Software Reliability Modeling Considering Fault Correction Process

Lixin JIA†, Bo YANG††, Suchang GUO†††, Nonmembers, and Dong Ho PARK††††(a), Member

SUMMARY Many existing software reliability models (SRMs) are based on the assumption that fault correction activities take a negligible amount of time and resources, which is often invalid in real-life situations. Consequently, the estimated and predicted software reliability tends to be over-optimistic, which could in turn mislead management in related decision-makings. In this paper, we first make an in-depth analysis of real-life software testing process; then a Markovian SRM considering fault correction process is proposed. Parameter estimation method and software reliability prediction method are established. A numerical example is given which shows that by using the proposed model and methods, the results obtained tend to be more appropriate and realistic.

key words: software reliability, software testing, fault correction process, continuous-time Markov chain, maximum likelihood estimation

1. Introduction

During the past decades, many software reliability models (SRMs) have been proposed, which are useful to management in related decision-makings such as software release time determination, e.g., [1]. Nevertheless, many SRMs are based on some unrealistic assumptions. Among them, immediate repair of software faults, also called instantaneous fault correction, is a common one. By this assumption, when a software failure is observed during software testing, the corresponding software fault which has caused the failure can be corrected immediately, i.e., the fault correction time is negligible.

Effort has been made to relax this unrealistic assumption. Schneidewind [2] first clarified the fault detection process (FDP) and the fault correction process (FCP) in software testing. Levendel [3] used a birth-death process to model software testing. Gokhale et al. [4] examined various kinds of fault removal policies and their effect on the residual number of faults. Huang and Lin [5] considered both fault dependency and the time lag between the FDP and the FCP. Xie et al. [6] modeled the FCPs by delayed detection processes with a random or deterministic delay.

In most of related research, point processes such as non-homogeneous Poisson process or birth-death process are commonly used. In this paper, we propose a SRM considering non-negligible fault correction time under Markovian modeling framework.

2. Software Testing Process Considering Fault Correction Activities

2.1 A Practical Scenario of Software Testing Process

In a real-life software project, there may exist two independent groups involved in software testing, i.e., the testing group (TG) and the repair group (RG). The TG conducts software testing and once a failure is observed, a test record will be generated and passed to the RG for analysis. The TG then continues testing the software. It can be noted that at this time, no modification has been made on the software (which is the responsibility of the RG), thus the failure process observed in the resumed software testing will be statistically the same as before.

After receiving the test record, the RG conducts investigation into it and identifies the corresponding fault. Then the RG modifies the software to remove the fault. The updated software is then passed back to the TG for further testing. At this time, since the software has been updated, the testing process afterwards will be statistically different from the one conducted on the old software.

2.2 An Illustrative Example

To illustrate, we give an example here. Denote by \( t_i (i = 1, 2, \ldots, N_0) \) the time elapsed from the beginning of the testing to the observation of the \( i \)th failure, where \( N_0 \) is the number of latent faults in the software before testing. Denote by \( s_i (i = 1, 2, \ldots, N_0) \) the time elapsed from the beginning of the testing to the completion of the correction of the \( i \)th fault; and denote by \( T_i (i = 1, 2, \ldots, N_0) \) the time spent to correct the \( i \)th fault. Denote by \( I(t) \) the cumulative number of failures observed up to time \( t \); and denote by \( J(t) \) the cumulative number of corrected faults up to time \( t \). \( I(t) \) and \( J(t) \) satisfy the following condition.

\[
0 \leq J(t) \leq I(t) \leq N_0 \text{ for any } t \geq 0.
\]

Figure 1 illustrates realizations of the FDP and the FCP. The first failure is observed at time \( t_1 \), then the test record is
passed to the RG (illustrated by a down arrow) and the fault correction activity begins. Before time $t_1$ the RG is idle, illustrated by a dotted line. The fault correction activity takes $T_1$ units of time and the first fault is corrected at time $s_1$. At this time, the software is updated and passed back to the TG for further testing, and the RG is waiting for the arrival of the next test record.

The time spent to correct the first fault is

$$T_1 = s_1 - t_1$$

(2)

The 4th failure is observed at time $t_4$; however, at this time the RG is still occupied by correcting the 3rd fault, thus the correction activity for the 4th fault does not begin until $s_3$. The time spent to correct the 4th fault is

$$T_4 = s_4 - s_3$$

(3)

Define $t_0 \equiv s_0 \equiv 0$, from (2) and (3) it can be seen that the time spent to correct a fault is

$$T_i = s_i - t_i \text{ or } T_i = s_i - s_{i-1}, \quad i = 1, 2, \cdots, N_0.$$  

(4)

3. Software Reliability Modeling Considering Fault Correction Process

3.1 Modeling the Fault Detection Process

The following assumptions are made.

A1. The number of initial software faults before testing is an unknown but fixed constant, denoted by $N_0$.

A2. Software failure rate is constant, and each remaining fault contributes the same amount, denoted by $\phi$, to software failure rate.

A3. Software failure rate only changes when a software fault is corrected.

A4. Times between two consecutive software failure rate changes are statistically independent.

These assumptions are similar to those made by the J-M model [7], except for A3 due to the consideration of FCP.

Suppose the software has been tested for $t_0$ units of time, and there are $n$ faults detected and $m$ faults corrected, $m \leq n \leq N_0$. From A3, the number of remaining faults is $N_0 - m$. Thus from A2 the software failure rate is

$$\lambda(t) = \phi(N_0 - m)$$

(5)

Software reliability, $R(\tau|t_0)$, is

$$R(\tau|t_0) = \exp \left[ -\tau \phi(N_0 - m) \right].$$

(6)

Note that software reliability obtained by the J-M model is

$$R(\tau|t_0) = \exp \left[ -\tau \phi(N_0 - n) \right].$$

(7)

Since $m \leq n$, it seems that if FCP is not considered, the obtained software reliability tends to be over-optimistic.

3.2 Modeling the Fault Correction Process

Because fault repair policy is not the main issue to be addressed by this paper, we take the simplest fault repair policy, i.e., we assume a constant fault correction rate, denoted by $\mu$. We also assume that fault correction times are statistically independent. In this case, fault correction times, $T_i$’s, are exponential random variables (r.v.’s) with cumulative distribution function as follows.

$$F_C(t) = 1 - \exp(-\mu t), \quad t > 0.$$  

(8)

3.3 Estimation of Model Parameters

In (6) and (8), there are three model parameters, $\mu$, $N_0$, and $\phi$, that have to be estimated. Note that the immediate event before the $i$th failure occurrence can either be the occurrence of the $(i-1)$th failure at time $t_{i-1}$ (referred to thereafter as Case I) or the completion of the $k$th fault correction at time $s_k$, $k \leq i - 1$ (referred to thereafter as Case II). For example, in Fig. 1 the immediate event before the 2nd failure occurrence is the completion of the 1st fault correction (at time $s_1$); while the immediate event before the 4th failure occurrence is the occurrence of the 3rd failure (at time $t_3$). In general, if there is no even of completion of fault correction between $[t_{i-1}, t_i]$, then we have Case I; otherwise we have Case II.

Denote by $Y_i (1 \leq i \leq n)$ the length of the time interval between the $i$th failure occurrence and the occurrence of the immediate event before it, then we have

$$Y_i = \begin{cases} t_i - t_{i-1} & \text{Case I} \\ t_i - s_k & \text{Case II} \end{cases}$$

(9)

$Y_i$'s are independent and exponential r.v.'s, with $\lambda_i$ having the parameter of $\phi(N_0 - J_i)$, where $J_i \equiv J(t_i)$. Therefore, the log-likelihood function is

$$\ln L(N_0, \phi) = n \ln \phi + \sum_{i=1}^{n} [\ln(N_0 - J_i) - \phi(N_0 - J_i)Y_i] \quad (10)$$

Thus we have

$$\phi = \frac{n}{\sum_{i=1}^{n} (N_0 - J_i)Y_i}, \quad \sum_{i=1}^{n} \left( \frac{1}{N_0 - J_i} \right) = \frac{n}{N_0 - p}.$$  

(11)

where $p \equiv \sum_{i=1}^{n} J_iY_i \sum_{i=1}^{n} Y_i$. $\hat{N}_0$ and $\hat{\phi}$ can be obtained by solving (11). $\hat{N}_0$ should be rounded to the nearest integer value, and it is often required that $\hat{N}_0 \geq n$. 

Fig. 1 A real-life software testing process.
Theorem

(1) If \( p \leq \frac{1}{n} \sum_{i=1}^{n} J_i \), then \( \mathcal{N}_0 \) and \( \hat{\phi} \) do not exist.

(2) If \( \frac{1}{n} \sum_{i=1}^{n} J_i < p < n \left[ 1 - \left( \sum_{i=1}^{n} \frac{1}{n - J_i} \right)^{-1} \right] \), then \( \mathcal{N}_0 \) and \( \hat{\phi} \) exist, which is the unique solution to (11).

(3) If \( p \geq n \left[ 1 - \left( \sum_{i=1}^{n} \frac{1}{n - J_i} \right)^{-1} \right] \), then \( \mathcal{N}_0 \) and \( \hat{\phi} \) are

\[
\mathcal{N}_0 = n, \quad \hat{\phi} = n \left( \mathcal{N}_0 - J_i \right) Y_i.
\]

The proof is similar to that in [8], thus it is omitted here.

For estimation of \( \mu \), note that \( T_i \)'s are independent and identically distributed (i.i.d.) exponential r.v.'s, thus

\[
\hat{\mu} = \frac{m}{\sum_{i=1}^{m} T_i}.
\]  

(12)

3.4 Software Reliability Prediction

This problem is defined as follows: If the software is tested for additional \( T \) units of time, then what will the software reliability, \( R(\tau|T_0 + T) \), be?

Define software state at time \( t \) as \( [I(t), J(t)] \). Software state changes over time, which can be modeled by a CTMC. Figure 2 shows software state transition from the initial state \([n, m]\) at time \( t_0 \). In Fig. 2, \( \lambda_i \) is defined as

\[
\lambda_i = \phi \left[ N_0 - (m + i - 1) \right],
\]

\[
1 \leq i \leq (N_0 - m), \quad m \leq j \leq N_0.
\]  

(13)

Define \( P_{i,j}(t) \equiv \Pr\{J(t) = i, J(t) = j, t > t_0\} \), then the Kolmogorov's differential equations are

\[
\frac{dP_{i,j}(t)}{dt} = -(\lambda_j + \mu)P_{i,j}(t) + \lambda_i P_{i,j+1}(t) + \mu P_{i,j-1}(t),
\]

\[
(n \leq i \leq N_0, m \leq j \leq i, t > t_0).
\]  

(14)

Using the boundary condition

\[
P_{i,j}(t_0) = \begin{cases} 1, & \text{for } i = n, \ j = m \ \text{and } \ N_0, \ \text{otherwise} \end{cases},
\]

(15)

and the ML estimates obtained, the differential equations (14) can be solved and \( P_{i,j}(t_0 + T) \) can be obtained.

To predict software reliability \( R(\tau|T_0 + T) \), using the full probability formula and we get

\[
R(\tau|T_0 + T) = \sum_{i=n}^{N_0} \sum_{j=m}^{\min(i,J_0)} \Pr\{X > \tau|T_0 + T = i, J(T_0 + T) = j\} \cdot P_{i,j}(T_0 + T),
\]

(16)

where \( X \) is time of failure-free operation of the software, starting from \( T_0 + T \). Similar as (6), we have

\[
\Pr\{X > \tau|T_0 + T = i, J(T_0 + T) = j\} = \exp\left[-\tau\phi(N_0 - j)\right].
\]  

(17)

Substitute (17) into (16), and we have

\[
R(\tau|T_0 + T) = \sum_{i=n}^{N_0} \sum_{j=m}^{\min(i,J_0)} \exp\left[-\tau\phi(N_0 - j)\right] \cdot P_{i,j}(T_0 + T).
\]  

(18)

From the boundary condition (15), it can be easily seen that when \( T = 0 \), the current software reliability given by (18) reduces to that given by (6).

4. A Numerical Example

The following parameters are used in the example.

\[
N_0 = 33, \ \phi = 0.1120, \ \mu = 0.2473.
\]

The FDP and FCP are generated by simulation. The FDP is terminated at \( T_0 = 255.1 \). By then 29 software failures have been observed, i.e., \( n = 29 \); and 28 software faults have been corrected, i.e., \( m = 28 \). Failure occurrence times, \( t_1, t_2, \cdots, t_{29} \), and fault correction completion times, \( s_1, s_2, \cdots, s_{28} \), are recorded.

Using the estimation methods in Sect. 3.3, we have

\[
\hat{\mathcal{N}}_0 = 32, \ \hat{\phi} = 0.1321, \ \hat{\mu} = 0.2651.
\]

From (6) we know that current software reliability is

\[
R(\tau|255.1) = \exp(-0.5895\tau).
\]

If FCP is not considered, e.g., using the J-M model, then the software reliability is

\[
R(\tau|255.1) = \exp(-0.6728\tau),
\]

which seems to be over-optimistic. The future software reliability, \( R(\tau|255.1 + T) \), for any given value of \( T > 0 \), is shown in Fig. 3 (a), assuming \( \tau = 3 \). Figure 3 (b) shows that predicted by the J-M model based on the same data set generated by simulation. Again, if FCP is not considered, then the predicted software reliability seems to be over-optimistic.
5. Conclusions

In this paper, we propose a Markovian SRM considering non-negligible fault correction time. Parameter estimation method and software reliability prediction method are also developed. It seems that the proposed model can describe the real-life software testing process more appropriately, and the results obtained appear to be more realistic.

In our future research, we will conduct comparative studies on the effectiveness and efficiency of the proposed SRM and existing SRMs considering FCP. We will also extend the proposed model by considering other fault repair policies.

Acknowledgments

This work is supported by National Natural Science Foundation of China (No. 50805018) and Key Project of Chinese Ministry of Education (No. 109138). Dong Ho Park’s work was in part supported by National Research Foundation of Korea Grant funded by the Korean Government (2009-0072742).

References