RELIABILITY IMPROVEMENT PREDICTIVE APPROACH TO SOFTWARE TESTING USING MATHEMATICAL MODELING

D. Vivekananda Reddy¹ and Dr. A. Ramamohan Reddy²

¹Assistant Professor Department of CSE, S.V.University, Tirupati
²Professor Department of CSE, S.V.University, Tirupati

ABSTRACT

The main objective of any software testing is to improve software reliability. Many of previous testing methods did not pay much attention towards how to improve software testing strategy based on software reliability improvement. The reason to it as the relationship between software testing and software reliability is a very complex task and this is because due to the complexity of software products and development processes involved in it. However any Testing strategy of software in order to improve reliability must need to possess the ability to predict that reliability. For this purpose an approach is used called Model predictive control, which provides a good framework to improve that predictive effect. There is an main issue in model predictive control is that how to estimate the concern parameter. In this case, Empirical Bayesian method is used to estimate the concern parameter: Reliability. This proposed reliability improvement predictive approach to software testing using Empirical Bayesian method can optimize test allocation scheme on line. In this the case study shows that it is not definitely true for a software testing method that can find more defects than others can get higher reliability. And the case study also shows that the proposed approach can get better result in the sense of improving reliability than random testing.

KEYWORDS: Software Testing, Software Reliability, Model Predictive Control, Empirical Bayesian Method

1. INTRODUCTION

Software testing is one of the most important methods to guarantee and improve software Reliability. In the traditional opinion, the main aim of software testing is not to prove software is Correct, but to detect software defects.
From this point, software test cases should be chosen for detecting more defects. However, the opinion that the main aim of software testing is to detect much more software defects is not universally reasonable. The reason that some researchers think the aim of software testing is to find more defects is based on the assumption that fewer defects is consistent with higher reliability. However, it seems it is not definitely true. The key problem here is software reliability is not only related to the defects distribution and is related to how to use the software as well. Frankl, P.G et al discussed the relationship between so-called debug testing and operational testing and discussed how to get delivered reliability. The purpose of debug testing is directed at finding as many bugs as possible. On the other hand, the purpose of operational testing is to evaluate reliability under the assumption that the software is subjected to the same statistical distribution of inputs that is expected in operation. They pointed out debug testing may be more effective at finding bugs (provided the intuitions that drive it are realistic), but if it uncovers many failures that occur with negligible rates during actual operation, it will waste test and repair efforts without appreciably improving the software. And then they pointed out operational testing, on the other hand, will naturally tend to uncover earlier those failures that are most likely in actual operation, thus directing efforts at fixing the most important bugs. However, sometime we cannot get good results only by operational testing. We give a simple example to explain it. We assume that the input domain of a program is divided into two subdomains with same size. Furthermore, we assume that one has 1000 failure-causing inputs and another only has 10 failure-causing inputs, and actual users use the first input subdomain with probability 51% and use the second input subdomain with probability 49%. If we assume that all failures cause equal effects, a question arises: is it a good way to get higher reliability by using operational testing? It seems it is not definitely true either. Therefore, neither debug testing nor operational testing can guarantee to get higher reliability. In order to get higher reliability, we should design software testing strategy with a clear quantified measure: software reliability.

Model predictive control provides a good framework to improve predictive effect based on the above philosophy. Model Predictive Control (MPC) is widely adopted in industry as an effective means to deal with large multivariable constrained control problems. The main idea of MPC is to choose the control action by repeatedly solving an optimal control problem on line. Therefore, it is reasonable to put those software testing processes which are with some goals into the model predictive control framework.

One of the main issues in MPC is the type of model of the system under control. In general, the most utilized types of models are deterministic, stochastic, or fuzzy. Adaptive testing is a software testing technique which results from the application of feedback and adaptive control principles in software testing. It is a form of adaptive control or can be treated as the software testing counterpart of adaptive control. In adaptive testing, the software testing proceed is divided into many testing steps. As software testing proceeds, the understanding of the software under test and the test suite is improved step by step. Some previous works reveal that adaptive testing can be used to detect defects and can be used to assess software reliability. However, the models used in adaptive testing are mostly to predict the detect defects or to assess reliability without defects removed rather than to predict reliability improvement.

This paper we proposed Empirical Bayesian method based software testing strategy in criterion of reliability improvement. This paper is organized as follows. Section 2 presents the
2. BACKGROUND

2.1 Operational Profile and Testing profile

Musa said „a profile is simply a set of disjoint (only one can occur at a time) alternatives with the probability that each will occur“. An operational profile simply consists of the set of all operations that a system is designed to perform and their probabilities of occurrence. It provides a quantitative characterization of how the system will be used in the field, making it an essential ingredient of software reliability engineering. Technically speaking, one can think of an operational profile as a generic random variable that mainly indicates the operations that will be performed.

On the other hand, testing profile is to describe how to test the software while operational profile is to describe how to use the software.

2.2 Software Run

Many software reliability models are based on the assumption that software reliability behavior can be measured in terms of calendar time, clock time or CPU execution time. Although this assumption is appropriate for a wide scope of systems, there are many systems, which depart from this assumption. For example, the reliability behavior of a bank transaction processing software system should be measured in terms of how many transactions are successful, rather than of how long the software system operates without failure.

Cai proposed the conception of “a run” to describe discrete time. A run is minimum execution unit of software. Any software execution process can be divided into a series of runs.

3. SOFTWARE TESTING AS A MODEL PREDICTIVE PROBLEM

The general software testing process is to use some testing strategy to generate test cases, and then to test the software by using these generated test cases. Sometimes these test cases are generated totally in one time, and sometimes these test cases are generated one by one or one batch by one batch. In the latter, once the test case (or one batch of test cases) is/are generated, the software will be tested by using this test case or this batch of test cases, and then see what will happen. The strategy of selecting next test case or next batch of test cases will be improved according to the test results of the previous step.

Model predictive control [3] provides a good framework to describe this kind of process. Software testing process can be put into model predictive control framework. In this paper, goals are reliability improvement and other constraints, such as test cost. In the decision making process, we determinate the optimal testing profiles based on reliability improvement predictive and select test cases according the corresponding testing profiles. We choose Empirical Bayesian method as predictive model in this paper. So, the software testing process we followed in this paper is shown in Figure 1, where r means inputs, Z means test results, and A can be a variable which describe how to test the software. For example, A=i means the ith test case is selected or the test case is sleeted from the ith equivalence class.
4. MODEL PREDICTIVE CONTROL BASED SOFTWARE TESTING WITH EMPIRICAL BAYESIAN METHOD

In Section 3, a model predictive control framework based software testing process with Empirical Bayesian method was introduced. In this section, we will discuss the details.

4.1 Empirical Bayesian Method Based Decision Making
Let \( A_i \) (\( i = 1, 2, \ldots, n \)) denote the action in the ith run. \( A_i = j \) represent the test case is Selected in the jth subdomain at the ith run.

Let \( q_{ij} \) denote testing profile at \( i \)th run.

Let \( q_i = [q_{ij}^{(1)} \mid q_{ij}^{(m)} = \text{pr} \{ A_i = j \} , 1 \leq j \leq m \} \)

1 if a failure occurs at the ith run
0 if no failure occurs at the ith run

\( \theta_j^{(1)} \) (\( j = 1, 2, \ldots, m \)) denote the failure rate of \( q_j \) at the beginning of the Testing.

\( \theta_j^{(i)} \) denote the failure rate of \( q_j \) after the (i-1)th run.

\( R_i \) denote the software reliability after the total i runs.

\[
R_i = \sum_{j=1}^{m} p_j (1 - \theta_j^{(i)})
\]
4.2 The Algorithm of Model Predictive Control Based Software Testing with Empirical Bayesian Method

Therefore, the proposed model predictive control based software testing with Bayesian method (STEBM) is as follows:

1. Given the prior distributions of \( \theta_j^{(0)} \) (j=1, 2, ..., m), operational profile \( p_j \) (j = 1, 2, ..., m), the decrement of failure rate \( d_j \) (j = 1, 2, ..., m), reliability Goal and the maximum test steps n.
2. i = 1.
3. Let \( Q_i \) \( \in \{ Q_t \mid \) according to the testing profile \( Q_t \), \( R_t \) will get its maximum Prediction value}. Then choose control action \( A_t \) according to the testing Profile \( Q_i \). To run the software, and we will get the test result \( Z_{ik} \).
4. Calculate the posterior distributions of \( \theta_j^{(i)} \), after the \( Z_{ik} \) (1 \( \leq k \leq i \)) are known. This is so called the feedback step in MPC.
5. Estimate the reliability \( R_t = \sum_{j=1}^{m} p_j \left( 1 - \theta_j^{(i)} \right) \).
6. If the reliability reaches the given reliability goal or if i \( \geq n \), go to (9); otherwise go to (7).
7. Calculate the prior distributions of \( \theta_j^{(i+1)} \). If \( Z_t = 0 \), the prior distributions of \( \theta_j^{(i+1)} \) just are the posterior distributions of \( \theta_j^{(i)} \). If \( Z_t = 1 \), the prior distributions of \( \theta_j^{(i+1)} \) can be calculated based on the formula \( \theta_j^{(i+1)} = \theta_j^{(i)} - d_j \).
8. i = i+1, go to (3).
9. End.

5. CASE STUDY

The subject software in this case study is known as Space program. Rothermel et al. [18] describe the program as follows: “Space consists of 9,564 lines of C code (6,218 executable), and functions as an interpreter for an array definition language (ADL). The program reads a file that contains several ADL statements, and checks the contents of the file for adherence to the ADL grammar and to specific consistency rules. If the ADL file is correct, Space outputs an array data file containing a list of array elements, positions, and excitations; otherwise, the program outputs an error message”. The purpose of this case study is to show the difference between using the software testing with Empirical Bayesian method (STEBM) and random testing (RT) in the context of reliability.

According the function of the program we discussed, the program’s input domain was divided into four subdomains.

In this study we assume the prior parameters as follows:

\[
\alpha_{11} = 1, \ \beta_{11} = 1; \ \alpha_{12} = 1, \ \beta_{12} = 1,
\]

\[
\alpha_{13} = 1, \ \beta_{13} = 1; \ \alpha_{14} = 1, \ \beta_{14} = 1.
\]
We used two methods to test the software respectively: the software testing with Bayesian method (STEBM), random testing (RT). We did experiments as follows:

Sept 1. We tested the software by the method STEBM. The failure-causing defects were removed when detected. We denoted this defect-removed software as Software 1.

Sept 2. We tested the software by the method RT. The failure-causing defects were removed when detected. We denoted this defect-removed software as Software 2.

Sept 3. We run the Software 1 according the real operational profile. The failure-causing defect was not removed when it was detected. The reliability was estimated.

Sept 4. We run the Software 2 according the real operational profile. The failure-causing defect was not removed when it was detected. The reliability was estimated.

5.1 Experiment 1

We used two methods to test the software respectively: the software testing with Empirical Bayesian method (STBM) with the assumption that the operational profile is (0.202 0.315 0.165 0.318), random testing (RT) with testing profile (0.202 0.315 0.165 0.318). We run the program 1000 times. Figure 2 shows the comparison between the two methods about the time to detect defects.

![Fig 2: STEBM vs. RT of Experiment 1](image)

Table 1 shows the numbers of detected defects of the two methods and the final reliabilities.

<table>
<thead>
<tr>
<th>Testing Strategies</th>
<th>Detected Defects</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEBM</td>
<td>15</td>
<td>0.9970</td>
</tr>
<tr>
<td>RT</td>
<td>14</td>
<td>0.9942</td>
</tr>
</tbody>
</table>

In this case, the operational profile is consistent with defects distribution and RT use a very powerful testing profile (it is consistent with defects distribution). From the results of this
experiment, we find that STEBM can find more defects than RT. Further more, STEBM can get higher reliability than RT.

5.2 Experiment 2

We use two methods to test the software respectively: the software testing with Empirical Bayesian method (STEBM) with the assumption that the operational profile is (0.385 0.1 0.415 0.1), random testing (RT) with testing profile (0.202 0.315 0.165 0.318). We run the program 1000 times. Figure 3 shows the comparison between the two methods about the time to detect defects.

![Fig 3: STEBM vs. RT of Experiment 2](image)

Table 2 shows the numbers of detected defects of the two methods and the final reliabilities.

<table>
<thead>
<tr>
<th>Testing Strategy</th>
<th>Detected Defects</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEBM</td>
<td>15</td>
<td>0.9983</td>
</tr>
<tr>
<td>RT</td>
<td>15</td>
<td>0.9961</td>
</tr>
</tbody>
</table>

In this case, the operational profile is not consistent with defects distribution and RT is with testing profile which is consistent with defects distribution. From the results of this experiment, we find that STEBM can not find more defects. In fact, in the most time of whole testing process, RT can find much more defects than STEBM. However, STEBM can get higher reliability than RT.

5.3 Experiment 3

We use two methods to test the software respectively: the software testing with Empirical Bayesian method (STEBM) with the assumption that the operational profile is (0.385 0.1 0.415 0.1), random testing (RT) with testing profile (0.385 0.1 0.415 0.1). We run the program 1000 times. Figure 4 shows the how fast the two methods to find defects.
Table 3 shows the numbers of detected defects of the two methods and the final reliabilities.

Table 3: Total Detected Defects and Reliability For Experiment 3

<table>
<thead>
<tr>
<th>Testing Strategies</th>
<th>Detected Defects</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEBM</td>
<td>15</td>
<td>0.9979</td>
</tr>
<tr>
<td>RT</td>
<td>15</td>
<td>0.9956</td>
</tr>
</tbody>
</table>

In this case, the operational profile is not consistent with defects distribution and RT has testing profile which is not consistent with defects distribution. From the results of this experiment, we find that STEBM can find as many defects as RT can and can get higher reliability than RT.

6. CONCLUSION

The aim of this paper is to develop a reliability improvement predictive approach to software testing with Empirical Bayesian Method. The interesting result of this paper is that it is not definitely true for a software testing method that can find more defects than others can get higher reliability. Because the proposed software testing in the paper can combine software testing with reliability prediction together and consider the operational profile as well, it really can get good result in the sense of improving reliability.

The initial values of related parameters play a key role in software testing, but it is difficult to get the certain values of them. Empirical Bayesian method overcomes these difficulties, but requires the users to develop prior distributions. There exist several ways to do this, such as the method based on expert opinion. The expert opinion includes the empirical experience of other similar software, the knowledge of software testing, or other background information.
REFERENCES


Authors

D. Vivekananda Reddy, working as Assistant Professor in the Dept of Computer science and Engineering, S V University college of Engineering, Tirupati, per past seven years and doing research in the field of software testing.