time $(\tau_1,t]$, we can obtain

$$\begin{aligned} m_d(\tau_1,t] q(\tau_1,t] &= [m_d(t) - m_d(\tau_1)] \int_{\tau_1}^{t} \frac{m_d'(x_2)}{m_d(t) - m_d(\tau_1)} G_2(t - x_2)\mathrm{d}x_2 \\ &= \int_{\tau_1}^{t} m_d'(x_2) G_2(t - x_2)\mathrm{d}x_2 \end{aligned} \quad (16)$$

From Eq. (12) and Eq. (16), we have

$$m_c(\tau_1,t] = [m_d(\tau_1) - m_c(\tau_1)] G_2(t - \tau_1) + \int_{\tau_1}^{t} m_d'(x_2) G_2(t - x_2)\mathrm{d}x_2 \quad (17)$$

From Eq. (7), Eq. (11) and Eq. (17), we can get

$$\begin{aligned} m_c(t) &= m_c(0,\tau_1] + m_c(\tau_1,t] \\ &= \int_0^{\tau_1} m_d'(x_1)G_1(\tau_1 - x_1)\mathrm{d}x_1 + \int_{\tau_1}^{t} m_d'(x_2)G_2(t - x_2)\mathrm{d}x_2 + [m_d(\tau_1) - m_c(\tau_1)]G_2(t - \tau_1) \end{aligned} \quad (18)$$

Specially, $\tau_1 = 0$ Eq. (18) can be written

$$m_c(t) = \int_0^{t} m_d'(x_2)G_2(t - x_2)\mathrm{d}x_2 \quad (19)$$

Meanwhile, if there were two change-points in fault remove process, we can obtain

$$\begin{aligned} m_c(t) &= m_c(0,\tau_1] + m_c(\tau_1,\tau_2] + m_c(\tau_2,t] \\ &= \int_0^{\tau_1} m_d'(x_1)G_1(\tau_1 - x_1)\mathrm{d}x_1 \\ &+ \int_{\tau_1}^{\tau_2} m_d'(x_2)G_2(\tau_2 - x_2)\mathrm{d}x_2 + [m_d(\tau_1) - m_c(\tau_1)]G_2(\tau_2 - \tau_1) \\ &+ \int_{\tau_2}^{t} m_d'(x_3)G_3(t - x_3)\mathrm{d}x_3 + [m_d(\tau_2) - m_c(\tau_2)]G_3(t - \tau_2) \end{aligned} \quad (20)$$

where $0 \le \tau_1 \le \tau_2 \le t$, $m_d(0) = m_c(0) = 0$.

Here, we consider that the number of allocated debuggers is infinite resource and first come, first served. In addition, human queue service time can be modeled as a birth- death process where a birth is a detected fault and a death is a removed fault [9]. Thus, we assume [5, 7, 9]. $G(t)$ is given by $G_{k+1}(x_{k+1}) = 1 - \exp[-\rho_{k+1}W_c(\tau_{k+1}) + \rho_{k+1}W_c(\tau_{k+1} - x_{k+1})]$, where ρ is a constant rate of fault remove. Substituting into Eq. (18) and Eq. (20) yields

$$\begin{aligned} m_c(t) &= \int_0^{\tau_1} m_d'(x_1)\left[1 - \exp(-\rho_1 W_c(\tau_1) + \rho_1 W_c(x_1))\right]\mathrm{d}x_1 \\ &+ \int_{\tau_1}^{t} m_d'(x_2)\left[1 - \exp(-\rho_2 W_c(t) + \rho_2 W_c(x_2))\right]\mathrm{d}x_2 \\ &+ [m_d(\tau_1) - m_c(\tau_1)]\left[1 - \exp\left(-\rho_2 W_c(t) + \rho_2 W_c(\tau_1)\right)\right] \end{aligned} \quad (21)$$

$$\begin{aligned} m_c(t) &= \int_0^{\tau_1} m_d'(x_1)\left[1 - \exp(-\rho_1 W_c(\tau_1) + \rho_1 W_c(x_1))\right]\mathrm{d}x_1 \\ &+ \int_{\tau_1}^{\tau_2} m_d'(x_2)\left[1 - \exp(-\rho_2 W_c(\tau_2) + \rho_2 W_c(x_2))\right]\mathrm{d}x_2 \\ &+ [m_d(\tau_1) - m_c(\tau_1)]\left[1 - \exp\left(-\rho_2 W_c(\tau_2) + \rho_2 W_c(\tau_1)\right)\right] \\ &+ \int_{\tau_2}^{t} m_d'(x_3)\left[1 - \exp(-\rho_3 W_c(t) + \rho_3 W_c(x_3))\right]\mathrm{d}x_3 \\ &+ [m_d(\tau_2) - m_c(\tau_2)]\left[1 - \exp\left(-\rho_3 W_c(t) + \rho_3 W_c(\tau_2)\right)\right] \end{aligned} \quad (22)$$

4. Numerical example and simulation results

4.1.Data description

We used a real software project to illustrate proposed models [1]. Table 1 gives the complete data set.

Table 1. System T1 of the RADC Project.

Weeks	CPU Hour	Identified faults	Corrected faults
1	0.00917	2	1
2	0.010	0	1
3	0.003	0	0
4	0.023	1	1
5	0.041	1	1
6	0.004	2	0
7	0.025	1	1
8	0.302	9	2
9	0.973	13	6
10	0.020	2	4
11	0.450	11	1
12	0.250	2	14
13	0.94	11	5
14	1.34	14	19
15	3.32	18	19
16	3.56	12	10
17	2.66	12	12
18	3.77	15	20
19	3.40	6	12
20	2.40	3	2
21	1.80	1	5

We use the Laplace test to analyze the trend tests of reliability growth [2]. The Laplace factor can be written as [2]

$$u(j) = \frac{\sum_{i=1}^{j}(i-1)n(i) - \frac{(j-1)}{2}\sum_{i=1}^{j} n(i)}{\sqrt{\frac{(j^2-1)}{12}\sum_{i=1}^{j} n(i)}} \tag{23}$$

In commonly, C chart is used to monitor the software process. Control limits are estimated in C chart, and the parameters of C chart are defined as Eq. (24), Eq. (25) and Eq. (26).

$$\overline{C} = \frac{\sum_{i=1}^{k} n(i)}{k} \tag{24}$$

$$UCL = \overline{C} + 3\sqrt{\overline{C}} \tag{25}$$

$$LCL = \overline{C} - 3\sqrt{\overline{C}} \tag{26}$$

We used the Laplace factor and C chart to calculate the change-point. Figure 1 shows the Laplace trend results. The C chart results are shown in Figure 2.

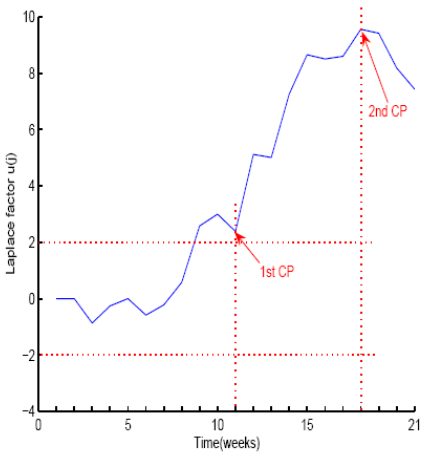

Figure 1. Laplace trend analysis

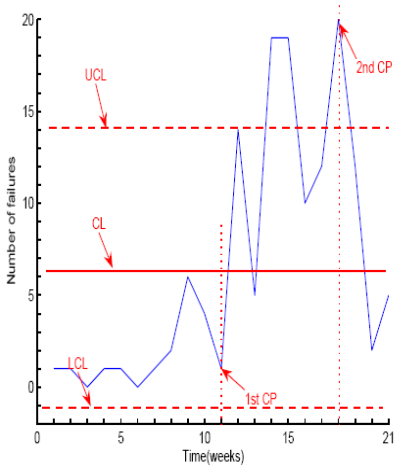

Figure 2. C chart

From Figure 1, we can see the Laplace factors are almost always negative value, which denotes a growth in reliability. As seen from Figure 2, we observe that two points falls outside the Upper Control Line (UCL) or Upper Control Line (LCL), this may be two change-points. However, researchers suggest that the numbers of change-points are not too many. Thus, this paper chooses one change-point.

4.2 Comparison criteria

The sum of squares for error (SSE) and the R-square are used to measure the differences between predicted values and actual observations, which are written as follows [1, 14, 15]:

$$\mathrm{SSE} = \sum_{i=1}^{n} (m_i - \hat{m}(t_i))^2 \tag{27}$$

$$\text{R-square} = \frac{\sum_{i=1}^{n} (\hat{m}(t_i) - \frac{1}{n}\sum_{i=1}^{n} m_i)^2}{\sum_{i=1}^{n} (m_i - \frac{1}{n}\sum_{i=1}^{n} m_i)^2} \tag{28}$$

4.3 Performance analysis

First, the formulations of some models are shown in Table 2. Second, we use the least square estimation to estimate the unknown parameters of the queuing models in the paper, which are listed in Table 3 when used to System T1 data set mentioned above. Table 4 lists the result of SSE and R-square computed by using the estimated parameters listed in Table 3. From Table 4, the queuing model with detection effort function, correction effort function and change-point provides the lowest value of SSE_d and SSE_c, the highest value of R-$square_d$ and R-$square_c$, and better goodness-of-fit when compared with other software reliability growth models. Therefore, the queuing models are suitable for depiction the software reliability and the fitted distribution curves are highly significant for this data set.

Table 2. Mean value functions of comparison of goodness-of-fit of S-RGMs

Model	mean value functions
1.	$W_d(t) = \zeta\alpha\left[1-\exp(-\beta t^m \exp(-\lambda t))\right]^\theta$ $W_c(t) = (1-\zeta)\alpha\left[1-\exp(-\beta t^m \exp(-\lambda t))\right]^\theta$ $m_d(t) = a[1-\exp(-bW_d^*(t))]$ $m_c(t) = \int_0^{\tau_1} m_d'(x_1)[1-\exp(-\rho_1 W_c(\tau_1)+\rho_1 W_c(x_1))]\mathrm{d}x_1$ $+\int_{\tau_1}^{t} m_d'(x_2)[1-\exp(-\rho_2 W_c(t)+\rho_2 W_c(x_2))]\mathrm{d}x_2$ $+[m_d(\tau_1)-m_c(\tau_1)][1-\exp(-\rho_2 W_c(t)+\rho_2 W_c(\tau_1))]$
2.	$W_d(t) = \zeta\alpha\left[1-\exp(-\beta t^m \exp(-\lambda t))\right]^\theta$ $W_c(t) = (1-\zeta)\alpha\left[1-\exp(-\beta t^m \exp(-\lambda t))\right]^\theta$ $m_d(t) = a[1-\exp(-bW_d^*(t))]$ $m_c(t) = \int_0^t m_d'(x)\left[1-\exp(-\rho W_c(t)+\rho W_c(x))\right]\mathrm{d}x$
3.	$W(t) = \alpha[1-\exp(-\beta t^m \exp(-\lambda t))]^\theta$ $m(t) = a[1-\exp(-bW^*(t))]$

Table 3. Parameters estimation of SRGMs based on System T1 data set

Model	a	b	ζ	ρ_1	ρ_2
1.	138.6	0.3749	0.3521	0.2156	0.01872
2.	133.1	0.4002	0.4002	0.9837	-
3.	140.5	0.1502	-	-	-

Table 4. Mean value functions of comparison of goodness-of-fit of SRGMs

Model	SSE_d	SSE_c	$R-square_d$	$R-square_c$
1.	1075	610.7	0.9729	0.9756
2.	1673	1417	0.9677	0.9715
3.	1673	1673	0.964	0.964

5. Conclusions

This paper has incorporated the works of detection effort function, correction effort function and change-point into software debugging process, respectively. From a somewhat different perspective other than the traditional approaches, the infinite queuing model is used to define software debugging process. Compared with the existing other models, the new queuing models can give a better result of goodness-of-fit for this software failure data set.

Acknowledgements

This work is supported by the Harbin University of Commerce under Grant (No. 15RW21).

References

[1] J. D. Musa, A. Jannino, and K. Okumoto, Software Reliability, Measuremen, Prediction, Application, (McGram Hill, New York, 1987).
[2] M. R. Lyu, Handbook of Software Reliability Engineering (McGram Hill, New York, 1996).
[3] A. L. Goel, Software Reliability Models: assumptions, limitations, and applicability, IEEE Trans. on Software Engineering SE-11 (12) (1985) 1411-1423.
[4] T. Dohi, S. Osaki, and K. S. Trivedi, An infinite server queuing approach for describing software reliability growth: unified modeling and estimation framework, in Proc.11th Conf. on Asia-Pacific Software Engineering, Busan, Korea, 2004.
[5] C. Y. Huang and T. Y. Huang, Software reliability analysis and assessment using queuing models with multiple change-points, Computers and Mathematics with Applications 60 (7) (2010) 2015-2030.
[6] P. K. Kapur, S. Anada, S. Inoue, and S.Yamada, A unified approach for developing software reliability growth model using infinite server queuing model, International J. of Reliability Quality Safety Engineer 17 (5) (2010) 401-424.
[7] C. Y. Huang and W. C. Huang, Software reliability analysis and measurement using finite and infinite server queuing models, IEEE Trans. on Reliability 57 (1) (2008) 192-203.
[8] R. Peng, Q. P. Hu, S. H. Ng, and M. Xie, Testing effort dependent software FDP and FCP models with consideration of imperfect debugging, in Pro. 4th IEEE International Conf. on Secure Software Integration and Reliability Improvement, 2010.
[9] K. S. Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Application,(John Wiley and Sons, 2002).
[10] S. Yamada, J. Hishitani, and S. Osaki, Software reliability growth with a weibull test effort a model and application, IEEE Trans. on Reliability 42 (1) (1993) 100-105.
[11] Y P Chang. Estimation of Parameters for Non-homogeneous Poisson Process Software
[12] Reliability with Change Point Model, Communications in Statistics Simulation and Computation30 (3) (2001) 623-635.
[13] H. J. Shyur. A Stochastic Software Reliability Model with Imperfect Debugging and Change Point, Journal of System and Software 66 (2) (2003) 135-141.
[14] J. Zhao, H. W. Liu, G. Cui and X. Z. Yang, Software reliability growth model with change-point and environmental function, The Journal of Systems and Software 79 (3) (2006) 1578–1587.
[15] C. Y. Huang and S. Y. Kuo, Analysis of incorporating logistic testing-effort function into software reliability modeling, IEEE Trans. on Reliability 51 (3) (2002) 261-270.
[16] C. Y. Huang, Performance analysis of software reliability growth models with testing effort and change-point, The Journal of System and Software 76 (2) (2005) 181-194.

