

## SENG 438 - Software Testing, Reliability, and Quality

### Lab. Report #4 – Mutation Testing and Web app testing

Group #	1
Student Names:	Huda Abbas
	Nuha Shaikh
	Lubaba Sheikh
	Rajpreet Gill

## 1 Introduction

This lab enabled the team to concentrate on fully grasping the principles of mutation testing and GUI testing. Throughout part one, we learned how mutants are injected into the code, using a testing tool, and learned how to properly interpret what all of the mutants imply. We also learned how to use this knowledge to design new test cases to improve the overall quality of the test suite. Subsequently, in the second part of the lab we focused on GUI testing and learned an automation approach that demonstrated how the record and replay functionality works. Specifically, we used Selenium to achieve the GUI testing automation which is a web interface testing tool.

Along with the fundamental automation testing knowledge the lab provided, it also gave us an essence into the strength and influence teamwork offers. Through strong collaboration and cooperation as well as by employing effective teamwork capabilities, we all ensured to maximize our expertise by taking part in active discussions throughout the lab and providing each other guidance and feedback frequently. Overall, this lab was a success, and the following report captures all of the components required for this lab assignment. The lab also provides reasoning and conclusions on how the requirements set in the lab document were met.

## 2 Analysis of 10 Mutants of the Range class

Analysis of at least 10 mutants produced by Pitest for the Range class, and how they are killed or not by your original test suite

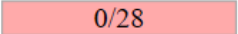
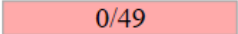
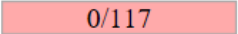
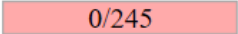
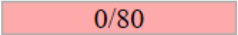
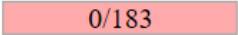
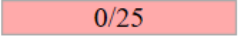
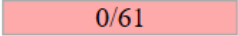
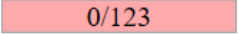
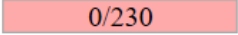
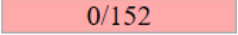
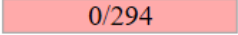
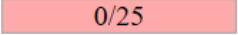
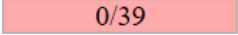
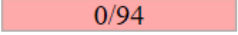
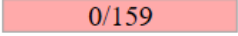
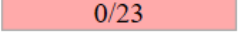
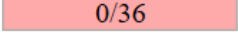
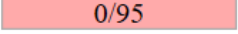
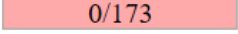
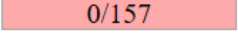
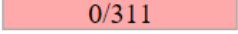
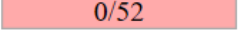
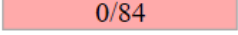
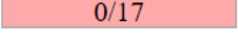
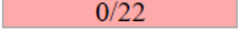
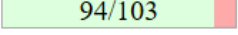
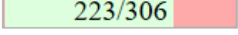
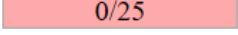
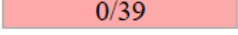
Mutant Number	Line Number	Mutation Analysis	Status
1	90	Changed conditional boundary - changed	Killed

		lower > upper in the Range constructor	
2	123	Replaced double subtraction with addition. Added this.upper + this.lower for the getLength() method	Killed
3	144	Removed conditional - replaced comparison check with false for the contains method	Killed
4	223	Removed call to Math.min in the combine method	Killed
5	306	Mutated return of Object value for the expandToInclude method to ( if (x!= null) null else throw new RuntimeException	Killed
6	158	Substituted 0 with 1 - changed the return statement from b0 to b1 in the intersects method	Alive
7	255	Negated conditional - switched it to true for the combineIgnoringNaN	Alive
8	281	Removed call to Double.isNaN() in the max method	Alive
9	448	Replaced return of integer sized with (x == 0 ? 1.0) for isNaNRange()	Alive
10	387	Substituted 0.0 with 1.0 for the shiftWithNoZeroCrossing	Alive

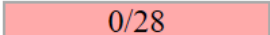
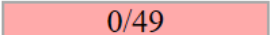
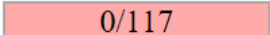
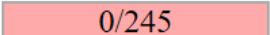
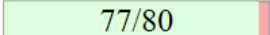
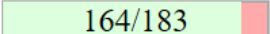
### 3 Report all the statistics and the mutation score for each test class

#### Running Mutation Tests on own test case

Range.java

Name	Line Coverage	Mutation Coverage
<a href="#">ComparableObjectItem.java</a>	0%  0/28	0%  0/49
<a href="#">ComparableObjectSeries.java</a>	0%  0/117	0%  0/245
<a href="#">DataUtilities.java</a>	0%  0/80	0%  0/183
<a href="#">DefaultKeyedValue.java</a>	0%  0/25	0%  0/61
<a href="#">DefaultKeyedValues.java</a>	0%  0/123	0%  0/230
<a href="#">DefaultKeyedValues2D.java</a>	0%  0/152	0%  0/294
<a href="#">DomainOrder.java</a>	0%  0/25	0%  0/39
<a href="#">KeyToGroupMap.java</a>	0%  0/94	0%  0/159
<a href="#">KeyedObject.java</a>	0%  0/23	0%  0/36
<a href="#">KeyedObjects.java</a>	0%  0/95	0%  0/173
<a href="#">KeyedObjects2D.java</a>	0%  0/157	0%  0/311
<a href="#">KeyedValueComparator.java</a>	0%  0/52	0%  0/84
<a href="#">KeyedValueComparatorType.java</a>	0%  0/17	0%  0/22
<a href="#">Range.java</a>	91%  94/103	73%  223/306
<a href="#">RangeType.java</a>	0%  0/25	0%  0/39

DataUtilities.java

Name	Line Coverage	Mutation Coverage
<a href="#">ComparableObjectItem.java</a>	0%  0/28	0%  0/49
<a href="#">ComparableObjectSeries.java</a>	0%  0/117	0%  0/245
<a href="#">DataUtilities.java</a>	96%  77/80	90%  164/183

## 4 Analysis drawn on the effectiveness of each of the test classes

The DataUtilities test suite is a lot more effective and thorough compared to the Range test suite since it was able to kill 164/183 mutants. This is because it had a higher line coverage as it had less methods overall so the test suite was a lot more effective. By increasing our mutation score we were able to get the mutation coverage from 90% to 99%. We weren't able to achieve full 100% coverage due to equivalent mutants that cannot be killed. The Range test suite killed 223/306 mutants which resulted in only 73% coverage. This number was a lot lower than the DataUtilities.java file mainly due to a lack of coverage as some methods were missed in the test suite as it was a larger class.

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

### Finding equivalent Mutants.

**Method.** Our process for detecting equivalent mutants manually is to look only for those mutants that have survived for all test cases every time even when the test set has been altered. Looking further, we would likely be able to identify mutants that are identical to that of the original program from this reduced mutant list. For our specific case the method we used was looking at lines highlighted red and then exploring the exact change made by the mutant to see if it was equivalent to our original program. Try to kill a surviving mutant, if not possible can assume its equivalent.

**Benefits.** A benefit this method presents is that it allows us to detect equivalent mutants without having to manually observe every single change created.

**Disadvantages.** This method requires individually scanning mutants and considering our different test cases for us to enable testing the program with different test cases. This method relies purely on observation and some equivalent mutants may not be caught due to them being missed.

**Assumptions.** The assumption that is being formed is that the same mutants will be used across all of the test cases.

### Results

Equivalent Mutant	Line Number & Method	Mutation Analysis	Explanation
1	Line 104 from clone in DataUtilities.java	Change less than to not equal to for "i < source.length" in the for loop	Changing i < source.length to i != source.length is equivalent as for int i = 0 both will fail when i is equal to

			source.length.
2	Line 89 from equal in DataUtilities.java	Substituted 1 with -1 for return true	This substitution is equivalent to returning true which is the same as the original program so it cannot be killed.

Equivalent mutants negatively impact the overall mutation score accuracy of the test class as they are mutants that are equivalent to the original program and therefore can not be killed by our test suite. This is the reason why we were only able to achieve 99% mutant coverage with our new test suite for DataUtilities.java as the final 1% can be assumed to be equivalent mutants.

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

**Added the following new test cases to increase the mutation scores by 10%**

DataUtilities.java

- public static boolean equal(double[][] a, double[][] b)
  - public void test\_equal\_equalToLessOrEqual81()
    - This covers mutant “equal to less or equal”. Tests that the length of a is greater than b so that the mutant becomes false and kills the test
  - public void test\_equal\_equalToGreaterOrEqual()
    - This covers mutant “equal to greater or equal”. Tests that the length of a is smaller than b so that the mutant becomes false and kills the test
  - public void test\_equal\_removedConditional81()
    - This covers mutant “removed conditional”. Tests that b is null because the mutant made the condition false so it has to be true in order to kill it
  - public void test\_equal\_removedConditional78()
    - This covers mutant “removed conditional”. Tests that the length of a and b are not equal because the mutant made the condition false so it has to be true in order to kill it
  - public void test\_equal\_substitutedZeroWithOne84()
    - This covers mutant “substituted zero with one”. Tests that the input variables a and b are not equal because the first two elements dont match but the other elements match. Since i = 1 is the mutant, it skips the first element hence it thinks the two arrays are equal when they are not hence it kills the mutant
- public static double[][] clone(double[][] source)
  - public void test\_clone\_replaceEqualityCheckWithTrue105()
    - This covers mutant “replace equality check with true”. Tests that first two of source is null because the mutant made the condition true so it has to be false in order to kill it
- Public static double calculateColoumnTotal(Values2D data, int column, int[] validRows)

- public void test\_calculatedColumnTotal\_NullPermitted()
  - This covers mutant “removed call to org/jfree/chart/util/ParamChecks::nullNotPermitted” for line 124 which is “ParamChecks.nullNotPermitted(data, "data");”
- Public static double calculateRowTotal(Values2D data, int row)
  - public void test\_calculatedRowTotal\_NullData()
    - This covers mutant “removed call to org/jfree/chart/util/ParamChecks::nullNotPermitted” for line 175 “ParamChecks.nullNotPermitted(data, "data");”
  - public void test\_calculatedRowTotal\_ColCountGreaterThanCol()
    - This covers mutant “changed conditional boundary” for line 155 if (row < rowCount)
- public static KeyedValues getCummulativePercentage(KeyedValues data)
  - public void test\_getCumulativePercentage\_vGetValueIsNull()
    - This covers mutant “negated conditional ” in line 267 and 274 which are if (v != null)

#### Range.java

- Public double getCentralValue()
  - test\_getCentralValue()
    - Tests the getCentralValue method with valid inputs
    - This increased the coverage of this method which resulted in killing all the mutants for this method
- public boolean intersects()
  - test\_intersects\_replacedComparisonCheckWithFalse157()
    - Test lower bound edge of the intersect function where this.lower is greater than b0. This also covers the mutant “Substitutes 0 with 1”
  - test\_intersects\_replacedComparisonCheckWithTrue158()
    - Tests lower bound edge of intersect where replace check of return (b1 > this.lower)
  - test\_intersects\_replacedEqualityCheckWithTrue189()
    - Tests the bug replaced equality check of !contains(value) with true
- combineIgnoringNaN
  - test\_combineIgnoringNaN\_return\_range1()
    - Tests the method with the first range object having valid values and the second range object being null
    - This tests the mutation that negated the conditional value
  - test\_combineIgnoringNaN\_returnnull()
    - Tests the method with the first range object being null and the second range object having a valid lower and upper bound
    - This tests the mutation that killed the mutant the altered the return object
  - test\_combineIgnoringNaN\_doublevalues\_returnNewRangeObject()

- Tests the method with both range objects having a lower bound and an upper bound
  - This tests the mutation that killed the “removed call” and “mutated return”
- test\_combineIgnoringNaN\_returnNull\_ifStatement()
  - Tests the method with both upper and lower bounds, for both range objects being NaN.
  - This tests the mutation that killed the mutated return of object value
- test\_combineIgnoringNaN\_ifStatement()
  - Tests the method with the first range object having NaN values for both lower and upper bounds and the second range object having a valid upper and a lower bound
  - This tests the mutation that kills the negated conditional statement and replaced equality checks in the if statement

## New Mutation Score

Range.java

<a href="#">KeyedObjects2D.java</a>	0%	0/157	0%	0/311
<a href="#">KeyedValueComparator.java</a>	0%	0/52	0%	0/84
<a href="#">KeyedValueComparatorType.java</a>	0%	0/17	0%	0/22
<a href="#">Range.java</a>	96%	99/103	83%	253/306
<a href="#">RangeType.java</a>	0%	0/25	0%	0/39

DataUtilities.java

## Breakdown by Class

Name	Line Coverage	Mutation Coverage
<a href="#">ComparableObjectItem.java</a>	0% 0/28	0% 0/49
<a href="#">ComparableObjectSeries.java</a>	0% 0/117	0% 0/245
<a href="#">DataUtilities.java</a>	99% 79/80	99% 181/183

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

**Advantages:** it allows you to check and verify the strength of your written tests to see if they will kill mutants. This helped us identify weak tests we had written and improve our test suite to

kill more mutants. Additionally, it helps with error detection creating more reliable and stable systems after testing

**Disadvantages:** the large number of mutants can lead to a confusing experience in testing each variation as it can result in equivalent mutants and ones that are semantically incorrect. Additionally, it can be expensive and time consuming. Complex mutations are also difficult to implement and not usually possible with a tool like Selenium.

## 8 Explain your SELENIUM test case design process

We analyzed the sportchek website first to determine some of the major functionalities of the website. We then determined the tests that we need and the verification that is required for each test to pass. Then we used the record function on selenium to automate each test. For each test we tried to incorporate tests with invalid and valid test data where possible.

## 9 Explain the use of assertions and checkpoints how did you test each functionality with different test data

Test #	Functionality	Test Data	Output/Defects	Tester
1	Writing a Review for a Product	<i>Valid data</i> <u>Test 1</u> <ul style="list-style-type: none"><li>- Product: footwear Reebok Men's Zig Dynamica Running Shoes</li></ul> <i>Invalid data</i> <u>Test 2</u> <ul style="list-style-type: none"><li>- Feedback form very short</li></ul>	No defects. Verifying that the end element says "Your review was submitted!"	Nuha Shaikh
2	Add to cart	<i>Valid data</i> <u>Test 1</u> <ul style="list-style-type: none"><li>- Add 2 items to cart</li></ul> <u>Test 2</u> <ul style="list-style-type: none"><li>- Add an item to cart</li></ul> <u>Test 3</u> <ul style="list-style-type: none"><li>- Remove an item from the cart</li></ul>	No defects. Verifying using assertions that element quantity cart in cart is 1.	Lubaba Sheikh
3	Choose your store	<i>Valid data</i> <u>Test 1</u> <ul style="list-style-type: none"><li>- City: Calgary</li></ul> <u>Test 2</u>	No defects. Unable to add explicit verification to store selection as element not found using Selenium.	Nuha Shaikh



		<ul style="list-style-type: none"> <li>- City: Toronto</li> </ul> <i>Invalid data</i> <u>Test 3</u> <ul style="list-style-type: none"> <li>- City: Harare</li> </ul>		
4	SearchBar	<i>Valid data</i> <u>Test 1</u> <ul style="list-style-type: none"> <li>- Searched for “jackets”</li> </ul> <u>Test 2</u> <ul style="list-style-type: none"> <li>- Searched for “shirts”</li> </ul>	No defects. Verifying the text that appeared after we search “jackets” in the search bar and “shirts” in the search bar.	Rajpreet Gill
5	Select Category	<i>Valid data</i> <u>Test 1</u> <ul style="list-style-type: none"> <li>- Selected “Women” and then “running”</li> </ul> <u>Test 2</u> <ul style="list-style-type: none"> <li>- Selected “Men” and then “running”</li> </ul>	No defects. Verifying the text that appeared after selecting the “women” category and then “running” as well as, the text that appeared after selecting “men” and then “running”.	Rajpreet Gill
6	Pic Upload	<i>Valid data</i> <u>Test 1</u> <ul style="list-style-type: none"> <li>- Drag &amp; Drop JPG image</li> </ul>	No defects. Verifying that page says “SUCCESS!” at the end. To assert that the image was correctly uploaded to the gallery a checkpoint was added at the end.	Huda Abbas
7	Social Media Footer Hyperlinks	<i>Valid data</i> <u>Test 1</u> <ul style="list-style-type: none"> <li>- Youtube</li> </ul> <u>Test 2</u> <ul style="list-style-type: none"> <li>- Facebook</li> </ul> <u>Test 3</u> <ul style="list-style-type: none"> <li>- Twitter</li> </ul> <u>Test 4</u> <ul style="list-style-type: none"> <li>- Instagram</li> </ul>	No defects.  Verifying the title of each social media page.	Lubaba Sheikh
8	View Product Description	<i>Valid data</i> <u>Test 1</u> <ul style="list-style-type: none"> <li>- Viewing description of product on homepage</li> </ul> <u>Test 2</u> <ul style="list-style-type: none"> <li>- Viewing description of product from category dropdown</li> </ul>	No defects. Verifying that description is present and verifying that text for product matches.	Huda Abbas

For functionality 6 only one valid set of test data was used as other possibilities like sending no picture and uploading using the browse files button are invalid through Selenium and were therefore unable to be tested.

## 10 Discuss advantages and disadvantages of Selenium vs. Sikulix

Table 1.1. GUI Testing Tools Comparisons

	Advantages	Disadvantages
<b>Selenium</b>	<ul style="list-style-type: none"><li>- Can export files into C#, Java, JavaScript, Python, Ruby</li><li>- Easy to integrate with simple Google Chrome extension</li><li>- Easily add automated verification points</li><li>- Only for testing websites/web applications</li><li>- Can easily click on unknown elements to access that target</li></ul>	<ul style="list-style-type: none"><li>- Login and Register test cases does not accustom to autofill feature and gets stuck</li><li>- Feedback form fails to get past rating selection, cannot recognize the numbers</li><li>- Failed code error message is displayed saying “Not allowed” when trying to browse your local computer for a file</li></ul>
<b>Sikulix</b>	<ul style="list-style-type: none"><li>- Can be used for desktop applications unlike Selenium which is only for websites</li><li>- Can easily click on unknown elements to access that target</li></ul>	<ul style="list-style-type: none"><li>- Can only export Jython/Python and Ruby</li><li>- Large jar file download is required and is not easily adaptable to a web browser</li><li>- Requires system settings changes for modifiable access</li></ul>

## 11 How the team work/effort was divided and managed

Huda: Selenium tests pic upload and view product description, increased mutation coverage tests for method intersects in class Range

Nuha: Selenium tests for adding a review for a product and choose your store, increased mutation coverage tests for calculateColoumnTotal, calculateRowTotal, getCummulativePercentage in class DataUtilities

Rajpreet: Selenium tests for search bar and select category, increased mutation coverage tests for combineIgnoringNaN and getCentralValue() methods in class Range

Lubaba: Selenium tests for add to cart and social media hyperlinks, increased mutation coverage tests for equal and clone in class DataUtilities

Additionally, we meet regularly for work sessions and to keep everyone on the same page and help each other with any struggles that we were having. The work was divided evenly and everyone contributed fairly.

## **12 Difficulties encountered, challenges overcome, and lessons learned**

Some of the difficulties encountered were that we had a hard time dealing with the dynamic nature of the sportchek website, this was out of our control and thus it was difficult to keep our Selenium test cases updated. Additionally, the large waiting times for running the PIT test decreased our efficiency as a team significantly so we had to strategically run the PIT test instead of doing it incrementally each time. The challenge we encountered was that we had a hard time with generating the PIT test results for all devices hence we had to collaborate together and use devices that achieved the software functionality this lab required. The lessons we learned were that for mutation testing to kill mutants the conditions required are the test must reach mutated statement, the test input data should infect program state by causing different program states for the mutant and the original program and the incorrect program state must propagate to the program's output and be checked by the test.

## **13 Comments/feedback on the lab itself**

Overall, the lab allowed us to understand the core principles of GUI testing and mutation testing and how it is beneficial to your testing suite. The lab instructions and goals required more clarity as there was lots of questions that had to be asked to ensure what we did was what was actually required. Some feedback would be to PIT test because it was outdated and required lots of debugging, time to run and incompatibilities with the operating system. Additionally, the setup was time consuming and with both parts the lab felt a little too long and covered too much material.