CMPSC 290
Principles of Software Development
Fall 2013

Laboratory Assignment Seven: Using Mutation Analysis to Evaluate Test Suites

Introduction

In the previous laboratory assignments, you learned about the tools that we will use to specify, design, implement, test, and document Java programs. You also had several experiences with working in progressively larger teams to complete the phases of the software life cycle, with a recent focus on the elicitation of software requirements, the planning of software projects, and the analysis of a project's design and implementation. In this assignment, you will install and use a mutation analysis tool that will help you to determine the quality of a test suite. During the final project, you can use this tool to better gauge how effectively you are testing your code. A week from now, you will demonstrate to the instructor that you are able to perform mutation analysis.

Mutation Analysis with MAJOR

Mutation analysis purposefully inserts hypothetical faults into the source code of your program and then checks to see if the test cases can find these faults. If at least one test case fails after the insertion of the mutant, then the tests can find this type of fault and we say that the mutant was killed. However, it is also possible that the tests will not be capable of finding the mutation fault. What does it mean if the mutant is alive after the execution of the test suite?

MAJOR is a mutation analysis tool that integrates into the standard Java compiler. You can learn more about MAJOR by visiting the Web site `http://iai.mathematik.uni-ulm.de/en/research/major.html`. After examining this Web site, you should continue to learn about MAJOR by studying the provided tutorial and the papers referenced by the tutorial. After you understand the concepts behind MAJOR, you should ask the instructor to give you read access to MAJOR's Git repository hosted on Bitbucket. Once you have access to the repository, you should type `git clone git@bitbucket.org:gkapfham/major-cs290f2013.git` in your terminal window. Now, create a Git repository for your team that follows the naming convention that we use for laboratory assignments and move all of the files associated with MAJOR into it.

In MAJOR's main directory you should see the scripts `ant`, `javac`, and `mmlc`. In order to continue with this assignment, you will need to use the `chmod` command to make all of the scripts executable. Once these scripts are executable, you should go into the `example/ant` directory and also make the `run.sh` script executable. Now, you can study the build.xml file provided with this example. What targets do you find in this file? If you wanted to perform mutation analysis of a Java program, in what order would you suggest running these targets? Why?

Go into the `src/triangle` directory and study the Triangle.java file that contains a `Triangle` class with a `public static Type classify(int a, int b, int c)` method. What is the purpose of this method? How does this method work? You should also notice that Triangle.java has a `public static enum Type`. What is the purpose of `Type`? Once you have carefully studied this program and discussed it with the members of your team, write a `main` method that shows how to call the `classify` method with at least five different inputs. What output does Triangle produce?

To ensure that your mutation analysis results match those found in the provided tutorial, you should now comment out the `main` method that you previously implemented; please uncomment this method before you print the required deliverables. Once you have returned Triangle.java to its original state, you should find and study the TestSuite.java file. How many test cases does this suite have? How do these test cases work? Make sure that you fully understand these tests.

Now, you are ready to execute the `run.sh` script in the `example/ant` directory. When you run the script, you should see the output that is the same as what you find in the tutorial. If you do not see output containing lines like those below, then please see the instructor. Working with your team members, carefully study the output from mutation analysis. How many mutants were killed? How many mutants are still alive? What do these results tell us about the quality of the tests?

```
[junit] MAJOR: Total runtime:  0.7 seconds
[junit] MAJOR: Mutation score: 88.61%
[junit] MAJOR: Mutant executions: 79
[junit] MAJOR: Mutants killed / live: 70 (70-0-0) / 9
```

Let's examine MAJOR's output in greater detail. Look in the kill.csv file and find a mutant that was killed by the test cases (i.e., a mutant with the "FAIL" annotation). Next, examine the mutants.log file and determine the source code manipulation to which this mutant corresponds. What type of test input will kill this mutant? Finally, you should repeat these steps for a mutant that remains alive after running the test suite (i.e., a mutant with the "LIVE" annotation). In what way did MAJOR modify the source code to produce this mutant? Why are the test cases not capable of killing this mutant? Can you devise a test case that will kill this or other live mutants?

Summary of the Required Deliverables

This assignment invites your team to submit one printed version of the following files:

1. A description of and justification for your team's chosen organization, roles, and tool support
2. A description of the targets in build.xml and the order in which they are run during mutation
3. A modified version of Triangle.java that contains a `main` method for demonstration purposes
4. The output from running the modified version of Triangle.java that contains a `main` method
5. The output from executing the `run.sh` script in the `example/ant` directory
6. A detailed analysis of the output produced by the execution of the `run.sh` script
7. A thoughtful commentary on at least one live and at least one killed source code mutant
8. A listing of the steps that you will take during next week's demonstration of MAJOR

You must also ensure that the instructor has read access to your Bitbucket repository that is named according to the convention `cs290F2013-lab7-team`*`k`*, with *`k`* representing the number of your assigned team. Your repository should contain all of the deliverables that you produced during the completion of this assignment. A team can earn extra credit if they successfully apply MAJOR to another software system (e.g., a system from the last laboratory assignment). To earn the extra points you must submit an instruction manual that fully explains how to use MAJOR to perform mutation analysis on your chosen project. Please see the instructor if you have any questions.