Journal of Innovative Technology and Education, Vol. 2, 2015, no. 1, 29 -35
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/jite.2015.543

A Unique Technique to Handle the Complexity and Improve the Effectiveness of Test Cases in Software Testing

S. Baladwarakanath

Department of IT, Faculty of Computing
Sathyabama University, Chennai 600119, India

K. Vijay

Department of IT, Faculty of Computing
Sathyabama University, Chennai 600119, India

Abstract

Software Testing is a critical part of the whole process of development, on which the quality of the products delivered strictly depends. The testing is performed to verify the software using various types of information that are available in the project. When the internal state is present, a group of various function calls is mandatory for testing the software. At times, to test a particular section of the program, a group of function calls which are already used is needed to obtain the internal state of the code in exact configuration. This unique method of internal states is used in OOPS and also in software's that are procedural. Here we are analyzing the impact of the length of test cases plays in testing particular software in a single branch of code that is covered during the process of testing. We use this in difficult circumstances of software testing, randomly shortlisted test case set along with the test cases and source code make their testing trivial. Hence, we proposed new technique is complexity module that makes accuracy in testing the Source code and modules of the test cases. This technique will keep the internal state in proper configuration to cover the branch code by function call.

Keywords: Complexity Module, Random Test, Software Testing

1. Introduction

Software testing is the method used to verifying the error, bug, issue or differences in software for a particular input and its output which appraises the characteristics of software product. The process of executing white box testing on particular software has certain rules for finding a set of compatible test cases that fulfils the rules. Usually test cases play a role of driver which makes the call for function under test by particular input principles. Then compare the attained output with the expected output. By all the input is impractical because the numbers are unlimited. Hence by automating the process of software testing for automatically finds the set of inputs that is smaller than other possible input sets so the usage of testing criteria is maximum [2]. There may arise many problems in internal state. Internal states may be the example of static variables used in C language. In software using object oriented programming; most of the code involves internal states. Internal state creating more problem cause the source code coverage could be depending on the internal state status. The function call sequence is needed to keep the internal state in proper state [2]. Nowadays, we can generate a test set with large amount of code coverage. The major issue in this procedure to produced input may be complex structure. We are focusing on the producing of the enhanced test suite for input invention [1]. The test cases for a procedure of testing a particular built may consist of one or a sequence of input values [6].

We analysis the length of test sequences which has an impact on final result of the testing process. So that, we use a unique technique to analyses and arrive on a optimal result on a time which is much lesser when compared to many other techniques that are given in literature. This property can also be used in many other techniques. Thus in this searching techniques, we perform an analysis to find the usage of searching to find the optimal sequence

2. Related Work

The test sequences may be complicated to test the functionality of the object oriented software, in which the object is the instance of the class. There are many constructors and super-classes available for software. Each function that is involved in the process of testing may take the respective objects as their input. The objects should be instantiated, and possess unique sequences for calling function to rearrange their internal states in the perfect configuration [2].

Author presents the problem of generating the sequences of test inputs has been considered as the generation of test set for the container classes that are used, in random testing process, abstraction based “model checking” technique and the process which follows symbolic execution are the most applicable methods that can be used successfully[4].

Results of the studies on advanced techniques for test selection methods with random testing, but all the above mentioned studies failed to provide the answer for either of them. All the recent techniques employs random test set generation

for object-oriented software to perform unit tests but failed to explain larger code bases and can only generate complex inputs, while testing containers exploits smaller code bases in depth and requires only simple data inputs [5]. In a sequence of generations the individuals that survive are the fittest in that generation. The fitness value is calculated by executing the test case [3].

Here focuses on software-in-the-loop, in this category, the software that are embedded are tested in a particular simulated environment which is set up for development. The only difference is that we employ an adapter which fits hardware platform that sends all the signals from the SUT to the environment.

3. Proposed Work

3.1 Overview

Software testing is a process by which a software application or program is verified and validated that. Software testing is used to find important bugs in the application that should be solved before delivery. The defects, bugs etc must be categorized by severity .In our proposed work, we are proposing a unique technique called complexity module that makes accuracy in testing the Source code and modules of the test cases. The testing is performed in the system with configuration windows 7, 8 GB RAM, Intel i5 processor (64 bit).

3.2 Overall Architecture

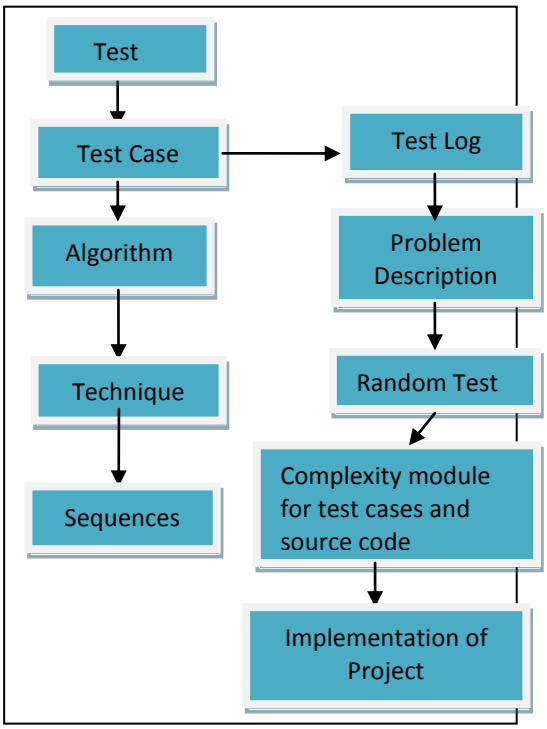

Fig.1. Architecture of the Software Testing

3.3 Test Case Generation

Test cases are the input given to the software that is to be tested. For example, a test case for testing a bill desk may contain a sequence of numeric values. There are various techniques available to develop a test cases, these creation of test cases depend upon the type of testing that is to be performed. The test case generation technique for user interface derived developed in this dissertation uses a model that is specific. The remainder of this section generates some test cases for user interface with their own limitations and shortcomings which are describes in some black-box testing techniques that are applicable for user interface testing.

3.4 Test Plan

Test plan is nothing but a requirement document. This is mandatory for testing. For straight-forward projects, the plan need not be detailed. The components that should be addressed in a test plan are as follows.

- Test process objective,
- Scope of the project,
- Testing process during the development,
- Environment set up for testing,
- Analyzing and identifying the risk factor,
- Bug Report, and so on.

3.5 Random Test and Complexity Module

Random testing is the basic technique for which will select the test cases randomly from the set of possible inputs to the program Complexity module is technique that makes accuracy in testing the code and modules. This technique will keep the internal state in proper configuration to cover the branch.

3.6 Algorithm

Algorithm: Complexity Module (f,c,r)
Input *f*: maximum fitness evaluations,
c: Number of consecutive test cases without improved fitness,
r: random test-cases that is generate for the purpose of comparison in Module.
Declare *E*: set of test cases that is executed = {},
R: set of test cases that are randomly generated = {}, *p*: performed fitness evaluations = 0, *i: consecutive* sequence of test cases without improved fitness = 0, *Rc*: Randomly selected test case, *Cf*: test case whose form is changed, *Tr*: test case that is segregated from *R*, *Ts*: test case segregated from *R which is selected* based on Complexity module, *Dm*: shortest distance covered by test case *Tr* with all possible cases to test *E*, *v*: maximum value of *Df* obtained over *R*
1. Begin
2. Set of randomly Generated test case Rc
3. Execute the randomly generated *Rc* and evaluate whether the environment error state is reached, Add *Rc* to *E*
4. **While** error in test Setup, then *AND p <= m AND i <= c*

5. Mutate *Rc* to get *Cm*, Execute *Cm* and check if the error state occurred in test environment
6. Add the value of *Cm* with *E*, Increase the value of f
7. **if** *fitness value(Cf) >= fitness value(Rc)* **then** *Rc = Cf, i = 1,* **else** Increment *i*
8. **while** test place error state not reached *ANDp <= m*
9. Sample *r* randomly selected test cases and add them to *R,v*= 0
10.**for each** *Tr* ∈ *R* , Calculate *Dm*
11. **if** *Df > v* **then** *v = Df, Ts = Tr*
12. Execute *Ts* and check if the error state is reached in test place
13. Add the value of *Ts* with the value of E, Increase the value of p
14. **end**

4. Experimental Result and Discussion

We have developed a unique technique for the software testing that make an effective approach in detecting the bugs in the software development. This technique will produce more accuracy in testing of the code.

4.1 Success rate for 8 Configuration of Complexity module on 8 Problems

Table.1. Success rate for 8 Configuration of Complexity module

Con→ Prob↓	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
Pt1	0.8	0.98	1	0.41	0.91	0.64	0.85	1
Pt2	0.7	1	0.96	1	0.96	1	0.83	1
Pt3	1	1	0.84	0.84	0.88	1	0.79	0.92
Pt4	0.9	0.94	0.48	0.58	0.85	1	0.76	0.76
Pt5	0.6	0.45	0.56	0.72	0.78	0.96	0.99	0.59
Avg.	0.75	0.82	0.83	0.75	0.91	0.84	0.837	0.84

In Table 1, we are showing the test result that representing our proposed technique Complexity module accuracy.

4.2 Success Rate of Complexity module and other Techniques

Table.2. Success rate of complexity module and other techniques

Method	**Pt 1**	**Pt2**	**Pt3**	**Pt4**	**Pt5**	**Pt6**	**Pt7**	**Avg.**
GA	1	0.90	1	0.72	0.92	1	0.85	0.88
EA	0.86	1	0.90	0.45	0.88	0.79	0.74	0.825
RSA	0.90	0.39	1	0.76	0.69	0.63	1	0.786
HC	1	0.54	0.86	0.71	1	0.97	0.82	0.832
CM	0.95	1	1	0.90	0.78	0.80	1	0.927

In Table 2, we are doing comparison between the Complexity module and other existing techniques (Pt is the problem in particular time).

4.3 Comparison of Complexity Module with other Techniques

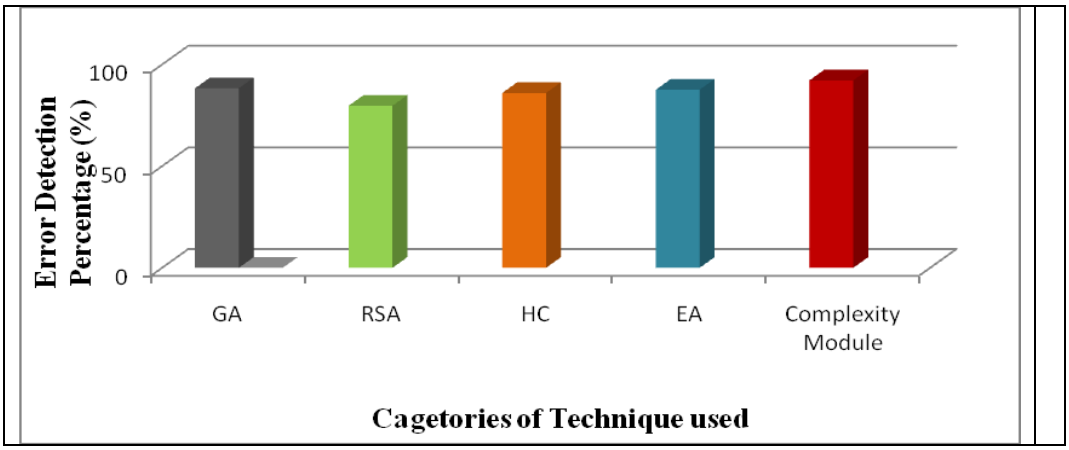

Fig.2. Complexity Module vs Different Techniques

In Fig 2, the graph is showing the direct comparison with Genetic Algorithm, Random Search Algorithm, Hill Climbing and Evolutionary Algorithm.

4.4 Code Coverage

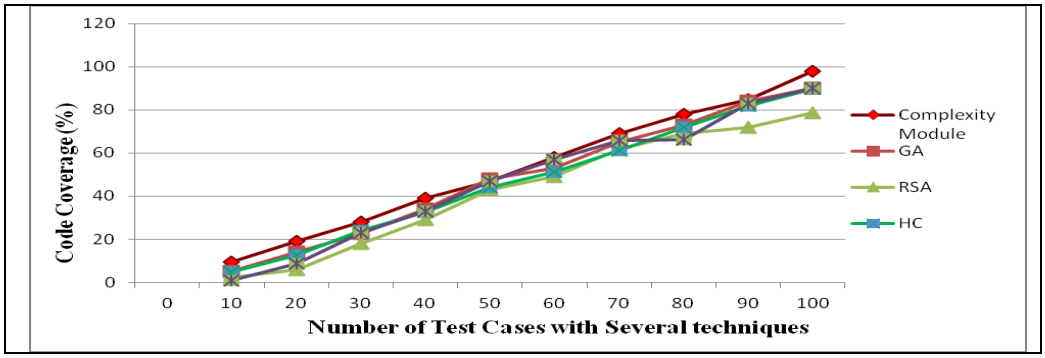

Fig.4. Code Coverage by Several Techniques

In Fig 4, the graph is showing the code coverage capability by the different techniques. Code coverage process is keeping the internal state in proper configuration that is making a most reliable testing.

5. Conclusion

We have implemented a unique technique to handle complexity modules that

make accuracy more in testing the source code and its modules of the test cases. After overcoming the problem of software testing, this unique technique will produce more effectiveness result in test cases. The capacity of detecting error is more in compare to several existed technique like GA (Genetic Algorithm), HC (Hill Climbing), RSA (Random Search Algorithm), and EA (Evolutionary Algorithm). This technique will keep the internal state in proper configuration to cover the branch code by function call. Here, we are going for the random test which provides the more accuracy in detecting the error.

References

[1] A. Arcuri, It Does Matter How You Normalize the Branch Distance in Search Based Software Testing, *Proc. IEEE Third International Conference on Software Testing, Verification and Validation* (2010), 205 - 214. http://dx.doi.org/10.1109/icst.2010.17

[2] A. Arcuri, Longer Is Better: On the Role of Test Sequence Length in Software Testing, *Proc. IEEE Third International Conference on Software Testing, Verification and Validation* (2010), 469 - 478. http://dx.doi.org/10.1109/icst.2010.16

[3] A. Arcuri, A Theoretical and Empirical Analysis of the Role of Test Sequence Length in Software Testing for Structural Coverage, *IEEE transactions on software engineering,* **38** (2012), 497 - 519. http://dx.doi.org/10.1109/tse.2011.44

[4] Alex Groce, Coverage Rewarded: Test Input Generation via Adaptation-Based Programming, *26th IEEE/ACM International Conference on Automated Software Engineering* (2011), 380 - 383. http://dx.doi.org/10.1109/ase.2011.6100077

[5] M. d'Amorim, C. Pacheco, T. Xie, D. Marinov, M. D. Ernst, An empirical comparison of automated generation and classification techniques for object-oriented unit testing, *International Conference on Automated Software Engineering (ASE)* (2006), 59 - 68. http://dx.doi.org/10.1109/ase.2006.13

[6] Sapna Varshney, Monica Mehrotra, Automated Software Test Data Generation for Data Flow Dependencies using Genetic Algorithm, *International Journal of Advanced Research in Computer Science and Software Engineering,* **4** (2014), 209 - 218.

Received: April 16, 2015; Published: July 17, 2015