http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 8 (5): 735-742, 2009
ISSN 1812-5638

Modeling of Software Fault Detection and Correction Processes Based on the Correction Lag

Yanjun Shu, Hongwei Liu, Zhibo Wu and Xiaozong Yang
School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China

Abstract: This study presents a software reliability growth model integrating the fault detection process with the fault correction process. Although, a few research projects have been devoted to the modeling of these two processes, most of them studied the correction lag from a theoretical viewpoint of time delay. In this study, the correction lag is characterized by the remaining uncorrected faults which can be clearly observed from the actual data. Through analyzing its varying trend, the Gamma curve is found to be appropriate in representing the correction lag function. Then, the proposed model is derived. Two real data sets of software testing are used to evaluate models. Experimental results indicate that the proposed model not only provides better performance than other models on both fault detection and correction processes, but also does better in describing the correction lag. Finally, a revised software cost model is presented based on the proposed model. From the analysis on the determination of software release time, the new cost model shows more practical than the traditional approach.

Key words: Fault detection process, fault correction process, correction lag, software reliability growth model, software release time

INTRODUCTION

As one of the most important aspects of software quality, software reliability is often defined as the probability of failure-free software operation for a specified period of time in a specified environment (Musa *et al*., 1987; Lyu, 1996; Gokhale *et al*., 2004; Williams, 2006). Many Software Reliability Growth Models (SRGMs) have been developed to describe the error-detection process and estimate the growth reliability of products during software development process (Musa *et al*., 1987; Lyu, 1996; Pham and Zhang, 1999; Zhao *et al*., 2006). Most existing SRGMs consider only the software fault detection process in the testing stage, assuming immediate fault correction activity (Schneidewind, 2001; Xie *et al*., 2007; Huang and Huang, 2008). However in practice, each detected fault is reported, located and then corrected. The software fault correction process plays a significant role in improving software reliability and ensures software quality (Schneidewind, 2003; Huang and Lin, 2006). Thus, the fault correction process cannot be simply treated as an immediately achieved process. In addition to predicting and assessing software failure occurrence, it is also meaningful for analyzing and modeling the software fault correction.

Recently, some researchers have emphasized the great importance of fault correction process modeling. Since, the fault correction process is usually lagging behind the fault detection process, the correction lag is a common phenomenon in the software testing. Schneidewind (2001, 2003) first modeled the fault correction process as a constant delay fault detection process. Within this framework, some extensions have been proposed by substituting various time-dependent delay functions for the constant delay (Huang and Lin, 2006; Lo, 2007; Xie *et al*., 2007). The introduction of time-delay functions is very useful to separate the fault correction from the fault detection. It can help people find more useful models. Wu *et al*. (2007) summarized several kinds of time-delay distribute functions in constructing the fault correction process models. Following the similar idea of time delay, some other attempts have also been made to model the fault detection and correction process. For instance, Huang and Huang (2008) incorporated server queuing models into the fault correction modeling by considering imperfect debugging; Gokhale *et al*. (2004, 2006) modeled the fault detection and correction processes with a non-homogeneous Markov chain. Lo and Huang (2006) proposed a framework to integrate two processes by the simultaneous differential equations. Kupar *et al*. (2008) indicated that there are two types of detected faults in the software which need different testing effort for correction. They used the simultaneous differential equations to integrate the two processes.

In the work mentioned earlier, the correction lag has been treated as a time delay between the fault detection

Corresponding Author: Yanjun Shu, Harbin Institute of Technology, No. 92, West Da Zhi Street, P.O. Box 415, Harbin, 150001, China

and correction processes. It is represented by many kinds of time-delay functions. However, these time-delay functions have different forms because they are established under different assumptions. In reality, the time to remove a fault depends on many factors such as the number of the detected faults, the complexity of the software structure, the skill of the correction personnel, etc. It seems difficult to choose an accurate time-delay function to describe the correction lag in various testing circumstances. In reality, as a fault must be corrected and removed after its detection, the number of corrected faults is naturally depended on the number of detected faults. Since the correction lag cannot be ignored, some detected but not corrected faults exist in the software. These latent faults are caused by the correction lag and reflect the relationship between the fault detection and correction processes. We can use them to integrate the two processes together.

In this study, the correction lag is defined as the difference between the number of detected faults and the number of corrected faults. We are going to model both fault detection and correction processes based on this lag.

THE CORRECTION LAG

The purpose of software testing is finding and correcting faults before the software product delivery to the customs. When a failure is detected, the fault correction personnel need a period of time to locate the fault and modify some codes to remove it. The correction lag is a common phenomenon in software testing. Some detected but not corrected faults exist in the software. We call them the remaining uncorrected faults in this study. These faults directly represent the correction lag between the fault detection and correction processes. The number of remaining uncorrected faults is the difference between the number of detected faults and the number of corrected faults, which is $\varphi(t) = m_d(t) = m_c(t)$.

Xie *et al.* (2007) used an actual set of data from the testing on a middle size software project to study the fault detection and correction processes with several kinds of time-delay distribute functions. Figure 1 plots the number of detected, corrected, remaining uncorrected faults of this data set. It is analyzed that the number of detected faults varies like the number of corrected faults means they are dependent on each other. Moreover, the number of remaining uncorrected faults is not constant, increasing from the beginning of software testing and decreasing to minimum value at the later stage. Musa *et al.* (1987) reported a data set of System T1 which includes the number of detected and corrected faults. System T1 was

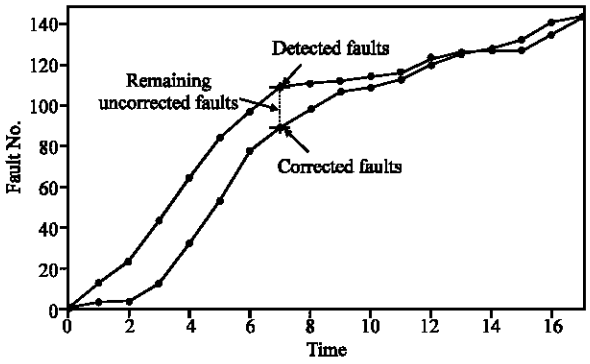

Fig. 1: The number of detected and corrected faults of a middle size software project

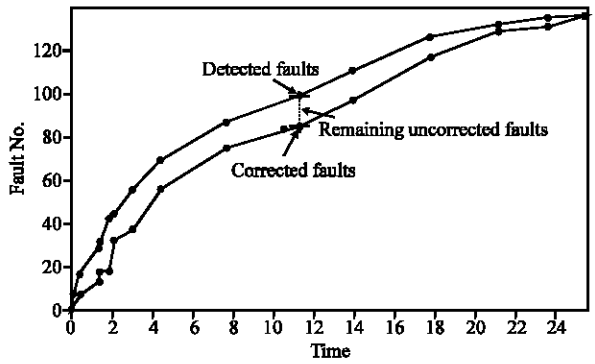

Fig. 2: The number of detected and corrected faults of system T1

used for a real-time command and control application. Figure 2 depicts the number of detected and corrected faults of System T1. Figure 2 shows that the number of remaining uncorrected faults has an increasing and then decreasing scenario. Next, we will discuss about why the correction lag has such a varying trend.

At the beginning of software testing, there are many faults in the software and they are easy to be detected. While in this time, the effective debugging is not easy, the testing personnel need time to read and analyze the failure report to find out the root cause of failure (Huang and Lin, 2006). Thus, the correction process gradually falls far behind the fault detection process and the correction lag is increasing. As testing proceeds, Fig. 1 and 2 show that the correction lag is decreasing. Although, the number of remaining uncorrected faults may not be monotonously decreasing versus time, it is approximately reduced at the latter stage of testing. During this time, faults in the software are almost completely detected and the number of detected faults increases very slowly. Simultaneity, the fault correction personnel are engaging to correct each detected fault.

Thus, the difference between the number of detected faults and the number of corrected faults is decreasing. Generally speaking, although, some of non-critical detected faults might be postponed to be corrected until next release, most of detected faults should be corrected before software release. A large number of remaining uncorrected faults cannot be acceptable for both software developers and users. It is natural to see that the number of remaining uncorrected faults is decreasing to the minimum value at the end of testing process. From the analysis above, it can conclude that the number of remaining uncorrected faults has an increasing and then decreasing scenario. This feature of correction lag is also testified by experimental results from Huang's server queuing models (Huang and Huang, 2008).

To characterize the varying trend of the remaining uncorrected faults, we use the Gamma curve and other curves (such as Weibull, Raleigh, Lognormal curves) are used as the correction lag function to fit it on these two data sets. Through simulation, we find out that the Sum of Squares Errors (SSE) value of the Gamma curve on these data sets are 93.89 and 169.48, which are the lowest SSE value compared with other curves. As the Gamma curve has the best goodness of fit, we select the Gamma curve as the correction lag function:

$$\varphi(t) = ct^{k-1}e^{-\frac{t}{\theta}} \tag{1}$$

SOFTWARE RELIABILITY GROWTH MODELING

Here, the fault detection and correction processes are modeled based on Non-Homogeneous Poison Process (NHPP). The correction lag between fault detection and correction processes is characterized by the remaining uncorrected faults. As studied in the former section, the number of remaining uncorrected faults has an increasing and then decreasing feature and the Gamma curve is selected to represent it. The followings are the explicit assumptions for both fault detection and correction processes models:

- The failure observation/fault correction phenomenon is modeled by NHPP
- The software system is subject to failures during execution caused by faults remaining in the software
- All faults are independent and equally detectable
- The software failure rate at any time is affected by the number of faults remaining in the software at that time
- Each time a failure occurs, the fault that caused it is not immediately removed because the correction lag φ exists in the debugging process

According to these assumptions, we have:

$$\frac{dm_d(t)}{dt} = b(t)[a(t) - m_c(t)] \tag{2}$$

and

$$\varphi(t) = m_d(t) - m_c(t) = ct^{k-1}e^{-\frac{t}{\theta}} \tag{3}$$

To solve the simultaneous differential equations simply and easily, we assume a(t), b(t) are constant, which means no fault is introduced and the failure rate is not changing during the software testing. So, Eq. 2 is changed into:

$$\frac{dm_d(t)}{dt} = b[a - m_c(t)] \tag{4}$$

Equation 3 is directly deduced from assumption 5. It can be transformed as:

$$m_c(t) = m_d(t) - ct^{k-1}e^{-\frac{t}{\theta}} \tag{5}$$

Substituting Eq. 5 into 4, we obtain a new differential equation as:

$$\frac{dm_d(t)}{dt} = b\{a - [m_d(t) - ct^{k-1}e^{-\frac{t}{\theta}}]\} \tag{6}$$

When c = 0, then

$$\varphi(t) = ct^{k-1}e^{-\frac{t}{\theta}} = 0$$

It means the correction lag does not exist in the software testing and the fault correction model is just equal to the fault detection model, $m_d(t) = m_c(t)$. In this case, by solving the Eq. 6 with marginal conditions $m_d(0) = 0$ and $m_d(+\infty) = a$, we can get $m_d(t) = m_c(t) = m_c(t) = m_c(t) = a(1\text{-}e^{-bt})$, the model becomes the Goel-Okumoto (G-O) model, which is a fundamental SRGM. Otherwise $c \neq 0$, by solving Eq. 6, we get a new SRGM for the fault detection process:

$$m_d(t) = a[1 - e^{-bt}] + bce^{-bt}\sum_{n=0}^{\infty}\frac{(b-\frac{1}{\theta})^n t^{n+k}}{n!(n+k)} \tag{7}$$

From Eq. 5 and 7, we can get the fault correction model which paired with the fault detection model as:

$$m_c(t) = a[1 - e^{-bt}] + bce^{-bt}\sum_{n=0}^{\infty}\frac{(b-\frac{1}{\theta})^n t^{n+k}}{n!(n+k)} - ct^{k-1}e^{-\frac{t}{\theta}} \tag{8}$$

MODEL EVALUATION

Here, we will evaluate the proposed model for both fault detection and correction processes by using two real data sets. The first data set (DS1) is from a middle size software project reported by Xie *et al.* (2007) and the second data set (DS2) is from System T1 of the Roma Air Development Center project (Musa *et al.*, 1987). Table 1 summarizes some existing SRGMs which modeled both fault detection and correction processes. These models are from Schneidewind *et al.* (2001, 2003), Xie *et al.* (2007), Lo *et al.* (2007) and Huang and Huang (2008).

Parameter estimation: With the model which combined the fault detection process with the fault correction process, a software testing can be described more practically than conventional SRGMs. There are unknown parameters in the model and they can be estimated by using the method of least squares. Against observations of fault detection and correction processes, the parameters of the new model are estimated by minimizing the Sum of Square Residual (SSR) (Xie *et al.*, 2007), which is defined as follows:

$$SSR = \sum_{k=1}^{n}\left[(m_d(t_k) - yd_k)^2 + (m_c(t_k) - yc_k)^2\right] \quad (9)$$

where, yd_k and yc_k denote the cumulative number of detected and corrected faults collected until t_k respectively, t_k, k = 1,2,…, are the running times from the begging of testing.

Substituting $m_d(t)$ and $m_c(t)$ into Eq. 9 with a data set which includes the fault detection and correction processes, then the equation of SSR is obtained. If we want to estimate the parameters of the proposed model on the data set, the proposed model is substituted into Eq. 9 and the minimum value of SSR can be deduced by solving the differential equations:

$$\frac{\partial SSR}{\partial a} = \frac{\partial SSR}{\partial b} = \frac{\partial SSR}{\partial \eta} = \frac{\partial SSR}{\partial \theta} = 0$$

The parameters of the proposed model can be estimated as the results of the differential equations. Commonly, numerical procedures have to be developed in order to obtain the Least Squares Estimates (LSEs).

Criteria for model comparison: In the following comparison criteria expressions, every variable or function is used for both fault detection and correction processes:

Table 1: Summarization of models

Model name	MVFs
Schneidewind model	$m_d(t) = a(1-e^{-bt})$
	$m_c(t) = a(1-e^{-b(t-c)})$
Xie model	$m_d(t) = a(1-e^{-bt})$
	$m_c(t) = a(1-(1+ct)e^{-bt})$
Lo model	$m_d(t) = a(1-e^{-bt})$
	$m_c(t) = a(1-\frac{\mu e^{-bt}}{\mu - b} + \frac{b e^{-\mu t}}{\mu - b})$
Huang mode	$m_d(t) = \frac{a(1-e^{-bt})}{1+ce^{-bt}}$
	$m_c(t) = \frac{a(1-e^{-bt})}{1+ce^{-bt}}$
	$-a\frac{c(e^{-\mu t}-1)-(1+c)(1+e^{-\mu t})[\log(1+c)-\log(c+e^{bt})]}{c+e^{bt}}$
Proposed model	$m_d(t) = a[1-e^{-bt}] + bce^{-bt}\sum_{n=0}^{\infty}\frac{(b-\frac{1}{\theta})^n t^{n+k}}{n!(n+k)}$
	$m_c(t) = a[1-e^{-bt}] + bce^{-bt}\sum_{n=0}^{\infty}\frac{(b-\frac{1}{\theta})^n t^{n+k}}{n!(n+k)} - ct^{k-1}e^{-\frac{1}{\theta}}$

- The goodness of fit of the curve is measured by the sum of squares errors, SSE. Sum of Square residual SSE sums up the squares of the residuals between the actual data and the mean value function m(t) of each model in terms of the number of actual faults at any time. The SSE function can be expresses as follows:

$$SSE = \sum_{k=1}^{n}(y_k - \hat{m}(t_k))^2 \quad (10)$$

where, y_k is the total number of faults observed at time t_k according to the testing time and $\hat{m}(t_k)$ is the estimated cumulative number of faults at time t_k obtained from the fitted value function, k = 1,2,..n. Therefore, the lower the SSE value, the better the model performs

- After the proposed model is fitted to the actual observed data, the deviation between observed and fitted values can also be evaluated by a using the Kolmogorov-Smirov (K-S) test. The Kolmogorov-Distance (KD) is defined as:

$$D_k = Sup_k|F^*(x) - F(x)| \quad (11)$$

Where, k is the sample size, F*(x) is the normalized observed cumulative distribution at x-point and F(x) is the expected cumulative distribution at x-point, based on the model

- The capability of the model to predict failure behavior from present and past failure behavior is called predictive validity, which can be represented by computing the Mean value of Relative Errors (MRE) for a data set:

$$MRE = \frac{1}{l}\sum_{t_z=t_1}^{t_2}\left|\frac{\hat{m}(t_z)-Z}{Z}\right| \quad (12)$$

Suppose Z failures have been observed by the end of testing time t_z, we use the failure data up to time $t_i(t_i \le t_z)$ to estimate the parameters of m(t). Moreover, a lower value of MRE indicates a better performance of fitting the real software data

Experiments on DS1: We estimate the parameters of the selected models and the proposed models by using the LSEs method on DS1. The estimated parameters of all models are listed in Table 2. Table 3 gives the performance comparisons in terms of SSE, KD and MRE. Table 3, show the proposed model provides the lowest value of all criteria only except the KD value on the detection process. On the average, it can figure out that the proposed model gives a better fit and prediction in both detection and correction processes than other models. As can be seen from Table 3, the proposed model, Schneidewind model and Lo model do well in fitting this data set on both fault detection and correction processes. Figure 3 and 4 depict the estimated data by these three models. We can see that all of these three models have a good performance on DS1.

It is also calculate the estimated number of remaining uncorrected faults by using the proposed model, Schneidewind model and Lo model. Figure 5 graphically shows the actual and estimated number of remaining uncorrected faults versus time. The difference between detected and corrected faults is initially zero at the beginning of software testing and it is increasing rapidly as test proceeds. This phenomena means effective debugging is difficult. The testing personnel need time to analyze the failure report and find the root of failure. The correction process falls far behind the fault detection process and the remaining uncorrected faults increase. After reaching the maximum value, the number of remaining uncorrected faults is decreasing. At this time, faults in the software are almost completely detected, the detected fault number increases very slowly. On the other hand, the testing personnel master the whole software system better than before the detected faults can be corrected in time. So, the corrected fault number gets gradually closer to the detected fault number. The remaining uncorrected faults decrease quickly to the minimum value. All three models can characterize the increasing and then decreasing feature of remaining uncorrected faults. Furthermore, the proposed model is the best one for representing the correction lag and it also has the best goodness of fit on DS1. Therefore, it conclude that using the remaining uncorrected faults is a better way to integrate the fault detection and correction processes than using the time delay functions.

Table 2: Parameter estimation of different SRGMs on DS1

Models	a	b	c	μ	θ
Schneidewind model	153.01	0.1487	1.94	-	-
Xie model	168.36	0.1193	0.028	-	-
Lo model	156.34	0.1404	-	0.5811	-
Huang model	148.59	0.2684	1.27	0.2562	-
Proposed model	152.05	0.1349	27.22	-	1.60

Table 3: Comparison results on DS1

	SSE		KD		MRE	
Models	Detected	Corrected	Detected	Corrected	Detected	Corrected
Schneidewind model	884.17	510.00	0.1064	0.1283	0.1396	0.1879
Xie model	987.02	2578.90	0.1568	0.1303	0.1228	0.9358
Lo model	861.22	1015.24	0.1203	0.1225	0.1324	0.4435
Huang model	1682.60	2189.20	0.1225	0.1662	0.1938	0.2094
Proposed model	824.61	410.64	0.1221	0.0785	0.1066	0.1701

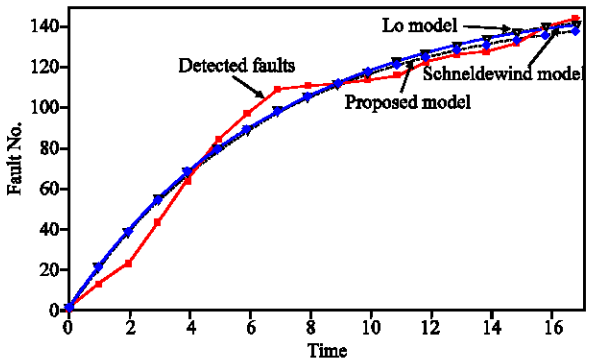

Fig. 3: Detected fault data on DS1: fitted versus actual

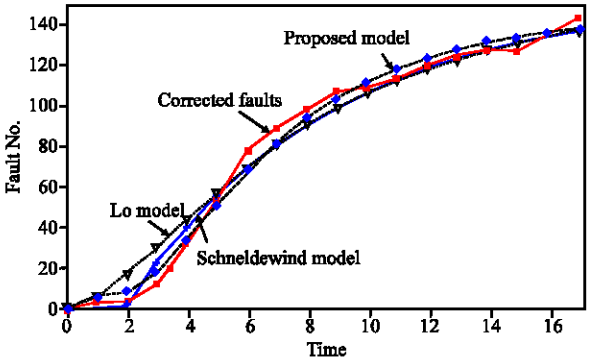

Fig. 4: Corrected fault data on DS1: fitted versus actual

Experiments on DS2: Similarly, we also estimated the parameters of the proposed model and other selected SRGMs by using the methods of LSEs on DS2. The estimated parameters are listed in Table 4. Table 5 lists the comparisons results in terms of SSE, KD and MRE. The

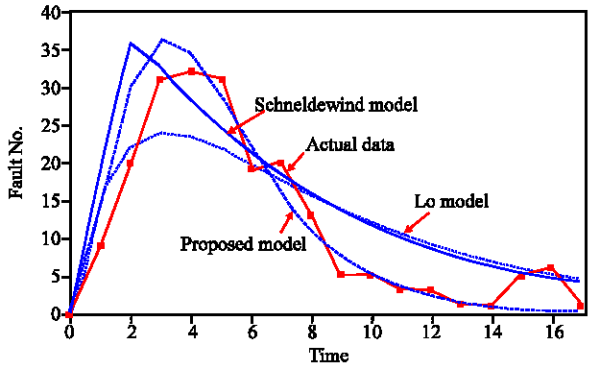

Fig. 5: Actual/estimated remaining uncorrected fault for DS1

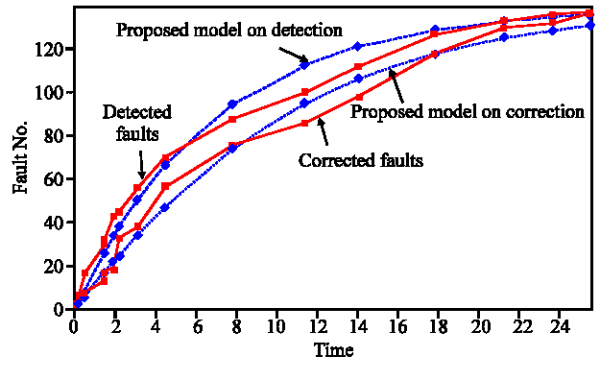

Fig. 6: Detected and corrected fault data on DS2: fitted versus actual

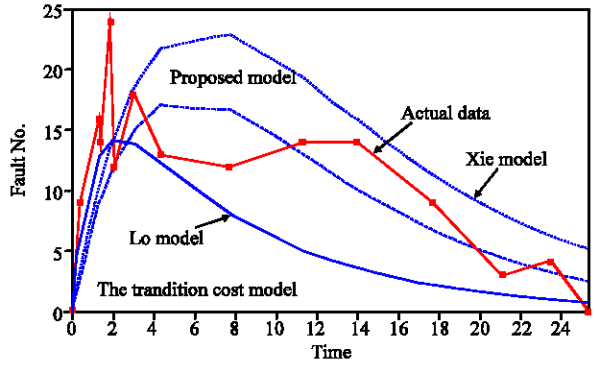

Fig. 7: Actual/estimated remaining uncorrected fault on DS

proposed model has the lowest value of SSE, MRE on both fault detection and correction processes. Although, the proposed model does not provide the lowest value of KD on the two processes, it also gives very close value to them. Figure 6 shows the estimated data by the proposed model versus the actual data set. It can see that the proposed model fit the data set very well in both detection and correction processes. Overall, the proposed model gives a better fit and prediction on DS2.

Table 4: Parameter estimation of different SRGMs on DS2

Models	a	b	c	μ	θ
Schneidewind model	138.17	0.1351	1.00	-	-
Xie model	139.60	0.1540	0.07	-	-
Lo model	138.70	0.1332	-	0.9422	-
Huang model	138.68	0.2084	1.17	0.3762	-
Proposed model	138.65	0.1351	8.44	-	5.69

Table 5: Comparison results on DS2

	SSE		KD		MRE	
Models	Detected	Corrected	Detected	Corrected	Detected	Corrected
Schneidewind model	694.52	1209.90	0.0904	0.1545	0.3418	0.4924
Xie model	754.35	494.32	0.1128	0.1089	0.3273	0.3465
Lo model	714.86	1027.90	0.0985	0.1501	0.3441	0.4429
Huang model	1744.40	660.25	0.1278	0.0983	0.4163	0.3385
Proposed model	688.39	494.72	0.0968	0.1019	0.3206	0.3329

As can be seen from Table 4, the proposed model, Xie model and Lo model do well in fitting DS2. Figure 7 graphically shows the actual and estimated number of remaining uncorrected faults versus time. We can see that the proposed model can characterize the actual remaining uncorrected faults better than other two models. As can be seen from Table 4, the Xie model also fits DS2 very well and it provides a very close SSE value to the proposed model. But Fig. 7 shows that the Xie model cannot characterize the correction lag as well as the proposed model. This fact means the Xie model cannot fit the DS2 as precisely as the proposed model does. Hence, the relationship between the fault detection and correction processes can be more accurately described by the proposed model from the viewpoint of fault amount.

SOFTWARE RELEASE PROBLEM

Although, testing is an efficient way to find and eliminate the faults in software systems, the exhaustive testing is impractical in fact. The software developers need to decide when to stop the testing and release the software to customers. As software reliability growth models can capture the quantitative aspects of the testing, they are used to provide a reasonable software release time. Some researchers have studied the release time problem from the cost-benefit viewpoint and they suggested that the optimal release time T* can be obtained by minimizing the expected total software system cost (Lyu, 1996; Pham and Zhang, 1999, 2003; Huang and Lyu, 2005). For instance, Pham and Zhang (1999) proposed a generalized software cost model including removal cost, warranty cost and risk cost, which is:

$$C(T) = c_0 + c_1 \cdot T^{\alpha} + c_2 m(T)\mu_y + c_3\mu_w[m(T+T_w) - m(T)] + c_4[1 - R(x/T)] \quad (13)$$

where, c_1 is the software test cost per unit time, c_2 is the cost of correcting a fault per unit time during testing period, c_3 is the cost of correcting a fault per unit time during the warranty period, c_4 is the loss due to software failure, C(T) is the expect total cost of software systems at time T, α is the discount rate of the testing cost, u_y is the expected time to correct a fault during the testing period, u_w is the expected time to correct a fault during the warranty period; T_w is the period of warranty time and R(x/T) is the reliability function of software by time T for a mission time x which can be calculate by the SRGMs and it is expressed as:

$$R(x/T) = e^{-[m(T+x)-m(T)]}$$

The conventional SRGMs, m(T), only describe the fault detection process, which is represented by $m_d(T)$ in this study. As the proposed SRGM models both fault detection and correction processes, the traditional cost function C(T), can be revised to be more practical by incorporating the fault correction process model $m_c(T)$:

$$C'(T) = c_0 + c_1 \cdot T^{\alpha} + c_2 m_c(T)\mu_y + c_3\mu_w[m_d(T+T_w) - m_c(T)] + c_4[1 - R(x/T)] \quad (14)$$

where, $m_c(T)$ is the expected number of corrected faults at the time of release T, $m_d(T+T_w)$-$m_c(T)$ is the expected number of corrected faults during the warranty period, which includes two components: undetected faults $m_d(T+T_w)$-$m_d(T)$ and remaining uncorrected faults $m_d(T+T_w)$-$m_c(T)$.

The latter component represents the correction lag between the fault detection and correction processes which has been specifically discussed in our study and it exists in most software testing. These detected but uncorrected faults remain in the software even after the software release for some potential reasons, such as the limited testing time or the testing resource, the detected fault is uncritical, etc. The correction of these detected faults will cost more efforts in the operation period. We can conclude that the revised software cost model C'(T), incorporates the fault detection and correction processes and reflect the correction lag phenomena, so it is very practical.

Pham and Zhang (1999) used DS2 to show how to determine the optimal release time by their generalized software cost model C(T) with the conventional SRGMs. They considered the coefficients as $c_0 = 50$, $c_1 = 700$, $c_2 = 60$, $c_3 = 3{,}600$, $c_4 = 50{,}000$, $T_w = 20$, $u_y = 0.1$, $u_w = 0.5$, $x = 0.05$, $\alpha = 0.95$. Substituting the proposed fault detection process model $m_d(t)$ to Eq. 13 with the estimated parameters on DS2, $\hat{\alpha} = 138.65$, $\hat{b} = 0.1351$, $\hat{c} = 5.44$, $\hat{\theta} = 8.65$, we get that T* is 30.1 h and the lowest cost is $22,760 by minimizing C(T). The corresponding cost function is plotted as a dash line in Fig. 8.

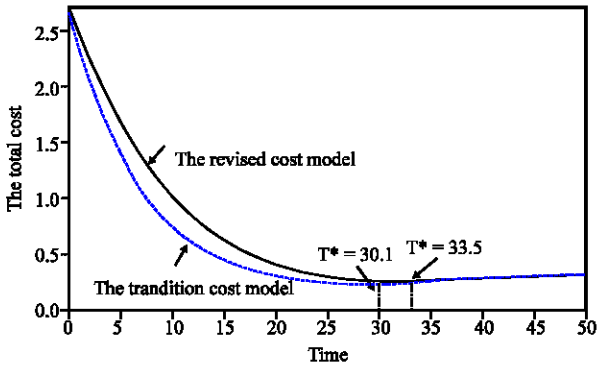

Fig. 8: Plot of the total cost

To show the difference in using the new software cost model, we also substitute the proposed model for both fault detection process and fault correction process, $m_d(t)$ and $m_c(t)$, into the revised cost function C'(T). With the same coefficients and estimated parameters value, the optimal release time can be obtained as T'* = 33.5 h with the lowest cost of $24,538. The revised cost function is also plotted as a solid line in Fig. 8. From the numbers and the figure, we can find out that a later optimal release time and a larger total cost are provided by the revised cost model. This result shows the effects of the correction lag. In reality, people need more time and recourse to perform the fault correction activity. As the traditional cost model does not consider the correction lag, it is optimistic in determining the software release time.

CONCLUSIONS

The fault correction process plays a significant role in ensuring the software quality and it always lags behind the fault detection process. In this study, the correction lag between the fault detection and correction processes was directly studied from a novel viewpoint of fault number. We investigated the number of remaining uncorrected faults on real data sets and found that it had an increasing and then decreasing feature. The Gamma curve was appropriate in characterizing the remaining uncorrected faults. Then we modeled the fault detection and correction processes based on the correction lag. Numerical examples showed that the proposed model successfully described the two processes and the comparison results also indicated that using the remaining uncorrected faults to integrate the two processes was more effective than using the time delay functions. Furthermore, we studied the software release problem. The proposed SRGM was applied to extend the software cost model and a more reasonable software

release time was achieved by considering the effect of the correction lag.

ACKNOWLEDGMENTS

The work described in this study is jointly supported by Hi-Tech Research and Development Program of China (2008AA01A201) and National Nature Science Foundation of China (60503015).

NOTATIONS

MVF	:	Mean value function
SSE	:	Sum of squared errors
MRE	:	Means of relative errors
NHPP	:	Non-homogeneous Poison process
SRGMs	:	Software reliability growth models
$m_d(t)$	:	The expected number of faults detected in time (0, t)
$m_c(t)$	:	The expected number of faults corrected in time (0, t)
$\varphi(t)$	:	The number of remaining uncorrected faults at time t
a	:	Expected number of initial faults
b	:	Fault detection rate
y_k	:	No. of actual faults observed at time t_k

REFERENCES

Gokhale, S., M.R. Lyu and K.S. Trivedi, 2004. Analysis of software fault removal policies using a non-homogeneous continuous time Markov chain. Software Qual. J., 12: 211-230.

Gokhale, S., M.R. Lyu and K.S. Trivedi, 2006. Incorporating fault debugging activities into software reliability models: A simulation approach. IEEE Trans. Reliab., 55: 281-292.

Huang, C. and M.R. Lyu, 2005. Optimal release time for software systems considering cost, testing-efforts test efficiency. IEEE Trans. Reliab., 54: 583-591.

Huang, C. and C. Lin, 2006. Software reliability analysis by considering fault dependency and debugging time lag. IEEE Trans. Reliab., 55: 436-450.

Huang, C.Y. and W.C. Huang, 2008. Software reliability analysis and measurement using finite and infinite queuing model. IEEE Trans. Reliab., 57: 192-203.

Kupar, P.K., D.N. Goswami, A. Badham and O. Singh, 2008. Flexible software reliability growth model with testing effort dependent learning process. Applied Math. Model., 32: 1298-1307.

Lo, J.H. and C.Y. Huang, 2006. An integration of fault detection and correction processes in software reliability analysis. J. Syst. Software, 79: 1312-1323.

Lo, J.H., 2007. Effect of the delay time in fixing a fault on software error models. Proceeding of the 31st Annual International Computer Software and Application Conference, Jul. 23-27, Beijing, China, pp: 711-716.

Lyu, M.R., 1996. Handbook of Software Reliability Engineering. McGraw-Hill and IEEE Computer Society, New York, ISBN: 0-07-039400-8.

Musa, J.D., A. Iannino and K. Okumoto, 1987. Software Reliability Measurement, Prediction and Application. McGraw-Hill, New York, ISBN: 0-07-044093-X.

Pham, H. and X. Zhang, 1999. A software cost model with warranty and risk costs. IEEE Trans. Comput., 48: 71-75.

Pham, H. and X. Zhang, 2003. NHPP software reliability and cost models with testing coverage. Eur. J. Opert. Res., 145: 443-454.

Schneidewind, N.F., 2001. Modeling the fault correction process. Proceedings of the 12th International Symposium on Software Reliability Engineering-ISSRE, Nov. 28-30, Hong Kong, China, pp: 185-190.

Schneidewind, N.F., 2003. Fault correction profiles. Proceedings of the 14th International Symposium on Software Reliability Engineering-ISSRE, Nov. 28-30, Denver, USA, pp: 257-267.

Williams, D.R.P., 2006. Prediction capability analysis of two and three parameters software reliability growth models. Inform. Technol. J., 5: 1048-1052.

Wu, Y.P., Q.P. Hu, M. Xie and S.H. Ng, 2007. Modeling and analysis of software fault detection and correction process by considering time dependency. IEEE Trans. Reliab., 56: 629-642.

Xie, M., Q.P. Hu, Y.P. Wu and S.H. Ng, 2007. A study of the modeling and analysis of software fault-detection and fault-correction processes. Qual. Reliab. Eng. Int., 23: 459-470.

Zhao, J., H.W. Liu, G. Cui and X.Z. Yang, 2006. Software reliability growth model with change-point and environment function. J. Sys. Software, 79: 1578-1587.