AN ENVIRONMENT FOR NON-DUPLICATE TEST GENERATION FOR WEB BASED APPLICATION

Manjeet Kumar\textsuperscript{1} and Dr. Rabins Porwa\textsuperscript{2}

\textsuperscript{1}Research Scholar, Mewar University, Rajasthan
\textsuperscript{2}Associate Professor, ITS, Mohan Nagar, Ghaziabad

ABSTRACT

Web applications quality, portability, consistency, interoperability and dependability are main parameters because software could block completely business. Internet based applications are very complex and heterogeneous also based on different kind of modules which can be implemented in a specific language, so testing is very much require. This paper represents the object oriented based environment. Web applications consist of Web related documents, class and objects and some different kinds of server like HTML, DHTML, and JAVA SCRIPT etc. This paper presents three kinds of layers of test: unit testing, integration testing and system testing. This paper also discusses the different techniques. This paper also based on the reverse engineering techniques which can be used to analyze the software which is probably connected with a model. It also consists a method of generate the test cases. This paper also describes software used to perform some results.

KEYWORD

Automated testing, class, object, oops

1. INTRODUCTION

The object oriented method to software development does is very much required for component based technique. If we talk about Object oriented software which is supposed to be tested with different kind of architecture and different kind of key features. The main testing issues of the object oriented software is updating in terms of functionality and also development of new testing techniques.

Class is the logical unit and abstract data type in object oriented software. Class works as encapsulation in which method and data can be encapsulated. The inheritance feature of the object oriented for reusability. Dynamic binding major feature of the object oriented approach binds the relevant method at runtime. Late binding creates indefiniteness in the testing process since the method to be executed is unknown until runtime. As opposed to the waterfall model used for traditional software development, object oriented software is developed using the iterative and incremental approach. Thus, object oriented testing too becomes iterative and incremental, requiring use of regression testing techniques.

Object oriented software is tested at the unit, integration and the system level. At unit level, class is the basic unit of testing. Different approaches like the state-based testing and data-flow testing use the black-box, white-box and gray-box testing techniques to test the class. The already tested classes interact with each other via relationships like inheritance and aggregation to form a subsystem. Integration testing tests for interface errors in the interacting units of the subsystem.
System testing tests the complete application software. Object oriented software testing is aided by testing tools. The testing tools automate the test case design and execution, and the result evaluation at different stages of testing.

The paper is divided in different sections. In section 2 we discuss in brief the features of the object oriented software. Section 3 deals with the testing issues unique to the object oriented software. Section 4 discusses the testing process used to test the object oriented software at the three levels of testing—unit, integration and system level. It includes a survey of the testing strategies used in the testing of the object oriented software. Section 5 describes briefly the testing tools developed to test the object oriented software.

Unit test tool like JUnit is very important component of software development environment. The programming environment let us take example JUnit for testing and controlled code changes. If we really look at manually testing. Manually testing equally important as in automated software testing but the main drawback is behavior of the class under unit test because manual test generation is very time consuming and developers often not include some test inputs. Different companies have different tools for test cases so we are not able to get the desired service and even frameworks do not support to the tool. If we really look at the constrained resources, existing test generation tool does not generate sufficient unit tests. As we already discussed we are wasting time on generating and running redundant tests.

![Testing web application overview](image)

Figure 1: Testing web application overview

If we have a constant set of values for method arguments and we are coordinating with non-redundant test, we required at least one new object state. Here we are talking about new state we required in order to generate non-redundant tests.
2. OVERVIEW OF BINARY SEARCH TREE

We are presenting binary search tree here Figure 4.1 shows the relevant parts of the code. The binary search tree class B1 which has a set of integers. Tree has a pointer to the root node.

```java
class B1 implements Set {
    Node r;
    static
    class N1 {
        int val;
        N1 le;
        N1 ri;
    } public void add(int val) {
        if (root == null) {
            root = new N1();
            root.val = val;
        } else {
            N1 n2;
            root = n2;
            while (1) {
                if (n2.val < val) {
                    if (n2.ri == null) {
                        n2.ri = new N1();
                        n2.ri.val = val;
                        break;
                    } else { n2 = n2.ri; }
                } else if (n2.val > val) {
                    if (n2.le == null) {
                        n2.le = new N1();
                        n2.le.val = val;
                        break;
                    } else { n2 = n2.le; }
                }
            }
        }
    }
    public void del(int val) { ... }
    public boolean has(int value) { ... }
}
```

Figure 2: binary search tree Implementation

We are presenting Jtest tool here and generating random sequences of methods. If we really look at Junit then this is a ideal solution of Unit testing.

```java
public class B2 extends T {
```
public void test1() {
    B2 tx = new B2();
    tx.add(0);
    tx.add(-1);
    tx.remove(0);
}

public void lp() {
    lp txp = new lp();
    txp.add(654321);
    txp.remove(333);
}

Figure 3: JTest Tool Implementation

We are adopting by existing tools is to explores all methods randomly to a given length. It may be possible that two tests generate same sequence.

3. CONCRETE-STATE EXPLORATION

Unit test for oops programs have two parts: object states and method arguments. The first part can be connected with. The second part introduce particular arguments for a method. The concrete-state approach presented in this section assumes a fixed set of method arguments have been provided beforehand and invoke these method arguments to explore and set up object states.

A method-argument state is characterized by a method and the values for the method arguments, where a method is represented uniquely by its defining class, name, and the entire signature. Two method-argument states are equivalent iff their methods are the same and the heaps rooted from their method arguments are equivalent. Each test execution produces several method executions.

This technique is a type of sequentional testing. Here we are presenting test generation for each possible input. In order to find generate method state our implementation consists of method arguments. This mechanism is based on huge amount of data. In this thesis we are not including
manual test cases so we need to collect method arguments by Jtest.

If we look forward for nonequivalent state then we required test inputs which consist of constructor. If we have empty state then we have to generate new tests for the same. All nonequivalent method-argument states.

4. SYMBOLIC-STATE REPRESENTATION

The symbolic execution consists of inputs in the form of symbolic variable instead of formal parameters. In OOPs, state can be generate in terms of symbol. Symbolic states differ from concrete states.

We find a symbolic heap as a graph and each node represent objects and edges represents object fields.
The usual execution of a method starts with an object and method-argument values, and then produces one return value and one concrete state of the receiver object.

4.1 Symbolic-State

In this technique I am presenting comparison of two symbolic states. These techniques also support to determine state subsumption

4.1.1 Isomorphism validation

We define heap graph based on node bijection. Here we are looking forward for isomorphic heaps. If we are really interested about it then we can get the method behavior.
This definition supports only object identities and symbolic

### 4.2 Symbolic-State Representations

In this section we are presenting Symstra approach for symbolic-state space. The state space means all symbolic states that are reachable with the symbolic execution.

Following Code is given below-

```java
Set S(Set K, Set N, int Num) {
    Set s = new Set();
    Set f = new Set();
    foreach (constructor in K) {
        RuntimeInfo rrp= symExecAndCollect(constructor);
        s.addAll(rrp.solveAndGenTests());
        fs.addAll(runtimeInfo.getNonSubsumedObjStates());
    }
}
```

![Figure 3.1: Algorithm for symbolic sequence](image)

### 4.3 Observation

Here we are presenting our evaluation for states and generating tests cases. We have performed the experiments on Window XP.

Table shows the lists of 11 Java classes that we use in the experiments. The some classes were previously used in evaluating our redundant-test detection approach.

<table>
<thead>
<tr>
<th>Class</th>
<th>Methods under test</th>
<th>Some private methods</th>
<th>Line of code</th>
<th>No of branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntStack</td>
<td>push,pop</td>
<td>-</td>
<td>46</td>
<td>11</td>
</tr>
<tr>
<td>UBStack</td>
<td>push,pop</td>
<td>-</td>
<td>63</td>
<td>18</td>
</tr>
<tr>
<td>BinSearchTree</td>
<td>add,remove</td>
<td>removeNode</td>
<td>101</td>
<td>38</td>
</tr>
<tr>
<td>BinomialHeap</td>
<td>insert,extractMin,</td>
<td>findMin,merge,</td>
<td>323</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>delete</td>
<td>unionNodes,decrease</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LinkedList</td>
<td>add,remove,removeLast,addBefore</td>
<td></td>
<td>267</td>
<td>18</td>
</tr>
<tr>
<td>TreeMap</td>
<td>put,remove</td>
<td>fixAfterIns,fixAfterDel,delEntry</td>
<td>383</td>
<td>200</td>
</tr>
<tr>
<td>HeapArray</td>
<td>insert,extractMax</td>
<td>heapifyUp,heapifyDown</td>
<td>91</td>
<td>33</td>
</tr>
</tbody>
</table>
We approach following data detection technique for the same.

5. CONCLUSION

Here we proposed Symstra, approach for non-redundant tests that contains high branch and intra-method path coverage for complex data structures. We finally discuss how this technique is useful for this research.

Specifications. The work in this thesis including the Symstra approach has been developed to be used in the absence of specifications. Still we don’t have any specification.

Performance: We already discussed about performance parameter which is non-technical attribute for the same.

REFERENCES


AUTHORS

Manjeet Kumar received MCA. degree in Computer Science from K.N.I.T Sultanpur, (2004), and M-Tech degree in IT(2010) from Karnatika University. At the moment he is Ph.D. scholar at Mewar University, Rajasthan. His research interest includes Automated software testing, Web technology. Email: manjeet2005@gmail.com,