SENG 438 - Software Testing, Reliability, and Quality

Lab. Report #3 - Mutation Testing and Web app testing

Group #:	19
Student Names:	Eric Renno
	Ryan Sommerville
	Quinn Ledingham
	Kaumil Patel

Introduction

Creating a test suite to perfectly test a system is a near impossible task. Even if you systematically perform white box and black box testing techniques, you can never be sure that you haven't missed something. Mutation testing is designed to help you build confidence in your system by checking whether it can catch common mistakes that are purposefully inserted into the system. In other words, it tests the test suite. You know that your test suite works well if it catches all of the mutants, and otherwise you revamp the test suite.

In this lab, we explored how to use Mutation testing to ensure our test suites worked well. Furthermore, we used another software called Selenium to test the GUI's of websites, ensuring that the websites worked smoothly by checking certain functionalities of the interface.

Analysis of 10 Mutants of the Range class

144 - Contains

1. Incremented (a++) double local variable number 1 -> SURVIVED
   increasing the value by one after it is used is equivalent to how the contains worked before because the local variable is not used again.

2. Incremented (a++) double field upper -> KILLED
   This is changing the value of the upper field after it is used but it is used again in other tests so this change causes incorrect behavior to be caught.

158 - Intersects

3. changed conditional boundary -> SURVIVED
   Survives because there is no intersect test that tests what should be returned when b0 is lower than lower and b1 is on the lower bound.

4. replaced boolean return with false for org/jfree/data/Range::intersects -> KILLED
   Gets killed because there is a test that expects true from this return.

189 - Constrain

5. removed call to org/jfree/data/Range::contains -> SURVIVED
   This is equivalent because if it isn't higher or lower than the bounds then it returns the same value. Does the same thing as contains without calling it.

6. removed conditional - replaced equality check with false -> KILLED
   There are tests that check if a value inside the bounds will not get changed which it will if this is always false.

303 - expandToInclude

7. Incremented (a++) double local variable number 1 -> KILLED

If you increase the local variable after it was used the first time it will change the result because it is used again.

8. Incremented (a++) double local variable number 1 -> SURVIVED
   If you increase the local variable after it was used the second time it will survive because it is not used after this time.

9. Incremented (a++) double local variable number 3 → SURVIVED
   The value is recorded then modified and never used again thus it doesn't affect the outcome

10. removed call to java/lang/IllegalArgumentException:: → SURVIVED
    Need to check for a specific exception

Report all the statistics and the mutation score for each test class

Original Assignment 3 Test Cases:

- Mutation Coverage
  - DataUtilities:
    - 87% - 596/687
  - RangeTest:
    - 71% - 894/1259

Assignment 4 Test Cases:

- Mutation Coverage
  - DataUtilities:
    - 93% - 639/687
  - RangeTest:
    - 82% - 1032/1259

Sample Test Cases:

- Line Coverage
  - DataUtilities:
    - 80% - 64/80
  - Range:
    - 93% - 96/103
- Mutation Coverage
  - DataUtilities:
    - 71% - 485/687
  - Range:
    - 66% - 826/1259

Analysis drawn on the effectiveness of each of the test classes

Originally, the mutation coverage for DataUtilitiesTest and RangeTest classes were already quite good at 87% and 71%. Once the tests were revamped to increase the mutation coverage, DataUtilitiesTest was increased to 93% and RangeTest to 82%. Considering that these numbers include a fair amount of equivalent mutations, which should not be considered, the DataUtilitiesTest has close to a hundred percent coverage, while the RangeTest is likely closer to 90%. Thus, it can be said both test classes are highly effective.

A discussion on the effect of equivalent mutants on mutation score accuracy

As mentioned above, equivalent mutants should not be considered when looking at mutation scores. However, it is not always possible to ignore them, especially when using an automated system that can not distinguish between equivalent and non-equivalent mutations. This causes the mutation score to decrease considerably and to decrease inaccuracy. Moreover, the mutation score will never be 100% because there will be some equivalent mutation that can't be removed.

A discussion of what could have been done to improve the mutation score of the test suites

The strategy to increase the mutation score for the test suites was to run the mutation test and then look at each function and see where mutations survived. Then if a new test could be created or a test could be edited to kill the mutation the change was made to the test suite. If there is an input that exists after the function is completed and a mutation changes the values of this input a test could be added to check that the function does not edit the values of the input. If a conditional boundary mutation survived then a new test could be added to test the boundary so that it passes before the mutation and fails after. If a mutation changed a return value to always be true or always be false test cases could be added that expect the return value to be true and false killing the mutation. By looking at what mutations survived more tests could be developed to kill these mutations by looking at what exactly the mutation was.

Why do we need mutation testing? Advantages and disadvantages of mutation testing

Mutation testing allows us to detect and fix loopholes in the test data and the test suite which are difficult to find by other testing methods. This methodology allows testers to detect ambiguous code, helps find defects early in the development cycle and facilitates complete code coverage. Additionally, mutation testing helps developers develop confidence in their test suites. Because it's impossible to tell whether a test suite is unable to find bugs because there are no bugs or because the test suite is ineffective, it is easy to feel uncertain about the completeness of your program even after spending time developing a test suite. Mutation

testing helps to give certainty to the effectiveness of the test suite, and therefore to the completeness of the tested system.

Large codebases create large amounts of mutants which can be a time consuming process and expensive. Mutation testing is a complex and long process which has to be automated and is very difficult to do manually. Without automated tools it is tough to do it properly. It requires access to the source code in order to produce mutants.

Explain your Selenium test case design process

First we came up with a list of features that we thought should be tested. Then for each feature we listed the requirements. We recorded the test cases keeping in mind these requirements and checked if they run as intended. Assertions and checkpoints were added to verify the functionality of the features. For tests that required user input and data, other tests were created to cover for invalid and boundary data.

Explain the use of assertions and checkpoints

Selenium can make assertions using the assert command. If the assert condition is not fulfilled in the case of assertions, the test case will be terminated. When we need to validate key functionality, we utilize Assert. If the test fails, the execution of subsequent statements becomes meaningless. As a result, the test procedure is terminated as soon as it fails.

Selenium can make checkpoints using the verify or wait for command. Checkpoints are soft assertions, and if they are not met, the test continues without interruption. WaitFor may also be used as a checkpoint in a test to see if the current state corresponds to the expected value. A timeout error will be shown if this command fails.

How did you test each functionality with different test data

For features that used external data, valid tests were created first to insure that the feature works properly. Then the test was replicated for different sets of valid and invalid input data. The checkpoint and assertions were modified to match the correct output like error message for invaild or missing input.

Discuss advantages and disadvantages of Selenium vs Sikulix

Selenium is a web application testing tool which doesn’t support desktop applications. It uses locators to verify and test web apps, it uses the underlying source code to achieve this therefore it can also be used to test things you can't see or are hidden. It has browser support for writing and running tests but it also can be used in various programming languages.

Sikulix is a tool for image-based automation which can also be used for GUI testing. It is restricted to what is seen on screen and may only be used to test what is visible to the user.

Since it is an image based tool it can be used on any GUI application. It is more complex and harder to write tests in compared to Selenium.

How the team work & effort was divided and managed

Ryan Sommerville

- Original mutation measurement for DataUtilities and Range
- Worked on increasing DataUtilities mutation score by developing test cases.
- Worked on the lab report.

Kaumil Patel

- Mutation scores for demo test cases
- Developed Selenium test cases
- Worked on increasing DataUtilities mutation score by developing test cases.
- Worked on the lab report.

Quinn Ledingham

- Developed Selenium test cases
- Worked on increasing Range mutation score by developing test cases.
- Worked on the lab report.

Difficulties encountered, challenges overcome, and lessons learned

Understanding what the description of a mutation was saying was difficult. Knowing exactly what the mutation looked like was hard to figure out from reading the mutation logs.

Comments & feedback on the lab itself

Because there were many equivalent mutations, we were unable to raise the DataUtilities mutation score by ten percent. We were only able to achieve an increase of six percent before concluding that the rest were all equivalent mutants.

Otherwise, the lab was fairly straightforward. The software used was easy to find and fairly self explanatory, and the effectiveness of mutation testing was clearly shown.