CS 569
Project Proposal
04/19/2016
Hao Wang (wanghao@oregonstate.edu)

Implementation of Feedback-directed Random Test in TSTL

1. Background

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied to virtually every level of software testing, unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well.[1] For Black-box testing, testers do not need to know the process of implementation, or focus on source code. The high generality of method and low requirements of testers lead to a high concurrency and collaboration of the whole project process, which is the core idea of software engineering.

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.[2] As an effective method of black-box testing, it is a complement of white-box testing to garantee the completeness of the test process. However, random testing is not a good enough method sometimes since its inherent shortage. For example, its generated test cases are not usually practical comparing with the real-world cases, and many test cases are redundant. What's worse, generating test cases randomly usually causes a high use of time and a low coverage of all possible test cases. Currently, there is a number of modifications that work with the shortage of random testing.

2. Strategy and Plan

I get the idea of algorithm from the paper "Feedback-directed random test generation".[3] In all cases, a good test is a test that finds the error, thus a good test plan is a test plan that has a high probability to find the error. Instead of just creating inputs randomly, this algorithm executes each input in every step, stores the outputs, and use the outputs to guide the next random input's generating. In other word, though the generation of inputs is still random, the inputs themselves are somehow dependent in order or weight. Algorithm generates not only random inputs, but also input sequences for each dependent input. This is a better way to obtain a high coverage. What's more, this algorithm provide a redundance cleansing, which makes the test more like a real-world case.

I am going to implement the Feedback-directed Random Test algorithm in a TSTL version that can apply on the public SUTs. The ideas are all in paper,[3] thus what I should do is delivering in code. Since it is hard in TSTL, I think I could reserve the

right to write and use my own SUTs which will be similar to the public SUTs but might has a simpler data structure or function

Reference

[1]. S. Debnath. *"Mastering PowerCLI"*. Packt Publishing Ltd. 2005. pp.35.

[2]. R. Hamlet (1994). *"Random Testing"*. In J. J. Marciniak. *"Encyclopedia of Software Engineering"*. John Wiley and Sons. 2013.

[3]. C. Pacheco, S. K. Lahiri, M. D. Ernst, T. Ball. *"Feedback-directed random test generation"*. In *"Proceedings of the 29th International Conference on Software Engineering"*. 2007. pp.75-84.