A Software Reliability Prediction Method Based on Process Information

1Weipeng Wang, 1Yumei Wu and 2Lei Sun
1School of Reliability and Systems Engineering, Beihang University, Beijing, 100191, China
2Sino Avionics Technology Co., Ltd, Beijing, 100000, China

Abstract: In this study, we propose an approach about software reliability prediction method based on process information. Compared with traditional software prediction methods, inherent characteristics and development characteristics are used in our method to replace test data for predicting software reliability. So this method can implement at the beginning of software development cycle and guide software reliability improving. The article gives detailed information about the method, implementation process of it. And at last, we use store management system as an example to verify our method.

Key words: Software reliability, reliability prediction, process information, early stage prediction

INTRODUCTION

Traditional software reliability prediction methods normally have three key factors: Software reliability model, parameter estimation algorithm and test data. And the prediction can only be implemented after obtaining test data from the software testing (Cheung, 1980; Littlewood, 1979; Rabiner, 1989). In this case, the prediction has few effects on improving software reliability. In order to improve the software reliability, reliability prediction must be implemented at the beginning of the software development cycle. Although the software product doesn’t exist at the beginning of the development cycle, the process information, such as functional requirements, application type and development environment, are clear (Brosch et al., 2012; Godhale, 2007; Godhale and Trivedi, 2006). In this study, we propose an approach of predicting software reliability with process information.

PREDICTION METHOD BASED ON PROCESS INFORMATION

The software reliability prediction method based on process information is a procedure of decomposition and compound. According to the mission and requirement of the software, we can decompose the software into a few modules and each module is a set of a series of functions. It is easier for us to understand the process information of a module compared with the software as a whole. We can use process information to predict the reliability of each module.

The execution of a software means that modules connect with each other following a specific topology. In this approach, we use Markov chain to build model of the software topology. Combining with the reliability of each module and software topology, we can calculate the reliability of the software.

Software functional decomposition: Before developing software, we need to analyze software requirements, confirm software functions and make functional decomposition. Software functional decomposition is similar to hardware function decomposition, namely decomposing the software based on the functions it achieves.

Software functional decomposition usually has six grades, including computer software configuration item, computer software component, unit, module, instruction and operands. As an entity of a series of functions, module is detailed enough to make reliability prediction with process information and contribute to construct software topology compared with instruction and operands. As a consequence, we choose module as the grade of the functional decomposition.

A good decomposition should be well-defined and easy for achieving. The coupling between modules should be as less as possible.

Mission profile description: After decomposing the software into several modules, we need to describe mission profile of the software. The mission profile is a specific topology of the modules and the software can obtain its function under the mission profile.

We consider that it is a state of the software that each module successfully completes its function and the execution process of the mission profile is the process of
charge of state. In the mission profile, the transition probability between different modules is transition probability of state of the software. (Jin, 2011)

In order to get realistic transition probability, we can use similar products method and expert estimation method. If some modules of the software are reused from other software, we can research their transition probability in its original software and use it in the mission profile. Otherwise, we can ask experts to estimate transition probability with their experiences.

After getting transition probability between modules, we can use directed graph to describe the mission profile. Each node \( N_i \) stands for a module, directed edge \((N_i, N_j)\) stands for the transition possibility from \( N_i \) to \( N_j \). And the transition probability of directed edge \((N_i, N_j)\) is \( P_{ij} \). (Huang and Lyu, 2011; Torrado et al., 2013).

With the directed graph of mission profile, we can get Markov state transition matrix of the software.

Module reliability calculation: From criterion of MIL-HDBK-217D, we know that we can give reliability of a component based on its type, usage characteristics, usage environment, quality level and many other factors in hardware reliability prediction. Likewise, the reliability of software module based on module property, module application character, module development characteristics and other factors can also be achieved by detailed statistics study of the information from software development process and software itself.

We reference the criterion of CMM (Capability Maturity Model) and GJB 5000A-2008, integrate software engineering theory and summarize two categories of factors including inherent characteristics and development characteristics to describe module process information. The two factors can be divided into many specific sub-factors. After researching many kinds of software, we conclude the data of each specific sub-factor influencing module reliability. Using the data, we can predict module reliability.

Software reliability calculation: According to the knowledge of Markov chain, we can use Markov state transition matrix and reliability of each state, namely module reliability, to calculate the reliability of software.

Implementation process of the method: First, as the designers know the software most, they could decompose the software into many modules.

Second, under the function of the software, the designers could give mission profile of the software.

As stated previously it is a state of the software that each module successfully completes its function. We assume the software has \( n \) states, the set of states is \( S = (S_1, S_2,..., S_n) \), in the set \( S_i \) is the initial state and \( S_n \) is the end state. \( R_i \) is the reliability of state \( S_i \), \( P_{ij} \) is the transition probability from state \( S_i \) to \( S_j \). So we can conclude the state transition matrix \( P \) as follows:

\[
P = (P_{ij})_{(n \times n)} = [1, 0, 0, \ldots, 0] + [0, 1, 0, \ldots, 0] + \ldots + [0, 0, 1, \ldots, 0] + \ldots + [0, \ldots, 0, 1]
\]

And then, the designers and developers should give the level of every specific sub-factor for each module.

As was said, the factors influencing module reliability can be divided into two categories. The detailed information of inherent characteristics is as follows.

We use \( i \) to identify each factor’s number of order, for the module with \( i \)th inherent characteristics, \( ac(i, j), cc(i, j) \) and \( ioc(i, j) \) represent the ratio between number of errors of \( j \)th class and total number of errors (\( j = 1, 2, 3, 4 \)). The 1st class error is logic error, the 2nd class error is interface error, the 3rd class error is input/output error, the 4th class error is calculation error). The level given by designers and developers of each specific sub-factor will match a data of \( cc(i, j), cc(i, j) \) and \( ioc(i, j) \). So the whole factor of inherent characteristics for one module \( C(j) \) is:

\[
C(j) = \frac{\sum_{i=1}^{n} ac(i, j) + \sum_{i=1}^{n} cc(i, j) + \sum_{i=1}^{n} ioc(i, j)}{N_{ac} + N_{cc} + N_{ioc}}
\]

\( N_{ac}, N_{cc} \) and \( N_{ioc} \) are the total number of application characteristics, cohesion characteristics and input/output characteristics, respectively.

Similar to the inherent characteristics, the detail information of development characteristics is as follows.

For the module with \( i \)th development characteristics, \( di(i, j), sx(i, j), sta(i, j), sda(i, j) \) mean the probability of avoiding \( j \)th class error (\( j \)th class error has the same meaning with previous explanation). For the documents factor, we use \( d(j) \) to express the probability of avoiding \( j \)th class error for the module.
Table 2: Specific sub-factor of development characteristics

<table>
<thead>
<tr>
<th>Category</th>
<th>Specific sub-factor</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development characteristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documents</td>
<td>Software development plan</td>
<td>d(i,j)</td>
</tr>
<tr>
<td>Software reliability and security</td>
<td>Software redundancy</td>
<td>srs(i,j)</td>
</tr>
<tr>
<td>Software testing activity</td>
<td>Unit testing</td>
<td>sta(i,j)</td>
</tr>
<tr>
<td>Software development activity</td>
<td>Structured designing</td>
<td>sda(i,j)</td>
</tr>
<tr>
<td>Software development supporting activity</td>
<td>Software con</td>
<td></td>
</tr>
<tr>
<td>Software development management activity</td>
<td>Risk management</td>
<td>sdma(i,j)</td>
</tr>
</tbody>
</table>

\[
d(\hat{j}) = \prod_{i=1}^{n} (1 - d(i,j)) \tag{3}
\]

\[
N_d = \text{the total number of documents factor.}
\]

For other specific sub-factor of development characteristics, we can get the probability of avoiding jth level class error for the module:

\[
srs(i,j) = \prod_{i=1}^{n} (1 - srs(i,j)) \tag{4}
\]

\[
N_{srs} = \text{the total number of software reliability and security factor.}
\]

\[
sta(i,j) = \prod_{i=1}^{n} (1 - sta(i,j)) \tag{5}
\]

\[
N_{sta} = \text{the total number of software testing activity factor:}
\]

\[
sda(i,j) = \prod_{i=1}^{n} (1 - sda(i,j)) \tag{6}
\]

\[
S_{sta} = \text{the total number of software development activity factor:}
\]

\[
sdsa(i,j) = \prod_{i=1}^{n} (1 - sdsa(i,j)) \tag{7}
\]

\[
N_{sdsa} = \text{the total number of software development supporting activity factor:}
\]

\[
sdma(i,j) = \prod_{i=1}^{n} (1 - sdma(i,j)) \tag{8}
\]

\[
N_{sdma} = \text{the total number of software development management activity factor:}
\]

Combining all the inherent characteristics and development characteristics, we can obtain the module reliability:

\[
R_{M} = \sum_{i=1}^{n} \frac{C(j)A(j)}{(1-M)(1-A(j))} \tag{9}
\]

Where:

\[
A(j) = 1 - d(j) \times srs(k) \tag{10}
\]

\[
M(j) = 1 - sta(j) \times sda(j) \times sdma(j) \tag{11}
\]

At last, the reliability of the software is the successful transition probability from state S_i to S_j. Successful transition from S_i to S_j is not only related with transition probability P_{ij} but also with correctly executed probability of S_i, namely reliability of module i. So the successful transition probability from S_i to S_j is R_{ij}. The state transition matrix can rewrite as follows:

\[
M = (P_{ij})_{(n \times n)} = \begin{bmatrix}
1 & S_1 & S_2 & \cdots & S_n \\
(n-1) & SR, 1 & R & \cdots & 1 \\
& (n-1) & R & \cdots & 1 \\
& & \cdots & \cdots & \cdots \\
& & & 1 & R
\end{bmatrix} \tag{12}
\]

According to Markov theory, the reliability of the software is:

\[
R = (1 - R_{nn}) \left| \frac{E}{1 - M} \right| \tag{13}
\]

In the formula, I is identity matrix of n x n.

\[\text{CASE STUDY}\]

We use a store management system as an example to verify our method.
Fig. 1: Mission profile of store management system

Table 3: Prediction results of module reliability

<table>
<thead>
<tr>
<th>Module</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-test module</td>
<td>0.9973</td>
</tr>
<tr>
<td>Mode conversion module</td>
<td>0.9914</td>
</tr>
<tr>
<td>Navigation module</td>
<td>0.9929</td>
</tr>
<tr>
<td>Terminal data control module</td>
<td>0.9936</td>
</tr>
<tr>
<td>Stores load module</td>
<td>0.998</td>
</tr>
<tr>
<td>Program load module</td>
<td>0.9955</td>
</tr>
</tbody>
</table>

When store management system is in ground mission mode, we can decompose it into modules as self-test module, mode conversion module, navigation module, terminal data control module, stores load module, and program load module.

Based on the function of store management system, mission profile is given in Fig. 1.

In this figure, S means ‘start’ and E means ‘end’. According to the mission profile, we can get the state transition probability matrix:

\[
P = \begin{bmatrix}
0 & 0.6 & 0.4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

After giving the level of all specific sub-factors for each module, we can predict module reliability.

Combining with module reliability and state transition probability, we can get software reliability:

\[ R = 0.988 \]

CONCLUSION

It can conclude from the example that the method has good performance on software reliability prediction. It can be implemented at the beginning of software development as using process information instead of test data. This approach will urge designers and developers to focus their attention on software reliability, direct the software development process and obtain high reliability products.

REFERENCES


