Analysis of the Debugging Model Based on Probabilistic State Transition

Jinyong Wang
Department of Computer Science and Technology
Harbin Institute of Technology
Harbin, China
wangjinyong818@163.com

Zhibo Wu and Yanjun Shu
Department of Computer Science and Technology
Harbin Institute of Technology
Harbin, China
{wzb, syj}@ftcl.hit.edu.cn

***Abstract*—In general, software reliability test is an important task in software developing process. Meanwhile, it called perfect debugging which assumed that detecting a fault and fixing it, no introducing new faults, in turn, introducing new faults, called imperfect debugging. We all hope the testing process is perfect debugging and no new faults will be introduced. Therefore, it can shorten the testing time and decrease the testing costs. However, in practice, the software test is a complex, intractable process and can be divided into two categories, which are perfect debugging and imperfect debugging. Furthermore, most of testing processes are imperfect debugging. Then, what is relation between perfect debugging and imperfect debugging and what is impact to imperfect debugging transiting to perfect debugging? In this paper, we propose two state models based on probabilistic state transition to analyze the relation between perfect debugging and imperfect debugging and express debugging time and costs during imperfect debugging transiting to perfect debugging process in terms of the parameters in that model. Threshold conditions for perfect debugging process to be beneficial are also derived. Finally, using the numerical examples prove our assumptions and inference.**

***Index Terms*—Perfect debugging, imperfect debugging, probabilistic state transition, perfect debugging; debugging time of test process, cost of test process.**

I. INTRODUCTION

Software test is a key part in software developing process. It is not only detecting the fault existed in software, but also improving the quality of software including availability and reliability. However, in reality, following the software increasing functions and size, code being more and more significantly and larger, software test becomes more and more difficult. In practical test, project managers always want an assessment of software developing expenditure regarding debugging time and testing costs. Thus, completely understanding the debugging process in software test becomes especially important.

In general, debugging in the test can be divided into two categories, one is perfect debugging which assumes that faults detected in the testing phase are removed immediately with no new faults introduction, the other is imperfect debugging which assumes that faults are removed with debugging time delay and introducing new faults. Gray [1] calls former bugs in the software test Bohr bugs, that is deterministic character, and later bugs Heisen bugs for faults that unpredictably lead to failure.

Considering debugging types discussed above, researcher proposed many related Software Reliability Growth Models (SRGMs), such as perfect debugging SRGM [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] and imperfect debugging SRGM [13], [14], [15], [16], [17], [18], [19] as well as the unified theory on imperfect or perfect debugging SRGMs [20], [21]. The traditional perfect SRGM [2], [3] does not consider the time to remove a fault and assumes that the faults are removed immediately after detecting a software fault in software testing process. However, in practice, software test is complex process including reporting, diagnosing, correcting, and verifying each fault observed. The time from detected fault to removal is a important parameter variable should not be neglected in a software testing process. Yamada et al. [11] and Schniedewind [7] considered the factor of removal fault time and proposed a two-stage process in which all observed faults were removed after a constant delay of time. Schneidewind [8] assumed that the time delay was a random variable with an exponential distribution. Following a similar idea, people made some attempts to model these two processes, the Schneidewind [7] model was extended by Xie and Zhao [10]. They used a time dependent delay function to substitute the constant delay. Gokhale et al. [4] modeled the two phases with a non-homogeneous Markov chain. Later, Stutzke and Smidts [12] combined fault-detection and fault-correction model to produce more practical models for the software testing process. Similarly, a general framework was proposed by Lo and Huang [6] for modeling failure observation and fault removal

doi:10.4304/jsw.8.11.2697-2705

processes in software test, and Xie et al. [9] described a delayed detection process with a random or deterministic delay, and emphasized the fault correction process where some existing NHPP models were re-evaluated from the viewpoint of a correction process.

On the other hand, the debugging activity is not always perfect because of a number of factors like the tester's skill, expertise and environment. When observating a fault, the testing team may not be able to remove a fault perfectly, and the original fault may remain, leading to a phenomenon known as imperfect debugging, or replaced by another fault resulting in error generation. It was Goel [13] who first introduced the concept of imperfect debugging. Later, Kapur et al. [15] and Obha et al. [16] proposed imperfect model applied on the G-O model [2]. Although these two models described the imperfect debugging phenomenon, the software reliability growth curve of these models was still always exponential. Furthermore, they assumed that the probability of imperfect debugging was s-independent of testing time. Kapur et al. [14], Zhang et al. [17] and Pham et al. [18], [19] proposed a relative imperfect software debugging model using an S-shaped fault detection rate, error generation and a logistic function for the fault detection rate, respectively. To conveniently research, two unified theories on perfect debugging or imperfect debugging was proposed by Kapur et al. [20] and Huang et al. [21], respectively.

Although a lot of perfect and imperfect debugging models were proposed above have improved the software reliability assessment quality, the relation between them is not fully studied. In this paper, we simplify the software test environment, and assume that is perfect debugging or imperfect debugging. In fact, testing process is a complex process, either including perfect debugging or imperfect debugging. In other words, when we find a fault, fault may be removed immediately with no introducing new fault or be removed with introducing new faults. What is impact for the imperfect debugging transiting to perfect debugging? How much time and costs are increased when introducing new faults in the practice software testing process? What is relation between perfect and imperfect debugging? How to balance them? In this paper, we will solve these problems.

The purposes of this paper are:

- To analyze the relation between perfect debugging and imperfect debugging in practice software testing process.
- How to estimate the times and costs of introducing new faults versus no introducing new faults.

The main contributions of this paper are:

- Relation: Through analyzing the relation between perfect debugging and imperfect debugging, it illustrates perfect debugging is an important influence on imperfect debugging in software testing process.
- Threshold: It indicates it is important for the effective software test by determining the threshold including debugging time and cost due to perfect debugging.
- Benefit: They can effectively decrease time and cost in the practice software test through preventing the imperfect debugging from the key component or module in software.

Considering the complexity of the fault debugging process, we model the fault test with probabilistic state transition [22, 23, 24, 25, 26] in this paper. In practice, fault test is subjected to many fields, such as test skill, test tool and test environment et, al.. We simplify the fault testing process and mainly consider two factors, one is the perfect debugging with no introducing new faults, and the other is the imperfect debugging with introducing new faults. In fact, the interaction between perfect debugging and imperfect debugging exists in practical software reliability testing process. Thus, the models based on probabilistic state transition proposed in this paper have some practical significance.

The rest of this paper is organized as follows. Section II describes the detailed analysis of testing model based on the probabilistic state transition in terms of time and cost. Section III shows numerical examples for proving the proposed assumption and inference. Finally, conclusions are drawn in Section IV.

II. MODEL AND ANALYSIS

In the section, we are interested in determining how perfect debugging can decrease a testing time and the cost due to no introducing new faults. Intuitively, testing process which never introduces new faults should not be non-deterministic event. On the other hand, if testing process always introduces new faults on third day after it starts and imperfect debugging cost is very high, we should pay attention to the key component or module in code every two days so that no introducing new faults will occur. In general, the decision about imperfect and perfect debugging depends on the debugging cost, the delay time cost and whether introducing new faults or not. We propose a model based on probabilistic state transition and analyze it for imperfect and perfect debugging in terms of those parameters and derive the threshold conditions. In the following, we use A to denote a testing process with imperfect debugging, that is only imperfect debugging, and B to denote a testing process with imperfect and perfect debugging. We also use Debugtime denotes delay time in perfect and imperfect debugging, and use Cost denotes cost due to debugging process. The less Debugtime is, the better efficiency is. Certainly the less Cost is, the better benefit is.

A. Debugtime and Cost due to Test

The efficiency of software test is of course increased due to imperfect debugging. Thus, the cost of debugging time delay due to introducing new faults must be taken into account to make a decision on imperfect debugging, since imperfect debugging involves non-determination to deal with faults. The cost of the imperfect debugging is

TABLE I.
ILLUSTRATION OF NOTATION IN FIGURE 1 AND FIGURE 2.

type	Description
S_0	Test starting state or program execution state
S_F	Detected fault state
S_P	Perfect debugging state
S_I	Imperfect debugging state
r_1	A transition rate of imperfect debugging from S_P to S_0 state.
r_2	A transition rate from S_0 to S_F state.
r_3	A transition rate of perfect debugging from S_I to S_0 state.
α	A transition rate of starting imperfect debugging from S_F to S_P state.
β	A transition rate of starting perfect debugging from S_F to S_I state.

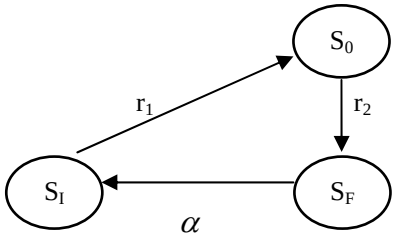

Figure 1. Probabilistic state transition model for A with imperfect debugging.

expected to be much higher than that of a determination dealing with faults in software test process. To compute removing fault costs, we first look at the probabilistic state transition diagram of an application with imperfect debugging as shown in Figure 1. Notation in Figure 1 and Figure 2. refers to Table I.

When it starts, the starting software test stays in a highly robust state S_0, then it goes into failure probable state S_F, and then to state S_I. Thus, fault debugging is a 3-step behavior as shown in Figure 1. where a testing A goes from state S_0 to state S_F at a transition rate of r_2 and from state S_F to state S_I at transition rate of α as well as from state S_I to state S_0 at a transition rate of r_1. Observe that $\frac{1}{r_2}$ corresponds to the base longevity interval for the testing program execution. We assume that the removing fault delay time for the testing process is also exponential distributed with a rate r_1. Under this assumption and deriving the equations

$$P_0+P_F+P_I=1 \quad (1)$$
$$P_F*\alpha = P_0*r_2 \quad (2)$$
$$P_I*r_1 = P_F*\alpha \quad (3)$$

Where P_0, P_F, P_I denote the steady-state probability in state S_0, S_F, S_I, respectively. We derive P_I to be equal to $\frac{1}{1+\frac{r_1}{\alpha}+\frac{r_1}{r_2}}$. Thus, the expected total Debugtime of A in an interval of L time units is:

$$\text{Debugtime}_A(L) = \frac{1}{1+\frac{r_1}{\alpha}+\frac{r_1}{r_2}} \times L \quad (4)$$

When a fault is detected, time and manual powers are necessary expensed to remove software faults. Therefore, there is a manual cost due to detecting and removing a fault during testing process. However, cost varies from different fault types. If C_I is the average cost per unit of A's imperfect debugging, that is introducing new faults, then expected cost of imperfect debugging in an interval of L time units is:

$$\text{Cost}_A(L) = \frac{1}{1+\frac{r_1}{\alpha}+\frac{r_1}{r_2}} \times L \times C_I \quad (5)$$

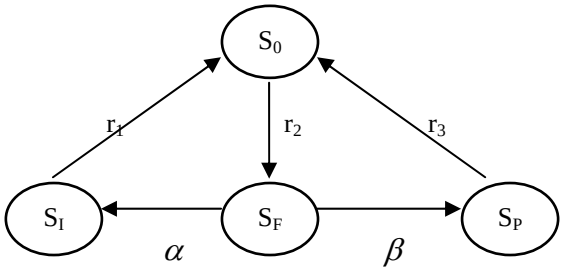

Figure 2. Probabilistic state transition model for B with perfect debugging.

Now consider the probabilistic state transition diagram of testing B with perfect debugging as shown in Figure 2. where state S_P is the perfect debugging state and the other states are as before. We assume that the starting perfect debugging rate β and the perfect debugging rate r_3 are also exponentially distributed. If the test is started perfect debugging after every t units, then β will be equal to $\frac{1}{t}$. We can derive the probability equalities produced from this model for B with perfect debugging, and then get the following expression for state probabilities:

$$P_F = \frac{1}{1+\frac{\alpha}{r_1}+\frac{\beta}{r_3}+\frac{\alpha}{r_2}+\frac{\beta}{r_2}} \quad (6)$$

$$P_I = \frac{\alpha}{r_1} * P_F \quad (7)$$

$$P_P = \frac{\beta}{r_3} * P_F \quad (8)$$

$$P_0 = \frac{\alpha+\beta}{r_2} * P_F \quad (9)$$

The expected total Debugtime of the testing B with perfect debugging in an interval of L time unites is:

$$\text{Debugtime}_B(L) = (P_I + P_P) \times L \quad (10)$$

If C_I is the average per unit cost of imperfect debugging as before and C_P is the average per unit cost of perfect debugging, then the total expected debugging cost in a interval L time units is:

$$\begin{aligned} Cost_B(L) &= (P_I \times C_I + P_P \times C_P) \times L \\ &= \frac{L}{1+\frac{\alpha}{r_1}+\frac{\beta}{r_3}+\frac{\alpha+\beta}{r_2}} \times (\frac{\alpha}{r_1} \times C_I + \frac{\beta}{r_3} \times C_P) \end{aligned} \quad (11)$$

We can verify that if perfect debugging is not performed then $\beta = 0$, $Debugtime_B(L) = Debugtime_A(L)$ in Equations 4 and 10, as well as $Cost_B(L) = Cost_A(L)$ in Equation 5 and 11. If fault debugged is performed during the most time of the imperfect debugging, then $r_3 > r_1$ and $C_I >> C_P$ and hence the total expected fault debugging cost in the testing process with imperfect debugging, as computed in Equation 5, would be higher than with perfect debugging as computed in Equation 11. Such that thresholds for imperfect debugging are discussed in the next section.

B. Perfect Debugging Thresholds

Intuitively, if r_3 is very large, i.e. Fast debugging from perfect debugging state, perfect debugging should reduce the total debugging time of testing phase since the testing phase can stay in state S_0 more often and the chance of fault occurred becomes smaller. On the other hand, if r_3 is very small, i.e. slow debugging from perfect debugging state, perfect debugging of software testing process increases the debugging time that the test stays in state S_P, and hence the total test time increases. Similarly, if the starting imperfect debugging rate α is very large, imperfect debugging could be performed frequently so that the test stays in state S_I longer, and hence, the chance of removing a fault becomes longer. However, if the faults in the software test is very rarely, perfect debugging occurred will increase the total debugging time. Similar threshold effects also exist for the base longevity interval ($\frac{1}{r_2}$) and the fixing rate after removing fault r_1. Such thresholds will be study below.

Let us analyze how the time and cost of debugging change when the starting imperfect debugging rate β changes. Substituting the values for P_I and P_P, Equation 10 can be written as following.

$$Debugtime_B(L) = \frac{\frac{\alpha}{r_1}+\frac{\beta}{r_3}}{1+\frac{\alpha}{r_1}+\frac{\alpha}{r_2}+\frac{\beta}{r_2}+\frac{\beta}{r_3}} \times L \quad (12)$$

In order to examine the behavior of debugging time when β changes, we can differentiate the above equation 12 with respect to β. Then, we can observe that the numerator and the denominator in Equation 12 are linear functions of β. Thus, the differentiation of the Debugtime function yields:

$$\begin{aligned} \frac{d}{d_\beta} Debugtime_B(L) = L \times \frac{\alpha}{r_1 r_2 r_3} \times \frac{1}{(1+\frac{\alpha}{r_1}+\frac{r_1}{r_2}+\frac{\beta}{r_2}+\frac{\beta}{r_3})^2} \\ \times [r_1(1+\frac{r_2}{\alpha}) - r_3] \end{aligned} \quad (13)$$

It is interesting to note that the sign of the numerator is determined by the expression $[r_1(1+\frac{r_2}{\alpha})-r_3]$ which is independent of β, and that the denominator in the above derivative is always positive. This means that, when β changes, the Debugtime increases or decreases depending entirely on the values of α, r_1, r_2 and r_3. When r_3 is less than $r_1(1+\frac{r_2}{\alpha})$, that is $r_3 < r_1$ because of $1+\frac{r_2}{\alpha} > 1$, the derivative is positive implying that the Debugtime always increases when the value of β increases. i.e. slow removing a fault from perfect debugging state. Similarly when r_3 is greater than $r_1(1+\frac{r_2}{\alpha})$, the derivative is negative and the Debugtime always decreases when the value of β increases. This implies that we should do perfect debugging as soon as the testing phase goes into state S_F. This conforms to our intuition that if the cost of perfect debugging is small and existing fault rate is large, we should do perfect debugging as soon as the testing phase into finding a fault state. For example, program could potentially fail when its memory size is over 10M bytes, the above condition implies that we should do perfect debugging without introducing new faults as soon as we find memory size reaches beyond 10M bytes and memory leaking phenomenon.

Now let us examine Equation 11 to determine the behavior of the expected debugging cost when the starting perfect debugging rate β changes. In practice, C_P is independent of β only for a small range of value of β. If β is large, increasing β will also increasing C_P because the perfect debugging will occur during a busy testing period. For simplicity, we assume C_P is independent of β and differentiate the cost B function in Equation 11 with respect to β. Thus, the differentiation of the Cost function yields:

$$\begin{aligned} \frac{d}{d_\beta} Cost_B(L) = L \times \frac{1}{r_1 r_2 r_3} \times \frac{1}{(r_1 r_2 + \alpha r_1 + \alpha r_2)} \\ \times \frac{1}{(1+\frac{\alpha}{r_1}+\frac{\alpha}{r_2}+\frac{\beta}{r_2}+\frac{\beta}{r_3})^2} \\ \times [C_P - C_I \frac{\alpha(r_2+r_3)}{\alpha(r_1+r_2)+r_1 r_2}] \end{aligned} \quad (14)$$

Again, the sign of the numerator is determined by the expression $C_P - C_I \frac{\alpha(a_2+a_3)}{\alpha(a_1+a_2)+a_1a_2}$ which is independent of β, and the denominator in the above derivative is always positive This means that, when β changes the total expected debugging cost increases or decreases depending entirely on the values of C_P, C_I, α, r_1, r_2 and r_3.

This brings us to a very meaning observation. From the debugging cost viewpoint, the decision as to whether an debugging should be perfect debugging does not depend on the β with which that is perfect debugging but on the other parameters in the model. For example, perfect debugging and imperfect debugging costs C_P and C_I in testing process might be such that the condition $C_P < C_I \frac{\alpha(r_2+r_3)}{\alpha(r_1+r_2)+r_1r_2}$ is satisfied. When the value of the debugging cost decreases, this means that the test benefits from perfect debugging. In this case, supposed that the total cost continues to decrease when β is increased, it is always better to perform fault perfect debugging as soon as the testing phase enters its fault state S_F, if the condition $C_P < C_D \frac{\alpha(a_2+a_3)}{\alpha(a_1+a_2)+a_1a_2}$ is satisfied. Similarly, consider a test process with a perfect debugging cost C_P is greater than $C_I[\frac{\alpha(r_2+r_3)}{\alpha(r_1+r_2)+r_1r_2}]$. If perfect debugging is performed on this testing phase, the total cost increases when the rate of starting perfect debugging β is increased. This implies that this test will not benefit from any perfect debugging at all.

From the above discussion, we can derive that there is a threshold effect. When β=0, there is no perfect debugging and the Debugtime and cost values can be computed as shown earlier. When β is increased, the Debugtime increases or decreases depending entirely on whether the condition $r_3 < r_1(1+\frac{r_2}{\alpha})$ is satisfied or not. Similarly, when β is increased and C_P is independent of β, the cost due to perfect debugging increases or decreases depending entirely on the condition $C_P > C_I \frac{\alpha(r_2+r_3)}{\alpha(r_1+r_2)+r_1r_2}$ is satisfied or not, cost functions of perfect debugging continue to increase or decrease as long as those condition hold. For the best results, we should perform perfect debugging as soon as a testing phase enters finding faults state. i.e. make $\beta = \infty$, (assuming that the cost functions do not change) or prevent a key component or module in code from imperfect debugging, i.e. make α =0.

C. Discussion of Relation between Imperfect and Perfect Debugging

The model we proposed in section II is helpful to determine whether perfect debugging will be useful or not. However, debugging does not have to be applied uniformly to every component or module in code. One can decide which of the components or modules are vulnerable to fault and choose them for doing carefully perfect debugging from the practice viewpoint. We also can choose the trade-off point to balance imperfect and perfect debugging, so as to reach the best fault debugging. With the proper analysis, selection and tuning of those parameters, as described in section Ⅲ below, it should be possible to obtain high availability and reduced cost due to debugging in many testing processes.

Obviously, determining those vulnerable component or modules in programme, fault detection rate, fault debugging rate and cost due to imperfect and perfect debugging, for a practical test, it is a difficult and imprecise task. Most tests collect fault data, do analysis of their fault and keep track of their costs, fault data collected from those cases may provide some insights into the values for those variables [22]. From those values, one can use Equations 13 and 14 to determine whether fault debugging would be beneficial at all or not. If it is beneficial, one can compute the cumulating Debugtime and cost due to perfect debugging using Equation 10 and 11 for a range of fault debugging rates and pick an appropriate fault debugging periods to prevent from imperfect debugging.

Observe that cost due to fault debugging, for the perfect debugging to be beneficial, the condition in Equation 14 should be negative which implies that C_P should must be less than C_I, i.e. the debugging cost after perfect debugging should be much less than that cost after imperfect debugging.

Therefore, the debugging time and perfect debugging period should be chosen carefully. Perfect debugging time of day, day of the week etc. Should be chosen when it is least disruptive to the separate debugging and collective debugging. Those times can perhaps be obtained from the activity intensity profile of a debugging process.

We discussed only the a_2 is the constant, if $r_2 = \infty$ [22], substituting this value in Equation 4 and 10 shows that no matter how fast debugging procedure is completed, Debugtime with perfect debugging will always be larger than the Debugtime with imperfect debugging. However, substituting $r_2 = \infty$ in Equation 11 and differentiating it with respect to β yields the threshold condition $C_P < C_D \frac{\alpha}{\alpha+r_1}$ for perfect debugging to be beneficial in this case. This condition implies that the cost of perfect debugging is to be much less than imperfect debugging cost after fault occurred for this case ($r_2 = \infty$).

III. NUMERAL EXAMPLES

Let us now look at some examples to illustrate benefits of perfect debugging quantitatively.

Example 1. Consider a test example 1 with the following “fault debugging profile”.

①. It takes 2 hours to go from state S_F to S_I. Then

than the previous value, the Debugtime will increase shold for

TABLE II.
FAULT DEBUGGING PROFILE OF APPLICATION EXAMPLE 1.

	No perfect debugging ($\beta = 0$)	Once every 4 hour($\beta = 0.25$)	Once every 2 hour($\beta = 0.5$)
Debugtime(hours)	0.5581	2.5075	3.4286
Cost (dollars)	558.1395	573.1343	580.2198
Parameters	$\alpha = 0.5,\ r_1 = 3,\ r_2 = \frac{1}{12},\ r_3 = \frac{1}{4}, C_I = 1000,\ C_P = 100,\ L = 1 \times 24$		

TABLE III.
FAULT DEBUGGING PROFILE OF APPLICATION EXAMPLE 2.

	No perfect debugging ($\beta = 0$)	Once every 4 hour($\beta = 0.25$)	Once every 2 hour($\beta = 0.5$)
Debugtime(hours)	0.8276	2.6667	3.541
Cost (dollars)	827.5862	746.6667	708.1967
Parameters	$\alpha = 0.5,\ r_1 = 2,\ r_2 = \frac{1}{12},\ r_3 = \frac{1}{4}, C_I = 1000,\ C_P = 100,\ L = 1 \times 24$		

TABLE IV.
FAULT DEBUGGING PROFILE OF APPLICATION EXAMPLE 3.

	No perfect debugging ($\beta = 0$)	Once every 4 hour($\beta = 0.25$)	Once every 2 hour($\beta = 0.5$)
Debugtime(hours)	6.3158	6.2222	6.1714
Cost (dollars)	6315.8	4622.2	3702.9
Parameters	$\alpha = 0.5,\ r_1 = \frac{1}{5},\ r_2 = \frac{1}{12},\ r_3 = \frac{1}{4}, C_I = 1000,\ C_P = 100,\ L = 1 \times 24$		

$$\alpha = \frac{1}{2} = 0.5 .$$

②. It takes 20 minutes to from state S_I to S_0, that is to need 20 minutes to remove a fault. Then $r_1 = 3.$

③. It takes 12 hours from test starting state to fault state; so $r_2 = \frac{1}{12}.$

④. It takes 4 hours to go from state S_P to S_0, that is to remove a fault. Then $r_3 = \frac{1}{4}.$

⑤. The average cost of debugging a fault, that is imperfect debugging, C_I, is \$1000 per hour.

⑥. The average cost of debugging a fault, that is perfect debugging, C_P, is \$100 per hour.

⑦. An interval of L time units is 1 day, that is L= 1×24.

Let us first consider the Debugtime. The express $r_1(1 + \frac{r_2}{\alpha})$ in Equation 13 can be computed to be $3[1 + \frac{1}{6}] = 3.5.$ Since $r_3 = \frac{1}{4} = 0.25$ and is much less the cost function is computed, we find that $C_I[\frac{\alpha(r_2 + r_3)}{\alpha(r_1 + r_2) + r_1 r_2}]$ evaluates to \$93.0233. The cost of perfect debugging C_p is \$100 per hour and this means that, when β is increased, the cost due to perfect debugging will increase. Thus both the Debugtime and cost due to perfect debugging increase when β is increased above zero and so perfect debugging is not beneficial in this test. Note that β is the rate of starting perfect debugging before testing phase goes into removing a fault state. Therefore, if an test is to start perfect debugging every 4 hours, $\beta = \frac{1}{4}.$ The Table 2 illustrates how those values increase when the rate of β is increased.

Example 2. Consider a test example 2 with the following "fault debugging profile".

①. It takes 2 hours to go from state S_F to S_I. Then

$$\alpha = \frac{1}{2} = 0.5 .$$

②. It takes 30 minutes to from state S_I to S_0, that is to need 30 minutes to remove a fault. Then $r_1 = 2$.

③. It takes 12 hours from test starting state to fault state; so $r_2 = \frac{1}{12}$.

④. It takes 4 hours to go from state S_P to S_0, that is to remove a fault. Then $r_3 = \frac{1}{4}$.

⑤.The average cost of debugging a fault, that is imperfect debugging, C_I, is \$1000 per hour.

⑥. The average cost of debugging a fault, that is perfect debugging C_P, is \$100 per hour.

⑦. An interval of L time units is 1 day, that is L= 1×24.

Let us first consider the Debugtime. The express $r_1(1+\frac{r_2}{\alpha})$ in Equation 13 can be computed to be $2[1+\frac{1}{6}] = 2.3333$. Since $r_3 = \frac{1}{4} = 0.25$ and is much less than the previous value, the Debugtime will increase when β is increased. Similarly, when the threshold for the cost function is computed, we find that $C_I[\frac{\alpha(r_2+r_3)}{\alpha(r_1+r_2)+r_1r_2}]$ evaluates to \$137.931. The cost of perfect debugging C_p is \$100 per hour and this means that, when β is increased, the cost due to perfect debugging will decrease. Thus, in this example, although the Debugtime increases when β is increased, the cost due to perfect debugging decreases when β is increased, So, if cost is the only factor to decide on the testing process, it is beneficial to perform perfect debugging as carefully as possible, as long as the above threshold condition is not violated. The Table 3 shows this behavior in this case.

Example 3. Consider a test example 3 with the following "fault debugging profile".

①. It takes 2 hours to go from state S_F to S_I. Then $\alpha = \frac{1}{2} = 0.5$.

②. It takes 5 hours to from state S_I to S_0, that is to need 5 hours to remove a fault. Then $r_1 = \frac{1}{5}$.

③. It takes 12 hours from test starting state to fault state; so $r_2 = \frac{1}{12}$.

④. It takes 4 hours to go from state S_P to S_0, that is to remove a fault. Then $r_3 = \frac{1}{4}$.

⑤. The average cost of debugging a fault, that is imperfect debugging, C_I, is \$1000 per hour.

⑥. The average cost of debugging a fault, that is perfect debugging, C_P, is \$100 per hour.

⑦. An interval of L time units is 1 day, that is L= 1×24.

Let us first consider the Debugtime. The express $r_1(1+\frac{r_2}{\alpha})$ in Equation 13 can be computed to be $\frac{1}{5}[1+\frac{1}{6}] = 0.2333$. Since $r_3 = \frac{1}{4} = 0.25$ and is much greater than the previous value, the Debugtime will decrease when β is increased. Similarly, when the threshold for the cost function is computed, we find that $C_I[\frac{\alpha(r_2+r_3)}{\alpha(r_1+r_2)+r_1r_2}]$ evaluates to \$1052.6. The cost of perfect debugging C_p is \$100 per hour and this means that, when β is increased, the cost due to perfect debugging will decrease. Thus both the Debugtime and cost due to perfect debugging decrease when

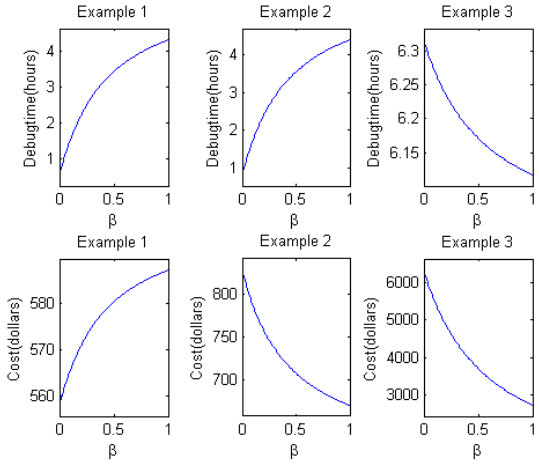

Figure 3. Compared with Debugtime and Cost in Three Examples.

β is increased above zero and so perfect debugging is beneficial in this case. The Table 4 illustrates how those values decrease when the rate of β is increased. In order to compare with Debugtime and Cost in Example 1, Example 2 and Example 3, Figure 3 shows the difference in them.

IV. CONCLUSIONS

In this paper, we propose the debugging model based on probabilistic state transition. We have analyzed the relation between imperfect debugging and perfect debugging using the debugging model and how to estimate the costs of introducing perfect debugging policies of the software versus imperfect debugging policies of the software. The numeral experimental results show through analyzing the relation between imperfect and perfect debugging, it illustrates perfect debugging is an important influence to imperfect debugging in software testing process. It also indicates it is important for the effective system run by determining the threshold between Debugtime and cost due to software debugging. They can effectively decrease time and cost in the practice software test through preventing

the imperfect debugging from the key component or module in software.

On the other hand, software testing is a complex and intractable process including many important factors, such as tester skill, testing tool, tester psychology as well as testing internal and external environment etc.. Perfect debugging is closely related to imperfect debugging. Thus, research on perfect and imperfect debugging is a strong practical significance. In the future, we will focus on doing more sensitivity analysis on the debugging model, in particular, investigate properties of the formula produced by the debugging model. We also will comprise experiments to assess exactness of the theoretical results and effectiveness of perfect debugging in this paper.

ACKNOWLEDGMENT

We would like to thank National Nature Science Foundation of China (NSFC) for its support of this research under grant number 61202091.

REFERENCES

[1] J. Gray and A. Reuter, Zhwaction Pmessing: Concepts and Techniques, Morgan Kaufmann Publishers, San Mateo, CA, 1993.
[2] A.L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," IEEE Transactions on Reliability R28, 1979, pp. 206–211.
[3] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," IEEE Transactions on Reliability R-32, 1983, pp. 475–484.
[4] S. S. Gokhale, M. R. Lyu, and K. S. Trivedi, "Analysis of software fault removal policies using a non-homogenous continuous time Markov chain," Software Quality Journal, vol. 12, 2004, pp. 211–230.
[5] P. K. Kapur and R. B. Garg, "A software reliability growth model for an error removal phenomenon," Software Engineering Journal, vol. 7, 1992, pp. 291–294.
[6] J. H. Lo and C. Y. Huang, "An integration of fault detection and correction, processes in software reliability analysis," The Journal of Systems and Software, vol. 79, 2006, pp. 1312–1323.
[7] N. F. Schneidewind, "Analysis of error processes in computer software," Sigplan Notices, vol. 10, 1975, pp. 337–346.
[8] N. F. Schneidewind, "Modelling the fault correction process," in Proceedings of the 12th International Symposium on Software Reliability Engineering, Los Alamitos, CA, IEEE Computer Society Press, 2001, pp. 185–190.
[9] M. Xie, Q. P. Hu, Y. P. Wu, and S. H. Ng, "A study of the modeling and analysis of software fault-detection and fault-correction processes," Quality and Reliability Engineering International, vol. 23, 2006, pp. 459–470.
[10] M. Xie and M. Zhao, "The Schneidewind software reliability model revisited," in Proceedings of the 3rd International Symposium on Software Reliability Engineering, pp. 184–192, May 1992.
[11] S. Yamada, M. Obha, and S. Osaki, "S-shaped software reliability growth modeling for software error detection," IEEE Trans. on Reliability, vol. R-32, no. 5, 1983, pp. 475–484.
[12] M. A. Stutzke and C. S. Smidts, "A stochastic model of fault introduction and removal during software development," IEEE Trans. On Reliability, vol. 50, 2001, pp. 184–193.
[13] A. L. Goel, "Software reliability models: Assumptions, limitations and applicability," IEEE Trans. on Software Engineering; SE-11, 1985, pp. 1411–1423.
[14] P. K. Kapur, D. Kumar, A. Gupta, and P. C. Jha, "On how to model software reliability growth in the presence of imperfect debugging and error generation," in Proceedings of 2nd International Conference on Reliability and Safety Engineering, 2006, pp. 515–523.
[15] P. K. Kapur and R. B. Garg, "Optimal software release policies for software reliability growth model under imperfect debugging," RAIRO, vol. 24, 1990, pp. 295–305.
[16] M. Ohba and X. M. Chou, "Does imperfect debugging effect software reliability growth," in Proceedings of 11th International Conference of Software Engineering, 1989, pp. 237–244.
[17] X. Zhang, X. Teng, and H. Pham, "Considering fault removal efficiency in software reliability assessment," IEEE Trans. on Systems, Man and Cybernetics-Part A: Systems and Humans, vol. 33, no. 1, 2003, pp. 114–120.
[18] H. Pham, L. Nordmann, and X. Zhang, "A general Imperfect software debugging model with S-shaped fault detection rate," IEEE Trans. On Reliability, vol. R-48, no. 2, 1999, pp. 169–175.
[19] H. Pham, "A software cost model with imperfect debugging, random life cycle and penalty cost," International Journal of Systems Science, vol. 27, no. 5, 1996, pp. 455–463.
[20] P. K. Kapur, H. Pham, S. Anand, and K. Yadav, "A Unified Approach for Developing Software Reliability Growth Models in the Presence of Imperfect Debugging and Error Generation," IEEE Transactions on Reliability, vol. 60, no. 1, 2011, pp. 331-340.
[21] C.Y. Huang and M. R. Lyu: "Estimation and Analysis of Some Generalized Multiple Change-Point Software Reliability Models," IEEE Transactions on Reliability, vol. 60, no. 2, 2011, pp. 498-514.
[22] Y. Huang, C. Kintala, N. Kolettis and N. D. Fulton, "Software Rejuvenation: Analysis, Module and Applications," Twenty-Fifth International Symposium on Fault-Tolerant Computing (Pasadena, CA, IEEE Computer Society), 1995, pp. 381-390.
[23] Z. Zhang, H. Song, Y. Li, and Hao. Yang, "Prediction Algorithm for State Prediction Model," Journal of Computers, Vol. 7, No. 2, pp. 507-513, February 2012.
[24] H, Zeng. "Efficient Graduate Employment Serving System based on Queuing Theory," Journal of Computers, Vol. 7, No. 9, 2012, pp. 2176-2183.
[25] Y. Gao, C. Zhang, and L. Zhang, "Comparison of GARCH Models based on Different Distributions," Journal of Computers, Vol. 7, No. 8, 2012, pp. 1967-1973.
[26] D. Wang, Y. Li, and X. Sun, "Using Markov Process for Passenger Structure Prediction within Comprehensive Transportation Channel," Journal of Computers, Vol. 8, No. 4, 2013, pp. 1072-1077.

Jinyong Wang is a doctoral candidate at Fault Tolerance and Mobile Computing Research Center, Harbin Institute of Technology, Heilongjiang Province, China. His research interests include fault tolerance, software reliability and natural language process.

Zhibo Wu is a Professor at Fault Tolerance and Mobile Computing Research Center, Harbin Institute of Technology,

Heilongjiang Province, China. His research interests include fault tolerance, software reliability and artificial intelligence.

Yanjun Shu is a lecturer at Fault Tolerance and Mobile Computing Research Center, Harbin Institute of Technology, Heilongjiang Province, China. His research interests include fault tolerance and software reliability.