A data driven framework for test case selection

## Chapter 1 Introduction

### 1.1 Problem Description

Test Case Selection is essentially a problem about choosing a subset of test cases from the test suite. Formally, Test Case Selection problem is defined as follows [] -

*Given the program P, the modified version of P, P’ and test suite T*

*Find a subset of T, T’, with which to test P’*

We will use the following notation in the discussion of test selection problem:

Let P be the current version of the program under test, and P’ be the next version of P. Let S be the current set of specification of P and S’ be the set of specifications for P’. T is the existing test suite and t is the individual test case in T. P(t) represents execution result of P with test case t. We define an execution trace ET(P(t)) for t on P to consist of the sequence of statements in P that are executed when P is executed with t.

#### 2.1.1 Graph Walk

#### 2.1.2 Textual difference

Other techniques have also been proposed to select tests to cover changed code from P to P’. Instead of identifying program modification based on graph representation, Volkolos and Fankl [13]proposed a technique based on textual difference between the source code of P and P’. Unix diff tool is used to compare source code of P and P’ to identify modified program statements. This technique also require source code of P and P’, test suite T and execution trace ET(P(t)) of every t on P. However, they do not keep execution order in the execution trace ET(P(t)). Execution trace only contains a set of basic blocks that have been executed when program P is executed on test t.

The empirical study results on a 11,000 lines of code software system provided evidence that textual differencing technique is very fast and capable of achieving substantial reductions in the size of the regression test suite. However, the case study was performed on a C program which may not be representative to Object Oriented Programs. Although authors believe their technique is as precise as graph walk techniques, there was no direct comparison on precision of this approach to others.

### 2.2 Design modification based techniques

Briand et al. presented a black box design level test selection technique based on UML[15,16] . This technique requires UML design models of P and P’, test suite T and execution trace ET(P(t)) on P at method level. Comparing to test selection techniques based on source code control flow and data flow analysis, this technique is efficient and provides better support for assessing and planning regression test effort earlier in the change process. Effort estimation of regression test can be performed as soon as design is complete.

One of the constraints of this technique is that it requires designs to be complete and up-to-date, which may not be the case in practice. The technique also assume one, complete sequence diagram for each use case. In practice, scenarios can be specified across several sequence diagrams to improve their readability . This technique requires class methods are described by providing their precondition and postcondition. Authors concluded that it’s likely this technique is not as precise as code modification based techniques.

### 2.3 Defect correction based techniques

Wikstand et al.[17] proposed a simple test selection technique that uses a cache to monitor fault-prone files and recommends test cases to rerun to cover updated files. This technique require source code of P and P’, test suite T and linkage information between defects, test cases found the defects and source files updated to fix the defects. Empirical study on a large system with five digit number of source files has shown effective selection of test cases covering fault-prone files.

Comparing to other code modification based techniques; this does not require execution trace information. This technique is useable in practice when code coverage information for each test case is not readily available, such as in real-time and embedded systems, or any environment collecting coverage information is too expensive. Test case code coverage information is instead computed from what files were updated to fix a fault found by the test case.

It’s reasonable to believe this technique’s fault detection effectiveness is lower than code modification based techniques [5-9] as this technique is not safe. In practice, this may be used as a test case recommendation system compliment to other fault detection approaches. Authors have not completed evaluating the quality of the test cases recommend by this technique, therefore the precision of the selection is unknown.

Related work on Economic Models

Hyunsook Do et al. conduced empiral study based on a economic models that capture both cost and benefits of prioritization, including factors related to rate of fault detection, omission of faults , and cost of applying techniques.

Chen and Rosendblum[5] ‘s technique is based on code entities that are coarser-grained than those by statement or control-flow-based techniques, it may select more test cases with less computation cost.