Proposal for Graduate Research

Student Names:	Waseem Ilahi	(wki2001@columbia.edu)
	Moses Vaughan	(mkb2014@columbia.edu)
Term:	Fall 2009	
Advisors:	Chris Murphy, Prof. Gail Kaiser	
Date:	September 20, 2009	

Introduction:

Much work has been done on solving the problem of being unable to check all possible combinations of configurations inside a newly developed or/and pre-deployed software, leading to the ability of detecting and reporting back the residual bugs in the software. “Guanyin Research Project” and “In Vivo Testing” are two of the projects that attempt to create a resolution to this issue. InVivo Testing Approach tries to execute the set of tests that the developer wants, but it does so not only after the development phase is complete but also after the deployment of the software into the production environment. The concept behind “InVite”, the software suite that implements the “In Vivo approach”, is to apply the continuous testing to execute these tests in a way that it never affects the original software or its performance, note also that there is no side effect caused as a result of these tests.

The tasks that are ahead of us this semester are to answer the question: Can the technique (InVivo Testing) be made more efficient by only running tests in application states that the test system has not come across before? We currently run tests probabilistically or according to the system load; however, certain states may unnecessarily be tested multiple times. This potential efficiency will not only cure the redundancy issue of multiple unnecessary executions but also have a positive effect on the reservation of wasted system resources.

To achieve such a goal, we will need to first define a “state” for any unit of execution and then figure out how we can judge whether a state has been accessed before by InVite, or not. We need to do this in constant time; O (1). For the second task, we will need to look into a data structure that could provide us with the time efficiency yet search capabilities that can occur in constant time such as a “reverse” bloom filter, which is a space efficient data structure which tests whether an element is part of a set or not. We will use this to search for potential states which our test system has come across previously. The one compromise with the bloom filter itself is that it allows false positives but not false negatives. In the case of unit testing we would like to achieve a scenario where false negatives run are allowed (units that were previously run but now will be run again) but false positives are not allowed (units that were never previously run erroneously labeled as seen before). Finding such a system can be as simple as making a tweak to the bloom filter, but also could involve investigation to discover a totally new structure to suit our skills.

Objective:

The primary objective is as follows:

- Research and implement a manner in which to perform the InVite tests such to avoid the tests that have already been executed. This less redundancy of running previously successful tests, thus providing a good performance gain. In order to achieve such a goal we must execute a number of primary tasks:
    - Research and develop a state, represented by some data structure that represents a semantically unique execution of the function/test in the program. This will produce great efficiency in equating semantically equal test runs.
    - Research and integrate a “fast” (constant time) data structure that would satisfy the needs of not only a time requirement of a member operation but must be able to provide if necessary false negatives but no false positives (Reverse Bloom Filter)
    - Measure the efficiency of the resulting product, by factoring in the overhead and comparing against the original technique used by InVite.

Approach:

To avoid the waste of time it has been decided that working in parallel would be best as we shall find a definitive manner to divide the research workload in a way that allows us to frequently collaborate on ideas that may be helpful to each other when the need arises. As for the division it shall go as follows: One will look into finding the minimal data set for the state representation, while the other investigates ways in which we can use the set of variables collected for the state to get their values right before the test is executed. After we have a state representation, we will investigate the possibilities of the “data structure” that can hold these states and provides the functionality to traverse the structure very fast to figure out whether we have seen a particular state before or not. Once this has been researched we shall come together to design, implement and integrate the respective components.

Project Plan:

This section describes the major action plan activities for the duration of the project.

1. September

Tasks:

- Plan and design the structure of the project, including objectives, scope, action plan, and end resulting deliverables.

2. October

Tasks:

- Define the execution unit's state as mentioned above.
- Research and define the appropriate data structure to represent a state.

3. November

Tasks:

- Research and define/implement a (data) structure to provide an efficient lookup of the aforementioned state.
- Research and determine a manner in which to run the test in a state and find out whether we have run it before or not given a new or repeated set of inputs with the help of the above mentioned "entity".

4. December

Tasks:

- Run empirical measurements of efficiency, speed and space as well as work overhead.

Deliverables:

- <u>Final report(s)</u>
- <u>All the source code & the Research Material</u>