Automated Mutation Testing Framework using AST

Dhrupad Kaneria
Graduate Student, UT Dallas
800 Campbell Rd
Richardson TX 75080
214-299-3874
dhrupad.kaneria@utdallas.edu

Purva Dahake
Graduate Student, UT Dallas
800 Campbell Rd
Richardson TX 75080
469-450-9959
purva.dahake@utdallas.edu

ABSTRACT

This report gives insights about the progress of the project undertaken by evaluating the statistics of the already existing test cases, specifying the approach being followed to complete the project and outlaying the future work to be done. This report gives a brief idea about the path decided on to proceed further. The ultimate goal of the project is to perform mutation testing on a real world Java application (>1000 lines of code) with JUnit tests (50 tests) taking the help of AST.

1. INTRODUCTION

1.1 Mutation Testing

Mutation testing [1] (or *Mutation analysis* or *Program mutation*) is used to design new software tests and evaluate the quality of existing software tests. Mutation testing involves modifying a program in small ways. Mutation Testing is a fault-based technique that measures the effectiveness of the test suites for fault localization, based on seeded faults. Each mutated version is called a *mutant* and tests detect and reject mutants by causing the behavior of the original version to differ from the mutant.

Following are the three types of mutation testing [2]:

1. **Value Mutations**: An attempt to change the values to detect errors in the programs. We usually change one value to a much larger value or one value to a much smaller value. The most common strategy is to change the constants.

2. **Decision Mutations**: The decisions/conditions are changed to check for the design errors. Typically, one changes the arithmetic operators to locate the defects and also we can consider mutating all relational operators and logical operators (AND, OR , NOT)

3. **Statement Mutations**: Changes done to the statements by deleting or duplicating the line which might arise when a developer is copy pasting the code from somewhere else.

1.2 Abstract Syntax Tree (AST)

An Abstract Syntax Tree [3] is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. The syntax is "abstract" in not representing every detail appearing in the real syntax. For instance, grouping parentheses are implicit in the tree structure, and a syntactic construct like an if-condition-then expression may be denoted by means of a single node with three branches.

1.3 Eclipse JDT

JDT [4] is a framework that provides provision to manipulate the java source code. Since it is generally attached to Eclipse, it is often called an Eclipse JDT. The JDT is broken down into components. Each component operates like a project unto its own.

The JDT project contributes a set of plug-ins that add the capabilities of a full-featured Java IDE to the Eclipse platform. The JDT plugins provide APIs so that they can themselves be further extended by other tool builders.

1.4 JUnit Tests

JUnit [5] is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit. A research survey across 10,000 java projects hosted on GitHub found that JUnit was most commonly used. It is was used by 30.7% of the projects.

2. APPROACH

Since we have changed the project topic earlier this week, we have not spent enough time into the project. At this point, we have spent most of the time in understanding and debating the approach for the project. The following are the outcome of the discussion and research conducted.

2.1 Stage 1:

The first step in the project is to select the appropriate specimen for mutation testing. The criteria for selecting the examination specimen are:

- it should be sufficiently large (>1000 lines of code)
- it should have more than 50 unit test cases.
- the examination specimen should be able to be executed programmatically.

Once the selection of the specimen is made, we would start writing the java code to access this subject taking the help of Eclipse JDT with the AST.

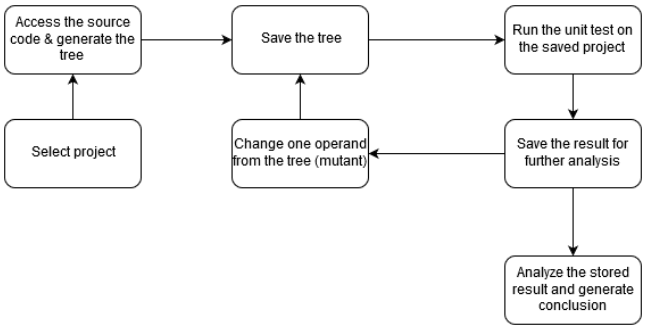
2.2 Stage 2:

The second step in the project will be to access the specimen and store it as an Abstract Syntax Tree (AST). After successfully generating the AST, the next step is to run the unit test cases on the actual program (the AST format). The result of the execution (the number of tests passed, failed) is noted down. This execution is carried on the actual java program before generating any mutants.

2.3 Stage 3:

The third step is to create a mutant. We have deiced upon creating the mutants based on the Statement Mutation mentioned earlier. This is done by changing one operator in the generated AST.

Sequentially, we intend to change one operator at a time and run the updated code across the existing unit test cases. At the end of each execution, the tests that passed and failed are noted down. Depending on the result of the execution, the mutants are either classified as dead or alive. This statistics will help us to determine how strong and accurate the unit tests are.



3. ACCOMPLISHMENTS

Due to our change of project topic recently, we have had very little time until this point to work on the new project. Despite of the time constraints, we have started with the project work and accomplished following things:

3.1 Research

Since we have to choose one real time Java project with more than 1000 lines of code and more than 50 JUnit tests, we have to go thoroughly through the list of projects provided by the professor. Our team is currently researching on the appropriate project to be considered as a specimen. Along with the research, we are also getting familiar with the using AST api and created a sample code to test the creation of AST as mentioned in the next point.

3.2 Created AST

Since we are still deciding on the examination specimen, we have created a sample test file with one operation. We are developing the code while testing this across sample test file. The code to create the Abstract Syntax Tree (AST) is successfully completed and it was tried on the sample test file that we created to check creation of AST.

3.3 Fetched Expression Statements

From the AST generated for sample test file, we fetched all the statements. Among these statements, we have separated the expressions which are of interest to us. We are currently working towards accessing the expression and changing the operator. Once the operator is changed, we will proceed to execute the newly generated mutant to execute programmatically.

4. FUTURE WORK

At this point, we have two main problems to face. One of them is to choose an appropriate project which meets all the criteria specified in the project description. And secondly, the project should run programmatically. We will be consulting the professor as to how to select the appropriate project if we get stuck.

The next step we would be implementing once a project is selected is to generate the AST of the new real world Java project and apply decision mutation on it to perform mutation testing on it. That is, we will be changing a single operator in the program and run it as one mutant. Then apply all the JUnit tests on it and analyze the results. Once it is successfully done, we will change other operators similarly and analyze the test statistics.

If time permits, we also intend to change more than one operator in one mutant and derive new insights in test statistics. We also have to decide on the metrics of measurements of test statistics. This metrics will inform us about the quality of the test cases and also the correctness of the test cases.

5. REFERENCES

[1] Mutation Testing, https://en.wikipedia.org/wiki/Mutation_testing

[2] Mutation Testing, http://www.tutorialspoint.com/software_testing_dictionary/mutation_testing.htm

[3] Abstract Syntax Tree, https://en.wikipedia.org/wiki/Abstract_syntax_tree

[4] Eclipse JDT, https://eclipse.org/jdt/

[5] JUnit, https://en.wikipedia.org/wiki/JUnit