**Coverage Collection and Test Generation in Java**

Project Proposal

Wilfred Labue  
 Computer Science Department   
 University of Texas at Dallas  
 Richardson, TX 75080  
 wxl180015@utdallas.edu

Lathwahandi De Silva  
 Computer Science Department  
 University of Texas at Dallas  
 Richardson, TX 75080  
ldd170230@utdallas.edu

Ning Kevin Chen  
 Computer Science Department  
 University of Texas at Dallas  
 Richardson, TX 75080  
nkc160130@utdallas.edu

**ABSTRACT**

In this project we will be using ASM byte-code manipulation framework [1] to build an automated coverage collection tool that can capture the statement coverage for the program under test. Then we will apply our tool to 10 real-world Java projects which contains at least 1000 lines of code with JUnit tests at least 50 tests collected from GitHub [2] to collect the statement coverage for its JUnit tests.

The tool will be a Java based agent with ability to perform on-the-fly code instrumentation, and it will be able to store the coverage for each test method in the file system. The tool will be integrated via the Mavin build system which will in turn will simplify the tool execution by reverting the execution command to “mvn test”. In order to achieve this integration with the Maven build system we will create the necessary changes in the pom.xml file of the project located in test.

1. **Coverage Collection - Diyohan**

The primary objective of phase one is providing an automated coverage tool that captures code coverage, specifically statement coverage in a program which is subjected to various test cases. In this section we will be examining some primary objectives that should be achieved in phase one.

**1.1 Statement Coverage**

The premises of the coverage collection tool is to collect statement coverage of program subjected to a test. Statement coverage is a technique that is used to design white box test cases. This technique involves execution of all statements of the source code at least once. This methodology is used to calculate the total number of executed statements in the source code against the total statements present in the source code.

1. **Test Generation Techniques**

Since the project is in preliminary stage we decided to expand the initial scope to garner a better understanding of finding the optimal techniques we could implement to refine the solution. In the following subsection we will be discussing the various techniques we are considering to implement on the solution.

**2.1 Pairwise Testing**

Pairwise testing is a black-box testing technique. That is, it does not rely on modifying or manipulating the code to generate test cases. Instead, it relies on passing in different values into parameters to the functions within the program to generate tests. Now, say you have 34 checkboxes on a website. Since each checkbox can be set to checked or unchecked, the number of total inputs is 234 inputs, or 1.7 \* 1010 inputs. That means that, in order to test all inputs individually, you need to generate 1.7 \* 1010 tests! Now, pairwise testing makes this much easier. If you consider that there are 3-way or 4-way interactions between the checkboxes, you can bring the number of tests down to 33 or 85, respectively. That’s a lot more efficient than 234 test cases! Usually, the more way-interaction a pair is, the more percentage of bugs it will catch. This method finds useful applications from pizza ordering websites to Traffic Collision Avoidance System modules**.**

**2.2 Random Test Generation**

This is a black box software testing technique where programs are tested by generating random independent inputs. In random testing a random number generator (e.g monkeys ) is used to generate test cases and derive estimates for the reliability of the software using some probabilistic analysis. With random testing random inputs are entered into a system to see what the results are. A common example is the use of random integers to test a software function that returns results based on those integers. Other type of random testing may involve the use of heuristic , which guide the use of random inputs.

1. **Code Coverage Tool Design**

**3.1 JUnit** l

JUnit is a unit testing framework used in Java programming. JUnit has been important in the development of test-driven development. It is a simple framework used to write repeatable tests and an instance of the xUnit architecture for unit testing framework. JUnit promotes the idea of “first testing then coding”, which emphasizes on setting up the test data for a piece of code that can be tested and then implemented.

3.2 Maven

Maven is a build automation tool used in Java projects. It automatically downloads Java libraries, as well as Maven plugins, from the Maven repository. Maven utilizes one file, the pom.xml, to configure the Maven project for building. POM stands for Project Object Model, and the file is in XML format. For example, one can use Spring Boot, a framework that allows you to build web applications using Java, in a Maven project..

3.3 ASM Bytecode Framework

ASM is an all purpose Java bytecode manipulation and analysis framework. it can be used to modify existing classes or dynamically generate classes , directly in binary form. The ASM library is used to generate , transform and analyze compiled Java classes , represented as byte arrays. ASM provides tools to read, write and transform such byte arrays by using higher level concepts than bytes , such as numeric constants , strings , JAva identifiers, Java types and Java class structure elements.

3.4 Java Agent

When it comes to the modification of the bytecode located in a file, there are several limitations induced as a result of this phenomenon. Initial limitation is the excessive consumption of time and secondly the risk induced approach that can create disruption among the bytecode in the file system in a fundamental level. Therefore a Java agent is used to transform the byte code files during JVM, this tactic leads less complexity in regards to unforeseen altercations that can occur to the actual byte code file located in the system.[2]

1. **Evaluation/ Team Schedule**

4.1. Code Coverage Tool Evaluation

In order to correctly evaluate the results presented by our code coverage tool, the primary course of action taken will be cross reference among results produced by a pre-existing code coverage tool. Currently we have couple pre-existing code coverage tools that we could implement due to the preliminary nature of the project we have yet to determine the final tool we will be implementing. The implementation options are as follows. Emma and Jacoco.

Emma is is an open-source toolkit for measuring and reporting Java code coverage. EMMA distinguishes itself from other tools by going after a unique feature combination. [1] The reason for considering Emma as a possible candidate is primarily due to the instrument classes for coverage either offline (before they are loaded) or on the fly (using an instrumenting application classloader). [1] The Code coverage tool Jacoco was created as a spiritual successor for Emma and Cobertura. The reason for selecting Jacoco as a candidate is due to its support for EclEmma plugin, support for Mavin, and active development team . [2]

4.2 Team schedule

The current team schedule consist of weekly physical meetings on Friday between 1pm- 3pm, The primary reason for selecting this time slot is due to mutual team member availability presented in the allocated time slot. Another note be regarded is due to the preliminary nature of this paper the time slot presented above is by no means permanent and it is subjected to change. Over the following weeks we will be fleshing out the details regarding the nature of the strategies we will implement in conjunction to overlay/ implementation of the custom code coverage, test case generation, and pre-existing code coverage tool that will be implemented for cross reference/ validation.

**References**

[1] Vlad Roubstov. 2006. a free Java code coverage tool. (January 2006). Retrieved September 30, 2019 from http://emma.sourceforge.net/

[2] Ling Ming Zhang. SE 4367 Course Projectl. Retrieved September 25, 2019 from <https://elearning.utdallas.edu/bbcswebdav/pid-2735995-dt-content-rid-69989771_1/courses/2198-UTDAL-SE-4367-SEC501-84822/course-project.pdf>

[3] About StackifyStackify provides developer teams with unparalleled visibility and insight into application health and behavior. 2019. Code Coverage Tools: 25 Tools for Testing in C, C , Java. (May 2019). Retrieved September 30, 2019 from <https://stackify.com/code-coverage-tools/>