Part 4: Final Submission

Student: Xinran Peng

Onid: pengxin

Class: CS569s16

Instructor: Prof Groce Alex

2016/6/6

Section 1: Introduction

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. [1] Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs. Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases. When we don't know anything about the code, we usually use black-box testing. Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it.

What we will use in this project is Random testing. Random testing is a black-box software testing technique where programs are tested by generating random, independent inputs. Results of the output are compared against software specifications to verify that the test output is pass or fail. In this project, we propose a modified version of random testing called adaptive random testing. This method seeks to distribute test cases more evenly within the input space. It is based on the intuition that for non-point types of failure patterns, an even spread of test cases is more likely to detect failures using fewer test cases than ordinary random testing.

Section 2: Algorithms

In this project, my idea of this algorithm came from Professor Alex. The main idea of this project is using fewer tests to detect the software faults in TSTL. The

project also uses some special predefined values that have high tendency of finding faults in the SUT. In my tester2.py file, what I mainly do is use the random test to generate test cases, and then do the random action. Then the program will check if the action is safe action. If the action is safe, then it will check if the coverage of the random test is smaller than the mean, then it will test these parts. In the first part of the program, I use the same function as the randomtester.py. it will add command line parameters. Then define a function also used in randomtester.py called make_config, and returns a dictionary.

Section 3: Run Test Generation

This program is able to run in the command line to enter parameters (timeout, seed, depth, width, faults, coverage and running) with it. We can use the input 'python tester1.py 30 1 100 1 0 1 1' as an example. The first parameter is for the timeout, which the test generation will stop when the runtime reaches that number. The second parameter is for seeding which generates a random number of seeds. The third parameter is for the depth and the fourth parameter is for width. The fifth parameter is for fault which is if we put 1 that mean the test generation will search for bugs and if it is 0, means the test generation do not look to find bugs. The sixth parameter is for printing TSTL internal coverage report to know the statement and branches were covered during the test to guide it. This will work when we specify 1 and 0 for do not print it. The last parameter is for running which will print the elapsed time, total branch count and new branch. It works when we put 1, 0 will not do anything.

Section 4: Experiment Results

In my tester, I used the command "python finaltester.py 30 1 100 1 0 1 1" to test the avl.py, which means that in depth 100, 30s running. I covered 191 branches and 141 statements. Here is the result I got from my tester.

```
59, 262, 263, 264, 267, 268, 270, 271, 27
TSTL BRANCH COUNT: 191
TSTL STATEMENT COUNT: 141
(u'/Users/wyh/Desktop/CS569/avl.py', 93)

6 failed
Total tests: 158
Total actions: 15258
Total running time: 30.0483119488
```

I run my tester for several times and the coverage is almost the same even if

I change the running time or the depth. I think this is because of the method I used.

Section 4: Conclusion and Further Work

The purpose of this project is using TSTL package to implement a test generator. At the end of this term, I finished my tester and got a (kind of) good result. Because I have already taken CS 562 last term, TSTL is not very new for me. This makes my work kind of easy then some other students. But python is still difficult for me because I rarely use this language. I struggled in the first submission, and cannot get the right output when the first time I submitted the code. Then I talked with some of my classmates and they also gave me some help. Then the work became easier then before, and I could get the correct result. Still, my tester is not good enough. I observed the outputs of the whole classmates and found that my tester is just in the middle ranking from branches coverage and statement coverage. In the future, I think I will read other classmates' codes, especially those with better performance, and try to learn from them. Random testing is a very useful method in software testing, and learning how to implement random testing helps me a lot in understand the basic idea of random testing and even some other testing method.

Reference

[1] Cem kaner(November 17, 2006). "Exploratory Testing". Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL. Retrieved November 22, 2014.

[2] Alex Groce, Gerard Holzmann, and Rajeev Joshi. Randomized differential testing as a prelude to formal verification. Software Engineering, 2007. ICSE 2007. 29th International Conference on, pages 621-631. IEEE, 2007.

[3] I. M. T.Y. Chen, H. Leung. Adaptive random testing, 2004.

[4] Richard Hamlet (1994). "Random Testing". In John J. Marciniak. Encyclopedia of Software Engineering (PDF) (1 ed.). John Wiley and Sons. ISBN 0471540021. Retrieved 16 June 2013.