A NHPP Software Reliability Growth Model Considering Learning Process and Number of Residual Faults

1 Tao Li, 2 Kaigui Wu

1 College of Computer Science, Chongqing University, Chongqing, China, 334623078@qq.com
2 College of Computer Science, Chongqing University, Chongqing, China, kaiguivu@cqu.edu.cn

Abstract

A large number of software reliability growth models have been proposed to analyze the reliability of software application during the testing phase. With the increasing demand to deliver high-quality software, more accurate software reliability models are required to estimate the optimal software release time and the cost of testing efforts. In this paper we firstly demonstrate that the G-O model based on NHPP doesn’t need to consider imperfect debugging and new mistakes introduction during the debugging process. Then on the basis of G-O model a flexible and accurate SRGM is proposed, which considers the time-dependent fault detection rate. And in this model the value of fault detection rate is not only calculated with the number of faults remaining in the software system, but also calculated with human’s learning process during testing phase. It is more realistic. Moreover, the experiment results show that the proposed model fits the public failure data better and can provide more accurate software reliability prediction compared with other existing models.

Keywords: Software Reliability Growth Model (SRGM), Non-Homogeneous Poisson Process (NHPP), Fault detection rate, Imperfect Debugging, Learning process

1. Introduction

The subject Software Engineering has evolved to make possible the formidable task of developing quality software economically. Software reliability which is defined as the probability of failure-free software operation for a specified period of time in a specified environment [1] [2] is the most important quality metric. Software reliability and the cost of software development have become a crucial standard of software quality. The cost of software development is inseparable with the estimation of software reliability. So the accurate and concise estimation of software reliability can minimize the cost of software development and get more profit. Therefore, a large number of software reliability growth models based on NHPP have been proposed to analyze the reliability of software application during the testing phase [3-7]. Such as J-M model which was proposed at first, and on the basis of J-M model Goel and Okumoto put forward G-O model. This model incorporates imperfect debugging into software reliability model at the first time. Then Yamada improved G-O model and suggested an S-Shaped model [8] [9], in which the trend of software reliability can be described as an S-Shaped curve. Since G-O model has the advantage of simple structure and estimation precision, many models are improved based on it through changing its assumptions. The new model will fit the failure data better after correction and improvement on the basis of G-O model.

Many improved models which based on G-O software reliability growth models consider imperfect debugging during software testing phase. Both the fault introduction and the fault removal efficiency are considered in this model [3] [10]. And a time-dependent fault removal efficiency function is introduced [11] [12] [13]. With the increasing demand to deliver high-quality software, more accurate software reliability models are required to estimate the optimal software release time and the cost of testing efforts. In this paper we discuss a concise and accurate G-O model based on NHPP. We have proved that this model needn’t consider imperfect debugging and new mistakes introduction during the debugging process. So we can simplify the structure of model without losing accuracy of reliability prediction. And then a modified NHPP software reliability growth model which considers time dependent fault detection rate is proposed. Finally, we show a numerical example with real software failure data, employing the proposed models. Comparing with some other existing NHPP models, the new model can provide a better goodness-of-fit.
Notations used:

\( t \): Time
\( N(t) \): Cumulative number of faults detected in the time interval \((0, t]\)
\( m(t) \): Expected number of faults detected in the time interval \((0, t]\)
\( b \): Initial value of fault detection rate
\( b(t) \): Fault detection rate at time \( t \)
\( \lambda(t) \): \( \frac{dm(t)}{dt} \), the failure intensity function for \( m(t) \).
\( a \): Number of errors to be eventually detected
\( R(x|t) \): Software reliability, the probability of no failures in \((t, x + t]\) given that the most recent failure occurred at time \( t \)

2. Demonstration of Imperfect Debugging

Both the fault introduction and the fault removal efficiency are considered in the model based on imperfect debugging. Below we will introduce classic G-O model and prove that the proposed model needn’t consider two aspects of imperfect debugging during the fault removal process on the basis of G-O model.

2.1. G-O Model

We consider software failure process on the basis of G-O model. The detection of software failure is a random process. We assume \( N(t) \) represents the number of faults detected in the time interval \((0, t]\) \((t > 0)\), which is a random number and \( N(0) = 0 \). So the fault detection process \( \{N(t), t \geq 0\} \) is a non-homogeneous Poisson process.

General assumptions of G-O model:
1. Failure observation is modeled by NHPP;
2. Value of fault detection rate is constant;
3. Each software failure and fault removal occurs independently;
4. Each time a failure occurs, the error which caused it is immediately removed, and no other errors are introduced;
5. The probability of failure occurring is proportional to the number of residual faults which are not yet observed;
6. The probability of fault detection is proportional to the number of residual faults which are not yet observed.

According to the several assumptions above we can construct a G-O model as follows:

\[
\frac{dm(t)}{dt} = b(a - m(t))
\]  

(1)

Solve the equation, we can get:

\[
m(t) = a(1 - e^{-bt})
\]  

(2)

Two points should be presented about G-O model:

- The parameter \( a \) denotes the number of faults to be eventually detected rather than the initial value of number of faults in software system. So it concludes the introduction of new faults during debugging a fault;
- The parameter \( b \) denotes fault detection rate. And we assume the value of \( b \) is constant in G-O model and it will not change with the time. This presents that the probability of every fault detected is invariable every time.
2.2. Demonstration of Fault Introduction

In fact, the parameter \( a \) represents number of faults to be eventually detected in G-O model. So this model has actually considered introduction of faults. Secondly, though we regard the parameter \( a \) as initial value of faults when testing begins, in literature [10] it also has been proved that fault introduction has no effect on software reliability in this model. Below is the brief description:

Since fault removal and new fault introduction are random process, we assume the probability of introducing a new fault during removing a fault is \( \beta \). So introducing \( \beta \) into G-O model the equation becomes from (2) to the following form:

\[
m'(t) = \frac{a}{1-\beta} (1-e^{-t(1-\beta)\beta})
\]

(3)

Assuming: \( a' = \frac{a}{1-\beta} \quad b' = (1-\beta)\phi \)

So we can get:

\[
m'(t) = a' (1-e^{b't})
\]

(4)

From the equation (4) and (2) above it can be found that the form of \( m(t) \) and \( m'(t) \) is exactly the same. So it has no critical effect on accuracy of reliability prediction whether incorporating fault introduction or not.

2.3. Demonstration of Fault Removal Efficiency

Assuming the fault removal efficiency is \( p \), equation is written as:

\[
\frac{dm(t)}{dt} = b(a - p \ast m(t))
\]

(5)

Solving the equation above we get:

\[
m'(t) = \frac{a}{1-\beta} (1-e^{-t(1-\beta)\beta})
\]

(6)

Assuming: \( a' = \frac{a}{1-\beta} \quad b' = (1-\beta)\phi \)

So we get:

\[
m'(t) = a' (1-e^{b't})
\]

(7)

From the equation (7) and (2) above we can get the same conclusion with chapter2.2 that the form of \( m(t) \) and \( m'(t) \) is exactly the same. So the fault removal efficiency has no critical effect on accuracy of reliability prediction in this model.

3. Proposed Software Reliability Model Considering Fault Detection Rate

The demonstration of imperfect debugging above indicates that both the fault introduction and the fault removal efficiency will vary with the sample, and they have no critical effect on accuracy of reliability estimation on the basis of G-O model. Instead the parameters \( b \), which indicates fault detection rate during testing phase, plays a significant role to guarantee the accuracy of software reliability growth modeling.

From practical field studies, it is evident that one can estimate the testing efforts consumption pattern and predict the trends of the fault detection rate. The fault detection rate is not constant apparent. During the process of software system testing phase testers will be gradually familiar with
the software system with the time, which has a positive effect on the value of fault detection rate; On the other hand, with the increase of testing time the number of residual faults is less and less. Then it will be gradually difficult to detect the faults in software system. So the time dependent fault detection rate should be estimated by considering the two aspects above jointly.

We suggest the proposed model has the following explicit assumptions:

1. Failure observation and fault removal phenomenon is modeled by NHPP;
2. Testers’ learning capability is a non-decreasing function which is relevant to testing time [14] [15] [16];
3. Each software failure and fault removal occurs at independently;
4. The detection rate of residual faults in software system is a non-increasing time dependent function;
5. The probability of failure occurring is proportional to the number of residual faults which are not yet observed.
6. The probability of fault detection is proportional to the number of residual faults which are not yet observed.

According to the several assumptions above we can construct a differential equation of G-O model as follows:

$$\frac{dm(t)}{dt} = b(t)(a - m(t))$$  \hspace{1cm} (8)

Since the parameter $b$ which is between 0 and 1 is a time-dependent function and it is estimated by testers’ learning ability and the number of residual faults in software system, then we can get:

$$b_1(t) = 1 - (1-b)e^{-k_1t}$$  \hspace{1cm} (9)

$$b_2(t) = be^{-k_2t}$$  \hspace{1cm} (10)

In (9), $b_1(t)$ is the fault detection rate incorporating the rise of human learning ability. Whereas, in (10) $b_2(t)$ is the fault detection rate considering that it is more and more difficult to find the remaining faults with time in software system. So considering these two factors jointly the equation of estimating fault detection rate $b(t)$ is written as:

$$b(t) = b_1(t) * b_2(t) = e^{-k_1t} - (1-b)e^{-(k_1+k_2)t}$$  \hspace{1cm} (11)

$$m(0) = 0$$  \hspace{1cm} (12)

We define that $m(0) = 0$, and using the equation (8), (11) and (12) to solve the equation, we can get the joint function as below:

$$m(t) = a(1-e^{-k_2t}) - \frac{(exp(-k_2t)-1)((1-b)exp(-k_1t)-1)}{k_2}$$  \hspace{1cm} (13)

As the proposed model is based on non-homogeneous Poisson process, the probability of observing $n$ faults until time $t$ is:

$$P\{N(t) = n\} = \frac{m(t)^n}{n!}e^{-m(t)}$$  \hspace{1cm} (14)

Finally, we can get the estimation function of software reliability according to the Poisson distribution:

$$R(x | t) = P\{N(t + x) - N(t) = 0\} = e^{-(m(t)+m(t))}$$  \hspace{1cm} (15)
4. Illustrative Examples

The performance evaluation of software reliability growth model is generally measured with sum of square errors (SSE) and correlation index of regression curve equation (R-square). Among them, the model performance is better when SSE is smaller and R-square is close to 1 compared with other models.

SSE is used to describe the distance between actual and estimated number of faults detected totally, which is defined as

\[ \text{SSE} = \sum_{i=1}^{n} (y_i - m(t_i))^2 \]

Where \( n \) denotes the number of failure samples in failure data set, \( y_i \) denotes the number of faults observed to the moment \( t_i \), and \( m(t_i) \) denotes the estimated number of faults detected to the time \( t_i \) according to the proposed model. The model can provide a better goodness-of-fit when the value of SSE is smaller.

The equation of calculating the value R-square is written as:

\[ \text{R-square} = \frac{\sum_{i=1}^{n} (\bar{y} - m(t_i))^2}{\sum_{i=1}^{n} (\bar{y} - y_i)^2} \]

Where \( \bar{y} \) denotes the mean value of faults detected. The model can provide a better goodness-of-fit when the value of R-square is close to 1.

Table 1. Data set from literature [17]

<table>
<thead>
<tr>
<th>Test period (week)</th>
<th>CPU hours</th>
<th>Defects found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>519</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>968</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>1430</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>1893</td>
<td>33</td>
</tr>
<tr>
<td>5</td>
<td>2490</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>3058</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>3625</td>
<td>54</td>
</tr>
<tr>
<td>8</td>
<td>4422</td>
<td>58</td>
</tr>
<tr>
<td>9</td>
<td>5218</td>
<td>69</td>
</tr>
<tr>
<td>10</td>
<td>5823</td>
<td>75</td>
</tr>
<tr>
<td>11</td>
<td>6539</td>
<td>81</td>
</tr>
<tr>
<td>12</td>
<td>7083</td>
<td>86</td>
</tr>
<tr>
<td>13</td>
<td>7487</td>
<td>90</td>
</tr>
<tr>
<td>14</td>
<td>7846</td>
<td>93</td>
</tr>
<tr>
<td>15</td>
<td>8205</td>
<td>96</td>
</tr>
<tr>
<td>16</td>
<td>8564</td>
<td>98</td>
</tr>
<tr>
<td>17</td>
<td>8923</td>
<td>99</td>
</tr>
<tr>
<td>18</td>
<td>9282</td>
<td>100</td>
</tr>
<tr>
<td>19</td>
<td>9641</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>10000</td>
<td>100</td>
</tr>
</tbody>
</table>

The set of software faults analyzed here was obtained from literature [17]. In this paper we test the performance of the proposed model by using the data in table 1. As shown in figure 1, the growth curve of detected faults for the actual data is S-shaped. This presents that in the initial stages of software testing the number of faults is so big that tester’s learning ability dominates the fault detection rate. And in later period of software testing it is more and more difficult to find the residual faults in software system, so the number of residual faults starts to dominate the fault detection rate instead of tester’s learning process. This situation is more consistent with the actual circumstance and we get more realistic experiment result.
A NHPP Software Reliability Growth Model Considering Learning Process and Number of Residual Faults
Tao Li, Kaigui Wu

Figure 1. Actual and estimated number of faults detected in the time interval

According to the value of parameter estimation in different software reliability growth model we can calculate the approximation of SSE and R-square. Then comparing with some classic models and new models such as G-O model, delayed S-shaped model, inflection S-shaped model and Bell-SRGM model [14], we obtain the results as follows:

<table>
<thead>
<tr>
<th>Model</th>
<th>SSE</th>
<th>R-square</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-O model</td>
<td>155.2</td>
<td>1.0707</td>
</tr>
<tr>
<td>Delayed s-Shaped model</td>
<td>824.1</td>
<td>1.2510</td>
</tr>
<tr>
<td>Inflection s-shaped model</td>
<td>144.6</td>
<td>1.0419</td>
</tr>
<tr>
<td>Bell-SRGM model</td>
<td>147.3</td>
<td>0.9929</td>
</tr>
<tr>
<td>The proposed model</td>
<td>140.5</td>
<td>1.0546</td>
</tr>
</tbody>
</table>

From the table 2 it can be seen that the value of SSE is smaller and the value of R-square is more close to 1 in the experiment of the proposed new model compared with other models. The results indicate that our NHPP model based on fault detection rate fits the data in table 1 the best and predicts the number of residual faults in software most accurately. Moreover, Bell-SRGM model also took into account the human’s learning ability and the number of residual faults in software system. But as selecting a different function Bell-SRGM model has a difficult problem of solving the integration equation. And the problem is finally solved by changing the question of integral into the question of finding the sum, which is comparatively difficult. In this paper the integral of the function can be directly obtained rather than transforming it into question of finding the sum. So the proposed model can achieve a better accuracy.

5. Conclusion

Software reliability growth model can estimate the optimal software release time and the cost of testing efforts [18]. And SRGM can help project managers to determine the testing resources and manpower needed to achieve desired reliability requirements. So more accurate model is needed to decrease the testing cost and increase the profit of releasing software [19] [20] [21]. In this paper we demonstrate that the improved G-O model doesn’t need to consider imperfect debugging, and then a time-dependent fault detection rate model is presented. In this model the fault detection rate is calculated with the number of faults remaining in the software and the human’s learning ability. Considering the two factors jointly the fault detection rate is more realistic and accurate. Moreover, we have discussed the performances of our new SRGM by using actual software failure data. The experiment result shows the new model can provide a better goodness-of-fit compared with other models.

In further studies, we need to check the validity and effectiveness of our modeling framework and SRGMs developed under the modeling framework by using many actual data. And we will develop a software cost model on the foundation of the presented new SRGM. This model takes into account the total number of faults discovered by users during the period of software in use or software maintenance
after the software is released. Then under this condition we will analyze the cost model to determine the optimal software release time to minimize the cost of software development and get more profit when releasing the software.

6. Acknowledgments

This work is supported by the Major Research Project of the National Natural Science Foundation of China under Grant No.90818028, Natural Science Foundation Project of CQ CSTC: 2011BB2064.

7. References
