Software Reliability Modeling with Imperfect Debugging and Change of Test Environment

Shinji Inoue1, Shigeru Yamada2
1Faculty of Informatics, Kansai University
2-1-1, Ryozenji, Takatsuki, Osaka 569-1095, Japan
ino@kansai-u.ac.jp
2Graduate School of Engineering, Tottori University
4-101, Koyama-Minami, Tottori, Tottori 680-8552, Japan
yamada@sse.tottori-u.ac.jp

Abstract — We discuss Markovian software reliability modeling with imperfect debugging environment and the effect of change-point for considering more practical situation of a software reliability growth process. Further, we discuss a parameter estimation method for applying our model to observed data. Finally, we show numerical examples of our model, and check the performance of our model with an existing Markovian imperfect debugging model by using actual software failure-occurrence time data.

Keywords — Software reliability, Software reliability modeling, Imperfect debugging environment, Change-point, Semi-Markov process.

I. INTRODUCTION

It is known that testing manager often observes that the characteristic of the software failure-occurrence is changed notably due to some factors which are related to the software reliability growth process. And such change influences on the accuracy of software reliability assessment based on software reliability growth models (abbreviated as SRGMs) [7,9]. Testing-time when such change is observed is ordinarily called a change-point [10]. Taking the effect of the change-point on the software reliability growth process into consideration in software reliability growth modeling is one of the effective approach for conducting more accurate software reliability assessment based on the SRGM. From the background mentioned above, software reliability growth models with the effect of change-point have been proposed so far [1-4]. Further, we need to consider the effect of imperfect debugging environment, which reflects actual situation, for describing software reliability growth process more accurately.

We describe the software failure-occurrence phenomenon with imperfect debugging activities by using semi-Markov process. Furthermore, the relationship between the software failure-occurrence time intervals before change-point and those after change-point is also considered by using test environment coefficient. Furthermore, we slightly discuss a method of applying our model to actual data. Finally, we show a numerical example of our model, and check the performance of our model with an existing Markovian imperfect debugging model by using actual software failure-occurrence time data.

II. MARKOVIAN IMPERFECT DEBUGGING MODEL

We describe the software failure-occurrence phenomenon under the imperfect debugging environment by using a Markovian imperfect model [8], which is based on the following assumptions:

(A1) Debugging activities are immediately conducted after the software failure-occurrence. Each fault which causes a software failure is corrected perfectly with probability \( a \), and is not perfectly corrected with probability \( b = 1 - a \). No new faults are introduced during the debugging activities.

(A2) The software failure-occurrence time interval, denoted by \( U_n \), follows the exponential distribution with mean \( 1/\lambda_n \) when \( n \) faults have been perfectly corrected. \( \lambda_n \) is a non-increasing function on \( n \).

The fault-correction time is not considered in this model.

Let \( \{X(t), t \geq 0\} \) be the counting process representing the cumulative number of faults perfectly corrected up to time \( t \). We can see that the counting process \( X(t) \) is governed by the perfect debugging rate \( a \) from the above assumptions. The one-step transition probabilities are given by

\[ Q_{i,i}(t) = b(1 - e^{-\lambda_i t}), \quad i = 1, 2, \ldots \]

\[ Q_{i,i+1}(t) = a(1 - e^{-\lambda_i t}), \quad i = 1, 2, \ldots \]

respectively.

Let us denote \( G_{i,n}(t) \equiv \Pr\{S_{i,n} \leq t\} \) the first passage time distribution from state (the number of perfectly corrected faults) \( i \) to state \( n \). We obtain the following renewal function:

\[ G_{i,n}(t) = Q_{i,i+1} * G_{i+1,n}(t) + Q_i * G_{i,n}(t), \quad i = 1, 2, \ldots, n. \]
where * denotes a Stieltjes convolution and \( G_{n,n}(t) \equiv 1(n = 1, 2, \cdots) \). We can obtain the solution by solving recursively forward the above renewal equation.

We are able to focus on the number of software failure occurrences. Let \( M_i(t) \) be the number of software failure occurred during time-interval \((0, t]\) on the condition that \( i \) faults have been already perfectly corrected at \( t=0 \). Supposing that the initial fault content \( N \) is known, \( M_i(t) \) is obtained as

\[
M_i(t) = \frac{1}{\alpha} \sum_{n=1}^{\infty} G_{i,n}(t), \tag{4}
\]

by recursively solving the following renewal function:

\[
M_i(t) = F_i(t) + Q_i, t \ast M_i(t) + Q_{i+1} \ast M_{i+1}(t), \tag{5}
\]

under the initial condition \( M_N(t) = 0 \). However, it is very hard to obtain the information how many faults has been perfectly corrected. Then, we incorporate the measure, \( M(t; l) \), representing the expected number of software failures occurred time-interval \((0, t]\) on the condition that the \( l \)-th debugging activity has just been completed at \( t=0 \), and is formulated as

\[
M(t; l) = \sum_{i=0}^{l} \binom{l}{i} a^i b^{l-i} M_i(t). \tag{6}
\]

Further, letting \( X_l (l = 1, 2, \cdots) \) be random variables representing the time-interval between the \((l-1)\)-th and \( l \)-th software failure-occurrence, a software reliability function \( R_l(x) \) is derived as

\[
R_l(x) = \sum_{i=0}^{l-1} \binom{l-1}{i} a^{i+1} b^{l-i-1} e^{-\lambda_i x}, \tag{7}
\]

by using the notion of Eq. (6).

### III. MARKOVIAN CHANGE-POINT MODELING

Let us denote by \( U^C \) the software failure-occurrence time-interval after change-point under \( i \) faults have been completely corrected. The relationship between \( U^C \) and \( F^C(t) = F_i(t, \alpha^{-1}(t)) \), where \( \alpha(t) = \alpha t (t \geq 0, \alpha > 0) \) and \( \alpha \) is an environmental coefficient representing the difference of test environment [4,6]. Then, the first passage time distribution after change-point, \( G^C_{i,n}(t) \), is given by

\[
G^C_{i,n}(t) = G_{i,n} \left( \frac{t}{\alpha} \right). \tag{9}
\]

Further, the expected number of software failures occurred during time-interval \((0, t]\) after change-point under the \( l = l' \)-th debugging activity has just been completed at \( t=0 \), \( M^C(t; l) \), is derived as

\[
M^C(t; l) = \sum_{i=0}^{l'} \binom{l'}{i} a^i b^{l'-i} M_i \left( \frac{t}{\alpha} \right) = M \left( \frac{t}{\alpha}; l \right). \tag{10}
\]

where \( l = l', l' + 1, l' + 2, \cdots \). And we also derive the software reliability function after change-point, \( R^C_l(x) \) as

\[
R^C_l(x) = R_l \left( \frac{x}{\alpha} \right)
\]

\[
l = l', l' + 1, l' + 2, \cdots, \tag{11}
\]

from Eq. (7).

### IV. PARAMETER ESTIMATION

The parameters in our model can be estimated by using the method of maximum likelihood. First of all, we need to derive the probability density function of \( X_l \), which depends on the number of perfectly corrected faults.

Due to the reason mentioned before, we approximately give the density by considering \( \Pr\{A_l = \lambda_l \} \equiv \Pr\{G_l = i\} \), where \( \lambda_l \) is a random variable for the hazard rate of \( X_l \) and \( G_l \) is a random variable representing the cumulative number of perfectly corrected faults at the completion time of the \( l \)-th debugging activity. Concretely, the hazard rate of \( X_l \) denoted by \( z_l \) is given approximately as

\[
z_l \approx E[A_l] = \sum_{i=0}^{l-1} \lambda_i \binom{l-1}{i} a^i b^{l-i-1}. \tag{12}
\]

In Eq. (12), it is implied that the \( z_l \) follows an exponential distribution because \( z_l \) does not depend on the elapsed time \( x \).

Then, the probability density of \( X_l \) before and after change-point are given by

\[
\phi_l(x) = z_l \exp[-z_l x], \tag{13}
\]

\[
\phi^C_l(x) = \frac{z_l}{\alpha} \exp\left[-\frac{z_l}{\alpha} x \right], \tag{14}
\]

respectively. From Eq. (13), the parameters in \( z_l \) can be estimated by using the method of maximum likelihood based on the software failure-occurrence time data observed before change-point. Further, the parameter \( \alpha \) is estimated by the method of maximum likelihood by using the software failure-occurrence time data observed after change-point. Consequently, the maximum likelihood parameter estimation of \( \alpha \) is obtained as

\[
\hat{\alpha} = \frac{\sum_{l=1}^{L} \sum_{i=1}^{m} z_i x_l}{m}, \tag{15}
\]

where we assume that we have observed \( m \) software failure-occurrences after the change-point.
V. PARAMETER ESTIMATION

We show numerical examples of our model by using actual data. We use the following software failure-occurrence time data: \((k, t_k)(k = 1, 2, \ldots, 31; t_{31} = 540 \text{ (days)})\) [7]. Change-point has been occurred at \(t_{27}\) due to changing the fault target. Therefore, we regard \((k, t_k)(k = 1, 2, \ldots, 26)\) as the data collected before change-point and \((k, t_k)(k = 27, 28, \ldots, 31)\) as the data collected after change-point, respectively. Regarding the hazard rate \(\lambda_n\), we apply Moranda model [5]:

\[
\lambda_n = D c^n (n = 0, 1, 2, \ldots; D > 0, 0 < c < 1), \quad (16)
\]

where \(D\) is the initial hazard rate for the first software failure-occurrence and \(c\) is the decreasing rate for the hazard rate. Therefore,

\[
x_t \approx E[\lambda_1] = D(ac + b)^{t-1}, \quad (17)
\]

from Eq. (12). Then, we estimate the parameters by following the parameter estimation procedure discussed before under the condition \(a=0.9\) for example. Fig. 1 depicts the estimated time-dependent behavior of the expected number of software failures occurred after the change-point, \(\hat{M}(t; 26)\) in Eq. (6) and \(\hat{M}^c(t; 26)\) in Eq. (10), respectively, with the actual behavior. In Fig. 1, we can say that our model describes better the behavior after the change-point than the model not incorporating the effect of change-point. This means that our model contributes improving the accuracy of software reliability assessment. It is worth mentioning that some software reliability assessment measures, e.g., software reliability function in Eq. (11), can be estimated by following our model assumptions. Fig. 2 shows the estimated the time-dependent behavior of the software reliabilities, \(\hat{R}_{27}^C(x)\) and \(\hat{R}_{27}(x)\), which are the estimated software reliabilities after the 27th software failure-occurrence. \(\hat{R}_{27}^C(x)\) is obtained by
from Eqs. (7) and (11). From Fig. 2, we can see that our model, which considers the effect of change-point, estimates the software reliability more optimistically than the existing model, which does not incorporate the effect of change-point. However, we can say that the estimated software reliability estimated by our model might be reflect the observed software failure-occurrence phenomenon due to the results in Fig. 1.

VI. CONCLUSIONS

We proposed a software reliability model considering with the imperfect debugging activities and the effect of change-point observed during the testing phase by applying a Markov process. Concretely, we described the behavior of the number of perfectly corrected faults with the constant rate of the imperfect debugging. Then, we derived the first passage time distribution. Further, we obtained some software reliability assessment measures with the effect of change-point from the first passage time distribution. Additionally, we discussed a method for obtaining parameters of our model from software failure-occurrence time data based on the method of maximum-likelihood. Finally, we checked that our Markovian change-point model depicts well the observed time-dependent behavior of the expected number of software failures occurred after the change-point. We still need to check the performance of our model by using other software reliability assessment measures and a lot of software failure-occurrence time data. Furthermore, we also need to discuss how to set the value of perfect debugging rate in the actual software testing activities.

ACKNOWLEDGMENT

This research was supported in part by the Grant-in-Aid for Scientific Research (C), Grant No. 16K00098, from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

REFERENCES