Selecting Software Testing Criterion based on Complexity Measurement

Wen C. Pai, Chun-Chia Wang and Ding-Rong Jiang

Department of Information Management
Kuang Wu Institute of Technology,
PeiTou, Taipei 112, Taiwan, R.O.C.
E-mail : wencpai@ms1.hinet.net

Abstract

Software testing is an important process in software quality assurance(SQA). Software testing consumes the majority of the software developers' time and cost of all the phases of system development. An important problem in software testing is to determine whether a program has been tested enough with a testing criterion and when one could stop testing. On the other hand, software measurement is used to estimate the quality of a software. Complexity measures can be used to compare software quality and development cost. This paper proposes a new testing approach which choosing testing path based on path complexity. The process is to use a set of testing paths that satisfying all-uses testing criterion, and then choose the testing path with the most complexity to test. This process is continued until the program can be accepted. The proposed formula is used to determine if it is possible to stop testing before satisfying all-du-paths testing criterion. The main contribution of the paper is that software developers can choose a proper testing action with lower testing effort.

Key words: Software Quality Assurance, Software Testing, White-Box Testing, Path Testing, Software Complexity

1. Introduction

Software testing is an important process in software quality assurance(SQA). Software testing consumes the majority of the software developers' effort of all the phases of system development. An important problem in software testing is to determine whether a program has been tested enough with a testing criterion and when one could stop testing.

Existing software testing methodologies are divided into two categories: static testing and dynamic testing. In static testing, the requirements documents, design documents and the code are analysed, either manually or automatically, without actually executing the code. In dynamic testing, the requirements documents, design documents and the program is actually executed in a controlled and systematic way to demonstrate that required functions are present.

Functional testing and structural testing are two major approaches in dynamic testing method. The functional testing is also called black-box testing; the tester is completely unknown about the internal behavior and structure of the program. Rather, the tester is only interested in finding circumstances in which the program does not behave according to its specifications. Test data are derived solely from the specifications. In structural testing, also called white-box testing, the tester derives test data from an examination of program logic, internal control flow and data-dependency.

The methodologies of structural software testing are divided two major categories: control flow oriented testing and data flow oriented testing. Control-flow testing is based on the control graph of the program and consider how to select the testing path for discovering more error. On the other hand, data-flow testing derives test data by
considering how a data is defined and how the data is used in a program.

Figure 1. The subsume relation of testing criteria

Path testing is one of the white-box testing technologies. Via test cases, one intends to execute all possible paths of control flow through the program, then possibly the program can be tested completely.

A number of papers presented a family of testing path selection criteria [4] [6] [8] and show the subsume relation. The relation is showed in Figure 1, where criterion $c_1$ is said subsume criterion $c_2$ if for every program, any set of complete paths of the program satisfies $c_1$ is also satisfies $c_2$. For example, a testing path set that satisfying all-du-paths criterion in testing a program is also satisfying all-uses criterion. If the subsume criterion be selected, the "more" effort have to make; subsequently, the testing cost is higher. On the other hand, if the subsumed criterion be selected, the software may be lower assurance of correctness with lower testing effort.

The shortcoming of the relation is that a subsume criterion did not guarantee better fault detecting ability than a subsumed criterion. Frankl and Weyuker investigated fault detecting ability between some software testing criteria and found that the subsume relation did not guarantee better fault detecting ability[2].

We had proposed a testing methodology which considering a subsumed criterion to make more efficient testing than the subsume criterion by improving the subsumed testing criterion[5]. We proposed the testing criterion by improving all-uses criterion. The basic idea is to use a base set of testing path that satisfying all-uses criterion, and then add new testing path, one at a time, to cover the remaining untested paths. The proposed formula is used to determine if it is possible to stop testing before satisfying all-du-paths criterion. The testing paths are added by randomly choosing in the methodology.

On the other hand, software measurement could be used to indicate the quality of the software. If we view a software system as a production system that produces the desired software, we can use the statistical sampling procedure and statistical inference to predict the reliability of the software. Cho[1] describes a testing methodology based on statistical quality control procedure. The test cases as the raw material and the produce is the software system's output. In this paper, we still use the formula to determine if it is possible to stop testing before satisfying all-du-paths criterion. However, the testing paths are choosed based on the path size instead of random choice in the new methodology.

In the next section, a review of path testing is presented. Section 3 proposes the new testing methodology, where will present the testing paths are added based on the path size. Section 4 rises an example to compare the testing effort between different testing criteria. The algorithm of the proposed methodology and conclusion are given in the last section.

2. A Review of Path Testing

A number of terminologies of path testing technologies will be reviewed in this section. A variable-definition is a statement that a value of a variable $x$ is assigned or modified (denoted by $\text{def}(x)$). An edge is called a $p$-use of variable $x$, denoted by $\text{pu}(x)$, if there is a predicate-use of $x$ in this edge. A node is a $c$-use of $x$, denoted by $\text{cu}(x)$, if there is a computation-use of $x$ in this node. A path is a $du$-path if the start node of the path is a variable-definition of variable $x$ and the path reaches an edge of $p$-use of $x$ or a node of $c$-use of $x$.

Each criterion in Figure 1 has its rule to derive a set of testing paths to conduct a test activity. The definition of all-uses criterion and all-du-paths criterion are presented in the following.

- A set of complete path is said to satisfy all-du-paths criterion (denoted by ADUP) if for every node $i$ and every variable $x$ defined in node $i$ (denoted by $\text{def}(x,i)$), this testing path set includes every $du$-path w.r.t. $x$.
- The testing path set satisfies all-uses criterion (denoted by AU) iff the set includes every def-clear path w.r.t. $x$ from $i$ to every node with $c$-use of $x$ and
to every node with p-use of x.

In a program graph, if there are more than one testing path from a specific def(x) to some pu(x) or some cu(x), a path set satisfying all-du-paths criterion must test every path from def(x) to every pu(x) or every cu(x). On the other hand, a path set satisfying all-uses criterion could only test some paths from def(x) to every pu(x) or every cu(x). This gives a measurement basis of different fault detecting ability between all-uses criterion and all-du-paths criterion.

We will present a measurement of different testing effort between all-uses criterion and all-du-paths criterion. The measurement result will be used to present a testing approach to determine if it is possible to stop testing before satisfying all-du-paths criterion. The testing approach can reduce the testing effort, or say cost, for software testers to test software.

3. The Testing Approach

Since subsume relation did not guarantee better fault detecting ability, some of the testing paths conducted for satisfying all-du-paths criterion may be redundant. The testing action is proposed to determine if it is possible to stop testing before all-du-paths be tested such that we can reduce the time and cost. The process is to use a base set of testing paths that satisfying all-uses criterion, and then add new testing path based on path size, one at a time, to cover the remaining untested paths. The proposed formula is continued until the program can be accepted.

The remaining question is how to decide the number of paths to be tested. Since exhaustive testing is impossible, we compare the different testing effort based on a subset of testing paths, which is defined in the following.

**Definition 3.1 PS(testing criterion)**

A path set, which contains the minimum number of paths such that each path traverse minimum number of nodes, satisfying the testing criterion is denoted by PS(testing criterion).

For example, PS(AU) represents the path set, which contains the minimum number of paths such that each path contains minimum number of nodes, satisfying all-uses criterion. Every path includes a loop statement such as while-loop will be traversed only once.

**Definition 3.2 N(testing criterion)**

The number of paths of a testing path set which satisfying the testing criterion is denoted by N(testing criterion).

In a program graph, consider a variable definition of x at node i, and a computation-use of x at node j or a predicate-use of x at edge (j,k). If there is any branch between node i and node j, for example: if-then statement, then the testing paths start from the node i should branch after node i and join at some node before node j. A testing action satisfying all-du-paths criterion must test all the paths. On the other hand, a testing action satisfying all-uses criterion could only test some paths of them. The number of testing paths conducted to satisfy all-du-paths criterion would more than the number of testing paths conducted to satisfy all-uses criterion. The difference of the number of testing path gives the basis of comparing different testing effort between all-du-paths criterion and all-uses criterion.

When a testing path set that satisfying all-uses criterion, there are some testing paths can be added to the path set for satisfying all-du-paths criterion. We will rise some definitions to explain the result.

**Definition 3.3 un-traversed path set (UTP)**

A testing path set P={p_1, p_2, ..., p_k} satisfying all-uses criterion. A testing path set P'={p_1, p_2, ..., p_n}, n ≥ k, satisfying all-du-paths criterion. The difference of P' and P are call un-traversed path set, denoted by UTP. i.e., UTP=P'\P={p_i \in P' : p_i \notin P}

**Definition 3.4 Untested set (UTS)**

The part of UTP to be added to test is called tested set, the remaining part of UTP that still not to be tested is called untested set, denoted by UTS. Since the error may occur relatively in untested set with "higher probability", the more the number of paths in tested set, the lower the probability of error-occurring. If some paths of UTP be tested, then the remaining paths in UTS may occur error with "higher probability". Although we can test all paths in UTP, it may not be necessary to do that. Given a level of tolerance, one may decide whether a program has been tested enough.

The proposed testing technology can be explained by Figure 2. When choosing a path of UTP or UTS to test, we consider the path-size of the path. Software measurement is an important process to assess software quality. The more complexity of a software, the higher error-occurring probability of the software. Line of code (LOC) is one of the software measurement method[3]. Since LOC is a common size measure and easy-to-use, we use LOC to measure the size of a path.

After establishing a LOC counting standard,
we can count the LOC size of the testing paths in UTS. The paths choosing from UTS based on LOC size. When a path with the largest LOC size among the paths of UTS, the path to be tested first.

Fig. 2  The testing approach

Assume all paths in UTP are equally and independently likely to be associated with an error. If the number of paths in UTP is m and the number of paths in UTS is n, then the probability of error-occurring in one specific path of UTP is 1/m and the probability of error-occurring not in the specific path is (m-1)/m. Error occur in some path of UTS may be:

\[
P = \sum_{i=1}^{n} \left( \frac{1}{m} \right)^i \left( \frac{m-1}{m} \right)^{n-i} = \frac{1}{n} \left( \frac{m-1}{m} \right)^{n-1}
\]

We can test the paths in UTP and compute P step by step. Given a level of tolerance, a, if P is less than a, we say the test action that only testing some of the paths in UTP can be accepted. The testing action with testing these paths is adopted.

4. A Comparison Between Different Testing Criteria

For example, in Figure 3, there is a def(x) and a def(y) in node 1, one pu(y) in edge (1,2) and one pu(y) in edge (1,9). One cu(x) in node 6 and another in node 7. A cu(y) in node 11, a def(y) in node 9, a pu(x) in edge (5,6) and a pu(x) in edge (5,7).

Figure 3. An example of program graph

<table>
<thead>
<tr>
<th>(1) all-uses criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>test all paths of UTP</td>
</tr>
<tr>
<td>(i.e., all-du-path criterion)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(2) the new testing approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>test another one path of UTP</td>
</tr>
<tr>
<td>test all paths of UTP</td>
</tr>
</tbody>
</table>

Similarly, If two of UTP be tested, then n=m-2, such that the probability of error occurring in UTS can be represented by

\[
P = \sum_{i=1}^{n} \left( \frac{1}{m} \right)^i \left( \frac{m-1}{m} \right)^{n-i} = \frac{1}{n} \left( \frac{m-1}{m} \right)^{n-1}
\]

i.e., N(all-uses criterion)=2
Since \( m = 4 \), we can assume that error occur in each of \((1,2,4,5,6), (1,2,4,5,7), (1,2,5,6), (1,2,5,7)\) is equally likely and independent. For example, the probability of error occur in \((1,2,4,5,6)\) is \(1/4\), not occur in \((1,2,4,5,6)\) is \(3/4\). Path \((1,2,4,5,7), (1,2,5,6), (1,2,5,7)\) are similar to path \((1,2,4,5,6)\).

**Step 1**
By counting LOC size, if path \((1,2,4,5,6)\) to be tested, then
\[
N(\text{new testing approach})=1
\]
Since only one of UTP to be tested, then the number of paths in UTS is 3, i.e., \(n=3\). The probability of error-occurring in UTS can be represented by
\[
P = 3C1 (1/4)^1 (3/4)^2 + 3C2 (1/4)^2 (3/4)^1 + 3C3 (1/4)^3 = 0.578
\]

**Step 2**
If \(((1,2,4,5,6), (1,2,4,5,7))\) to be tested, then
\[
N(\text{new testing approach})=2
\]
the number of paths in UTS is 2, i.e., \(n=2\), thus
\[
P = 2C1 (1/4)^1 (3/4)^1 + 2C2 (1/4)^2 = 0.4375
\]

**Step 3**
If \(((1,2,4,5,6), (1,2,4,5,7), (1,2,5,6))\) to be tested, then
\[
N(\text{new testing approach})=3
\]
the number of paths in UTS is 1, i.e., \(n=1\), thus
\[
P = 1C1 (1/4)^1 = 0.25
\]

If the level of tolerance is 0.5, then we can say the program may be accepted with only one of UTP be tested (since 0.4375 < 0.5). On the other hand, if the level of tolerance is 0.1, then the four paths of UTP must be tested; and this leads to all-du-paths criterion.

**3) all-du-paths criterion**
\[
PS(\text{all-du-paths criterion}) = \{(1,2,3,5,6), (1,2,3,5,7), (1,2,4,5,6), (1,2,4,5,7), (1,2,5,6), (1,2,5,7)\}, \text{thus}
\]
\[
N(\text{all-du-paths criterion}) = 6
\]
Since subsume relation did not guarantee better fault detecting ability, software developers can choose a proper testing action before satisfying all-du-paths testing criterion. One can derive a higher quality software with lower testing effort.

**5. Algorithm and Conclusion**
This paper proposes a testing approach which conducting test case step by step. When a testing path set satisfying all-uses criterion, we add a path, based on path complexity, to test. The proposed methodology is like a jack used for a broken vehicle. We name it jack all-uses criterion. In Figure 4, the jack all-uses criterion tests software using a base set of testing paths that satisfied all-uses testing criterion, and then add new testing path to cover the remaining untested paths. The jack all-uses subsume all-uses and be subsumed by all-du-paths criterion. The software developers can choose a proper testing action with lower testing effort. We rise the algorithm of the testing approach in the following.

In this paper, we apply software complexity to software testing. A path with higher LOC size is tested first. Some of the testing paths conducted for satisfying all-du-paths criterion may be redundant. We can reduce the testing time and cost by choosing a proper testing action.

Since LOC is not the best indicator and easy
to get confused, the other software measures may be used properly. Besides, if all paths in UTP are not equally and independently likely to be associated with an error, this leads to another model need be solved. These problems will be our future works.

References


Manuscript Received: Mar. 05, 1999
Revision Received: Apr. 07, 1999
and Accepted: Apr. 13, 1999