
Zhi-Yao Deng and Nan Sang

Abstract—Safety-critical system (SCS) has highly demand for dependability, which requires plenty of resource to ensure that the system under test (SUT) satisfies the dependability requirement. In this paper, a new SCS rapid testing method is proposed to improve SCS adaptive dependability testing. The result of each test execution is saved in calculation memory unit and evaluated as an algorithm model. Then the least quantity of scenario test case for next test execution will be calculated according to the promised SUT’s confidence level. The feedback data are generated to weight controller as the guideline for the further testing. Finally, a comprehensive experiment study demonstrates that this adaptive testing method can really work in practice. This rapid testing method, testing result statistics-based adaptive control, makes the SCS dependability testing much more effective.

Index Terms—Adaptive, feedback, safety-critical system, statistics, test.

1. Introduction

As microprocessors are developed speedily, more and more safety-critical system is applied and adapted to many safety-critical industries, such as aviation, defense, nuclear power management, medical system [1] and so on. In these applications, any unexpected failure would be disastrous, so to insure the high-dependability of system under test (SUT) is important.

A safety-critical system (SCS) usually has complex system requirement, architecture, safety-critical tasks, and even the modification of requirement during development. For the test of SCS, not only the functions in requirement are needed to test, but also it needs to ensure the reliability and safety performance to achieve the confidence level promised, all of these make the dependability testing become harder [2]. Roughly speaking, conventional strategies for software testing include random testing and partition testing [3]. In partition testing, the testing goal may be to cover all statements, control flows, data flows or critical paths. All the test cases in the generated test suite are executed [3]. In random testing, the testing goal may be to perform the given number of tests in accordance with a given probability distribution or stochastic process, and the generated test suite is simply the whole input domain of the software under test or the state space of interest. The test cases are selectively executed [3].

The dependability of SCS focuses on keeping away the certain failures with high risk [3]. If the failure rate is considered as a constant in partition testing, which means that all the system test cases will be executed afresh without any statistic and feedback strategy after the bug fixed, some “lucky case” with high risk has more possibility to accidentally pass the testing while the executing environment changing [2]. For the random testing in practice, test cases, which are selected and executed against the SUT, may or may not reveal software failures [3].

Based on the analysis above, a new testing method was proposed. It focuses on that the test cases in the generated test suite are exhaustively or selectively executed. The testing data are collected and analyzed on-line during testing so that the current status of software testing process and the software under test are continually updated for the adaptive control of software testing process. In details, the new testing method makes use of one memory unit to dynamically adjust the weight of test scenario, then calculates the least quantity of test cases for the next test execution. In this way, the weight of system testing can be on-line adjusted. This testing method makes use of the limited time do more test work in effect. This adaptive software testing method specifies what next testing policy should be employed and thus in turn what test suite or next test cases should be generated or selected in accordance with the promised confidence level of SUT.

This paper is organized as follows. Some theoretical backgrounds are introduced in Section 2. Section 3 presents a new adaptive SCS dependability testing method. An application example of the testing method is given in Section 4. Finally, Section 5 concludes this paper.

2. Relative Work

Basically, verification pattern (VP) is a new technique to test embedded systems rapidly, and it has been used to test industrial safety-critical embedded systems successfully.
Moreover, an appropriate algorithm for evaluating the test result is the important requisite. The feedback strategy can make them work well in this method.

2.1 Test Case Generation

The principal idea of verification pattern [4] is to classify system scenarios into patterns, and use the same code template to test all the scenarios in the same verification pattern [4][7].

For many industrial applications we encounter, especially applications to real-time, safety-critical medical devices [1], large complex systems needed only a small number of scenario patterns (SPs), even for complex industrial systems with hundreds of thousands of scenarios [5]. For example, eight scenarios patterns are sufficient to cover 95% of system scenarios for an industrial safety-critical implantable medical device [1]. Another significant advantage of using VPs is that the test cases developed are much more consistently with each other, and thus reducing the debugging effort.

2.2 Testing Result Evaluation

Bayesian functions can effectively figure the prior probability distribution and posterior probability distribution. This method makes use of Bayesian functions to figure out the system failure probability density [8].

Assume the system failure probability density as \( p \); every testing experiment would satisfy the independence of Bernoulli. The probability of \( R \) time’s failure in \( n \) time’s testing will obey binomial distribution, namely \( P(R = r) = C_r^p (1 - p)^{n-r} \) [8]. This shows that the failure probability \( p \) will affect the SCS dependability estimate. The failure probability density function (using Bayesian) is:

\[
f(p) = \frac{p^{a-1}(1-p)^{b-1}}{B(a, b)}
\]

where \( B(a, b) = \int_0^1 p^{a-1}(1-p)^{b-1} \, dp, a > 0, b > 0 \).

\[
B(a, b) = \int_0^1 p^{a-1}(1-p)^{b-1} \, dp, a > 0, b > 0 .
\]

\( B(a, b) \) shows the tester’s subjective prior estimate [9]. Mostly, testers can not get this value since the SUT is not verified before [9]. In this case without prior probability verification, \( a = b = 1 \) in (2) is generally considered, and then one has

\[
f(p) = \frac{p^{1-1}(1-p)^{1-1}}{B(1,1)} = 1 .
\]

Based on this consideration, the posterior probability of \( R \) time’s failure in \( n \) time’s testing distributes as Beta \( (a + r, b + n - r) \), and namely

\[
f(p | r, n, 1, 1) = \frac{p^{r-1}(1-p)^{n-r}}{B(r+1, n-r)} .
\]

Assume the user acquires \( p_0 \) as the SCS failure probability, \( C \) as confidence level, and then it expresses \( P(p > p_0) \geq C \).

When there is no failure occurred during testing, set \( R = 0, n_t \) test cases are needed to ensure this system satisfies the user requirement \( p_0 \) and \( C \). Hereinto, the least cases number \( n_t \) is the minimal \( n \) in expression [10]:

\[
P(p > p_0) = \int_0^{p_0} f(p | n, 0, 1, 1) \, dp = \int_0^{p_0} \frac{(1-p)^n}{B(1,1+n)} \, dp \geq C
\]

and then we can estimate the failure probability density of \( R_f \) failures in \( N_f \) test cases execution is [10]

\[
P(R_f = r_f | r, n, a, b) = \int_0^1 P(R_f = r_f | p) f(p | r, n, a, b) \, dp
\]

\[
= \int_0^1 C_{r_f}^n p^{r_f}(1-p)^{n-r_f} p^{2+n-r_f}(1-p)^{1+n-r_f} B(a+r, b+n-r) \, dp \geq C
\]

where \( R_f = 0 \) means no failure, one has

\[
\int_0^1 (1-p)^{N-f} p^{f} (1-p)^{n-r} B(1+r, 1+n-r) \, dp \geq C
\]

This basic algorithm guides the calculation unit to make estimate for each test execution result. The expression (4) can get the least quantity of test cases for next testing and the expression (5) can verify if the case quantity is sufficient according to the SUT’s promised confidence level.

3. New Method with Calculation and Increment Memory Unit

A new rapid testing method is designed by making use of memory unit to record and evaluate the last testing result. And then the memory unit sends out feedback data to guide the further testing. The new method is suitable for the SCS dependability test and has greater efficiency without introducing any excessive complexity. The integrated workflow of the new method is shown in Fig. 1.

![Fig. 1. Test method work flow.](image-url)
1) Derive the scenarios specification from the requirement: Each scenario in an SCS development usually has preconditions (or causes), post-conditions (or effects), and optional timing constraints\cite{4,6}.

2) Map scenario patterns to VPs: define a verification pattern as a predefined mechanism that can verify a group of behavioral requirements that describe similar temporal pattern or cause and effect relations\cite{10}.

3) Verification pattern specifies the verification mechanism and the scenario pattern configuration (ACDATE model\cite{11}): each atomic scenario is represented as a collection of Actor, Condition, Data, Action, Timing and Event\cite{5}, then the verification code-test cases are generated.

4) Test the implementation using the test cases.

The feedback strategy (dashed line in Fig. 1) achieves on-line adjustment for the SCS dependability testing. It is generated from the increment memory unit in which test execution results are evaluated. And then send the result back to weight controller as the guideline of the next test cases generation.

The calculation process of estimate in increment memory is as follows:

In case of \((p_0, C) = (10^{-4}, 0.99)\), the minimal \(n\) value \(n_i\) is 46047 by expression (4).

Now the increment memory got the statistic data of test result: the \(s_1\) \((s_1 < n_i)\) test case caused system failure in this testing. Except the original test cases, how many increment for new test cases needed to ensure the target system satisfies the safety requirement \((p_0\) and \(C)\) in the next test execution. Now assume \(n_2\) new test cases needed in second testing, the prior probability distribution is

\[
f(p|1,s_1+n_2,1,1) = \frac{p(1-p)^{s_1+n_2-1}}{B(2,s_1+n_2)}
\]

So the \(n_2\) is the minimal \(n\) in expression\cite{10,12}:

\[
\int_0^1 (1-p)^n \frac{p(1-p)^{s_1+n-1}}{B(2,s_1+n)} dp \leq C
\]

In this way, if \(n_2\) test cases passed in the second testing, it is sufficient to prove the target system satisfies the safety requirement; but if the \(s_2\) \((s_2 < n_i)\) test case caused failure, the third testing with \(n_3\) test cases more will process in the same way. In a word, if the testing occurred \(j\) time’s failure, and the failure respectively occurred in such test cases: \(s_1\), \(s_1+s_2\), \(s_1+s_2+s_3\), \(\cdots\), \(s_1+s_2+\cdots+s_j\). The next increment for test cases \(n_{j+1}\) is the minimal \(n\) in expression:

\[
\int_0^1 (1-p)^n \frac{p(1-p)^{\sum_{i=1}^{j}s_i+n-1}}{B(\sum_{i=1}^{j+1}s_i+n-j+1)} dp \leq C
\]

In this way, the calculation memory unit and the weight controller adaptively adjust the testing weight of each scenario on-line.

### 4. Experiment

This paper was supported by National 863 Program. Table 1 shows the distribution of scenario patterns in this project, which shows that a large portion (around 45%) of system behaviors belong to the basic requirement pattern, with key-event pattern the next popular (about 15%). It is interesting to observe that these two alone cover 60% of the system requirements. In other words, the cost and effort reduction by using this approach to generate test cases can be huge.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Coverage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic requirement pattern</td>
<td>45</td>
</tr>
<tr>
<td>Key-event driven requirement pattern</td>
<td>15</td>
</tr>
<tr>
<td>Timed key-event driven</td>
<td>5</td>
</tr>
<tr>
<td>Requirement pattern</td>
<td>7</td>
</tr>
<tr>
<td>Key-event driven time-sliced</td>
<td>7</td>
</tr>
<tr>
<td>Requirement pattern</td>
<td>7</td>
</tr>
<tr>
<td>Total</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 2 provides some evidence to show the feedback data calculated from the memory unit.

### Table 2: Testing weight calculation by failure times

<table>
<thead>
<tr>
<th>Failure occurred times</th>
<th>Testing weight(test case quantity)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>46047</td>
</tr>
<tr>
<td>1</td>
<td>92383</td>
</tr>
<tr>
<td>2</td>
<td>138691</td>
</tr>
<tr>
<td>3</td>
<td>184998</td>
</tr>
<tr>
<td>4</td>
<td>231306</td>
</tr>
<tr>
<td>5</td>
<td>…</td>
</tr>
</tbody>
</table>

Considering the extreme condition that \(s_1=1\) and \(s_i=46047\), with the \(n_2=92383\) and \(n_2=46337\), the basic conclusion is that the earlier failure occurred, the worse expectation the system have in the further testing, the better expectation with the fewer testing cases conversely.

In the conventional way, it usually just assigns the average weight to each scenario, without any concern about the scenario testing history before\cite{3}. In this way, the scenarios with high risk have more possibility to pass the test when the execution environment keeps changing\cite{13}. In comparison, the new adaptive testing method is better than the conventional one by adaptively controlling the testing.
weight of each scenario based on the testing result statistics.

In summary, the experiment proved this new method useful in practice, and this new testing method makes the dependability testing become more efficient.

5. Conclusions

This paper suggests that software testing strategy should be adjusted on-line by using the testing data collected during software testing as our understanding of the SUT is improved. The analysis of the special demand of dependability test in SCS engineering shows that the conventional testing framework should be improved. The integrated testing method in the working flow is explained. Based on the relative theoretical backgrounds on pattern-based testing and Bayesian estimate algorithm, the testing weights of all system scenarios are online adjusted to insure the SUT’s confidence level. This circumvents the drawbacks in conventional random testing and partition testing. At the end, this paper provides some detail data in the entire SCS dependability testing process (National 863 Program) to prove this new method useful in practice.

References


Zhi-Yao Deng was born in Sichuan Province, China, in 1983. He received the B.S. degree from the University of Electronic Science and Technology of China (UESTC), Chengdu, in 2005. He is currently pursuing the M.Sc. degree with Department of Software Engineering, UESTC. His research interests include software reliability engineering, and information theory.

Nan Sang was born in Sichuan Province, China, in 1964. He received the M.S. degree from the Sichuan University, Chengdu, in 1987. He is currently a professor with UESTC. His main research interests include embedded real time dependable computing, embedded development platform technology, and embedded system application.