# Project Report

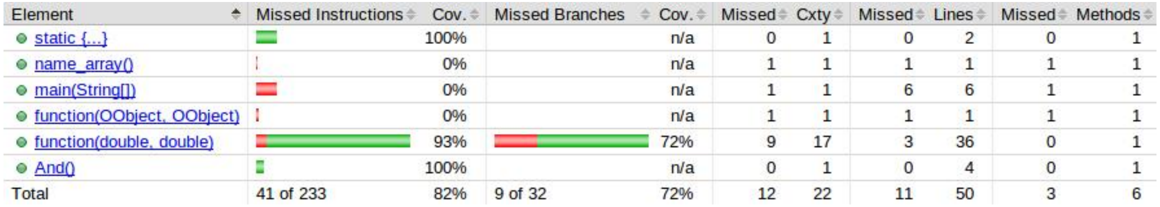
## SENG437 – Software Testing

George Dimitrov, Yasir Al-Bender, Cory Ebner & Will Nguyen

### ABSTRACT

We are all human beings and human beings can commit errors. Some of these errors do not impact much in our day to day life and can be ignored, whereas others can have adverse effects. In software development human errors can cause faults. These faults lead to failures which in turn leads to an incident. Some incidents are so severe that they can break the whole system or software. In software development you need to take care that such errors are caught. Ideally these errors are caught well in advance before deploying the system/software in production environment or are caught as soon as possible after release. Testing plays an important role in today's System Development Life Cycle. This document outlines the test plan and testing strategy followed while performing software testing for the Java Scientific Calculator application.

## Table of Contents



| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject, OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| function(double, double) | 🟥🟩 | 93% | 🟥🟩 | 72% | 9 | 17 | 3 | 36 | 0 | 1 |
| And() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| Total | 41 of 233 | 82% | 9 of 32 | 72% | 12 | 22 | 11 | 50 | 3 | 6 |

# 1. Framework, discussions, recommendations and future work

## 1.1 Framework

During testing, we followed a systematic procedure to uncover defects and faults in the application.

### 1.1.1 Framework expectations

Do to our limited experience with the application and testing methodologies it was determined that if testing does not uncover any errors then the test process was not effective.

### 1.1.2 Methods used to evaluate the application

We performed the following testing methods:

- Black box testing
- White box testing
- Mutation testing
- GUI testing / Integration testing

For each testing method we:

- Created test code or test cases. The use of an automated test framework to create the test code was used when possible.
- Ran the test code or test cases. The use of an automated test framework to create the test code was used when possible.
- Logged results from the tests into spreadsheets. Spreadsheets maintained on drive.google.com to ensure ease of access for all testing members and to provide redundancy in the event of data loss.
- Created and ran additional tests. Developed additional tests for testing of the application based on the results of the initial tests.

### 1.1.2 Tools used

- Integrated development environment
    - Eclipse 3.6+
- Black-box Testing
    - JUnit 4 (Eclipse Plug-in)
- White-Box Testing
    - EclEmma 2.2.1 (Eclipse Plug-in)
- Mutation Testing
    - PITClipse 0.32 (Eclipse Plug-in)
- Bug Tracking System

- Bugzilla 3.6.5 (Mylyn Plug-in)

### 1.1.4 Reporting Results

Results were reported in milestones. Each testing method was considered a milestone resulting in five milestone reports being reported. Each milestone contained the testing processes, results, and recommendations based on the method of testing undertaken during that milestone.

## 1.2 Discussion

During the three month testing period over 205 tests were performed. These tests focused on the areas of the application that were the most testable and that would allow us to follow our testing framework. During our testing 22 bugs were found.

The testing team feels that the approach worked really well, specifically in the first few stages of the testing cycle. Black-box testing, in combination with White-box testing was the most effective aspect at finding errors and faults. Specifically in this application, many, if not all, bugs were found in this stage. Mutation testing, GUI and integration testing have their place in software testing but the testing team did not find that they were especially useful testing the Java Scientific calculator.

Overall the testing team feels that test plan and framework resulted in software testing that was effective.

## 1.3 Recommendations and Future Work

The bugs found represent only a portion of the possible bugs in the software application should not be taken as an exhaustive list. While the testing team tried to minimize the chance of errors being caused by outside sources such as other software running on the test machines, the bugs produced are not guaranteed to be a direct cause due to a problem with the application itself.

It is the recommendation of the testing team that the results displayed in this document be verified by a separate outside source to ensure that the results are correct before any action is taken based on the results. It is also recommended that additional tests be ran on aspects not covered by the testing outlined in this document to ensure a higher confidence in the software as software testing can only show the presence of bugs and not their absence.

# 2. Test Plan

## 2.1 Objectives

The following test plan for the Java Calculator will support the following objectives:

i. To detail the activities required to the preparation and the conduct of the test.
ii. To communicate the responsibilities and the schedule of the test plan to the test team.
iii. To define sources of information used in preparation of the test.
iv. To define the tools needed within the test.

## 2.2 Background

The calculator is based in Java and is a replacement scientific calculator so that users can get extra functionality. The basic calculators don't always offer everything a user may need and java allows the program to be accessible for all users on any platform.

## 2.3 Scope

The test plan will be covering the main functionality of the Java Calculator. This entails testing the main classes that use mathematical utilities in performing its purpose. One of the non-functional features that will be tested and evaluated is the graphical user interface (GUI).

## 2.4 References

Java Scientific Calculator Official Homepage - http://jscicalc.sourceforge.net/
JUnit Official Homepage/Wiki - http://junit.org/

## 2.5 Test Items

### 2.5.1 Program Modules

The program modules to be tested will be identified as follows:

| **Type** | **Library** |
|---|---|
| Executable Code | jscicalc2-0.5.src.jar |
| Source Code | pobject |

### 2.5.2 User Procedures

The Basic operations procedures specified in the Java Scientific Calculator website (http://jscicalc.sourceforge.net/) and in Javadoc will be tested.

### 2.5.3 Operator Procedures

The system test includes the procedures specified in the Java Scientific Calculator website (http://jscicalc.sourceforge.net/) and in Javadoc.

## 2.6 Features to be Tested

*Basic Calculator Functions*

- Addition
- Numbers

*Ways of Interaction*

- GUI

### *Advanced Calculator Functions*

- Exponential Functions
- n$^{th}$ Root
- Trigonometric Functions (ie. Sine, Cosine…etc.)
- Logarithmic Functions
- Combination/Permutations

## 2.7 Features not to be Tested

### Hidden functionality

Since the user is not using this functions like "`public string tooltip()`" located in `CalculatorButton.java`, we don't need to test their functionality. Only a developer would notice this issue and the program would have noticeable bugs if these types of functions were not functional. The developers of the application aren't even sure if this exact function is used in the final product so if we tested it we might be testing a dead piece of code, which would be a waste of time.

### Dead Functions

Any functions the developers find no use for and have managed to stay in the code for fear of it potentially getting called somewhere. The previous example can be used in this case.

## 2.8 Approach

The program testing team will use the program documentation along with a created Javadoc to prepare all test designs and procedures.

### 2.8.1 Accuracy Testing

Accuracy of the answers given by the various calculations will be tested against the accuracy requirements as stated in the programs documentation.

### 2.8.2 GUI Testing

Combinations of manual and automated GUI testing will be used to determine the quality of the GUI.

### 2.8.3 Correctness Testing

The correctness of the calculations provided by the program will be tested using a pre-existing known list of answers to ensure the program is calculating as intended as outlaid in the documentation provided. Several comparable programs will be used to determine a list of known answers.

### 2.8.4 Constraints

All testing must be done by the schedule as provided in the document "SENG 437 - Project Timeline.pdf" located in Blackboard.

## 2.9 Item Pass/Fail Criteria

In a program such as this, the calculator has one main criteria of pass/fail: it must produce the correct answer to the calculation inputted. This must remain consistent for all mathematical queries that are inputted into the program.

The other main criterion for a calculator is usability. The GUI of the calculator must be able to display the correct answers and must be responsive to user interaction. State changes must be obvious to the user.

## 2.10 Suspension Criteria and Resumption Criteria

### 2.10.1 Suspension Criteria

i. The inability to interpret the code, due to lack of documentation, will cause the temporary cessation of testing activities.

ii. When a new iteration is released, there will be a temporary cessation of testing activities.

### 2.10.2 Resumption Criteria

i. When documentation is found, and after a period of interpretation has occurred, testing can then be resumed.

ii. When documentation of bug fixes or updates is found, and after a period of interpretation has occurred, testing can then be resumed.

## 2.11 Test Deliverables

The following documentation will be submitted to the Department Manager by the test team after test completion.

### Test Documentation

- Test Plan
- Test Case Specifications
- Test Results
- Test Procedure Specifications
- Test Logs
- Bug Report
- Test Summary Report

### Test Data

i. Copies of input and output test files will be submitted along with the Test Documentation to the Department Manager

## 2.12 Testing Tasks

See Appendix A for details.

## 2.13 Environmental Needs

### 2.13.1 Hardware

Testing will be conducted on machines that can run Java 2 Runtime Environment 1.5. This assessment was made from the hardware requirements listed on the program's homepage. The minimum requirements for Java 2 Runtime Environment 1.5 are: Windows 2000/XP/2003/Vista, IE 5.5/6.x or Mozilla 1.4+ or Firefox.

### 2.13.2 Software

#### *2.13.2.1 Operating System*

The primary operating systems that will be used is Windows 8.1 and Mac OSX Maverick 10.9.

#### *2.13.2.2 Tools*

- Eclipse – A integrated development environment.
- JUnit – A programming-oriented framework for Java will be used as the primary testing tool.
- EclEmma – A code coverage tool for Eclipse.
- PITClipse – A mutation testing plug-in for Eclipse.
- Bugzilla – A bug tracking system

## 2.14 Responsibilities

### 2.14.1 Program Testing Team

This team will be providing all technical testing expertise and will be conducting all manners of the test plan.

### 2.14.2 Test Manager

This person will be overseeing all activities of the test team and will ensure that the team meets all testing schedule milestones.

## 2.15 Schedule

See Attachment A for details.

### 2.16 Risks and Contingencies

To prevent interruptions that may be caused by computer hardware failures, all activities done by the team will be documented and backed up to the cloud storage systems available.

If a test team member is unable to fulfill their duties assigned, that member must communicate that to the team leader well in advance; those tasks will be assigned to another team member.

If a milestone will be missed, the team manager will attempt to modify the schedule in order to meet the most (if not all) objectives of the milestone missed.

# 3. Black-box Testing

## 3.1 Test-Case Design for Functional Requirements

Since this program is mainly a calculator, it remains logical only to use numbers as inputs. A variety of mathematical functions were identified and selected to ensure functionality of the calculator. The main requirements to test are correctness. This differs from accuracy in the sense that accuracy test for correct number of decimal places while correctness tests for the correct answer. Test cases can be divided into 3 main categories: Equivalence Partitioning, Boundary-Value Analysis and Error Guessing.

The procedure for testing functional requirements is as follows:

1. Determine the input domain of a function based on documentation and requirements.
2. Based on the input domain and function, identify characteristics in which to define partitions.
3. Based on partitions, identify which type of testing to use.
4. Create test cases based on partitions in JUnit.
   i. For Equivalence Partition – Ensure partition satisfy 2 conditions: Completeness and Disjoint
   ii. For Boundary-Value Analysis – Identify boundaries for each partitions and uses those values in testing
   iii. Error Guessing – Use prior mathematic knowledge and experience to use values in testing
5. Record test results in cloud-based spreadsheet.

For most functions, Equivalence Partition testing was utilized. Most if not all functions take the same types of inputs. The main partitions used:

- Large Positive Values
- Small Positive Values
- Large Negative Values
- Small Negative Values
- Zero
- Infinity

For more special cases of testing, like trigonometric functions, Boundary-value Analysis was used. However, the upper bounds in most cases resulted in the test failing. This is due to the max value being too great for the calculator to handle. Since it allows for 64-bit IEEE-754 format, it took that as input but it did not compute it properly. Still, upper bound and lower bound values were determined and used for trigonometric testing.

Lastly, bitwise and some logarithmic functions are tested using Error Guessing. Using prior knowledge about certain properties about these mathematic concepts, the correctness of the implementation can be tested. For example:

$$\ln e^{x} = x$$

This property was tested by using java.Math.log to get the expected answer. Then comparing that answer to the actual one received from the function.

### 3.2 Test Case Design for Non-Functional Requirements

The main Non-Functional requirements are accuracy and usability. Accuracy was tested by using a leniency of 0.1. Meaning actual results from the functions must not have more of a difference of 0.1 with the expected results. This was to ensure the precision of all outputs from the functions were mathematically exact. Although the leniency is 0.1, test cases can be tested to the 0.000000000000008 to test for accuracy according to the developer. All test failed upon changing the delta to the developer stated number so testing will need to be redone to with new numbers to check for accuracy.

Usability will be tested in a future date and was not examined in this milestone. The team concluded that usability of the calculator remained largely within the graphical user interface (GUI). Since, the GUI will be inspected more in depth in a later milestone, it was decided that the GUI will not be tested right now.

### 3.3 Report Results

For actual results, see Appendix B. – Black-box Testing Results.

### 3.4 Conclusions and Recommendations

The degree of effectiveness in the testing was very high considering it was the team's first attempt at using formal testing techniques. The preparation and definition of the partitions early in testing certainly helped synchronized the team. Another important factor was the independence and trust given to each tester. Each member of the team had freedom to choose which types of tests were to be performed. Although this factor was mainly positive, a downside to the flexibility given to each tester was inconsistent communication and inconsistent naming scheme. A more centralized approach may be taken for future testing.

It is a consensus among the team that Equivalence Partition was the best method for testing in a project like this one. The partitions defined were certainly a factor in the ease of testing experienced by the team. Error Guessing was only helpful if the tester had a large amount of knowledge regarding the function. It was effective in testing the correctness of a function if the tester knew what the function was base off. Boundary-Value Analysis was somewhat difficult to work with since it required prior knowledge about which numbers worked and which didn't within the boundaries of mathematical concepts used in the functions.

From the results, we can conclude that the calculator's functions were, for the most part, implemented correctly. However, the tests performed showed some precision problems with some functions.  Precision is a very important requirement especially when dealing with mathematics. The other interesting result was the Java Overflow error experienced when testing the Combination function. It is particularly intriguing because the program, itself, does not handle the error.

# 4. White-box Testing

## 4.1 Initial Code Coverage Results

Our test team consists of four members: George, Yasir, Cory and Will. Each member performed Black-Box testing on four functions of their choosing. The code coverage tool used was **EclEMMA**, a coverage measuring toolkit made for Java.

The following tables are a summary of the results of automated code coverage measurements done by each respective member. For each table, there are corresponding collections of screenshots taken of raw data located in Appendix A.

### 4.1.1 George's Initial Results

| **Class Name** | **Code Coverage** |
|---|---|
| Cube | 100% |
| Square | 100% |
| Inverse | 100% |
| Factorial | 100% |

### 4.1.2 Yasir's Initial Results

| **Class Name** | **Code Coverage** |
|---|---|
| Inverse Cosine | 100% |
| Cosine | 100% |
| Sine | 100% |
| Tangent | 100% |

### 4.1.3 Cory's Initial Results

| **Class Name** | **Code Coverage** |
|---|---|
| Combination | 100% |
| Cube Root | 100% |
| Logarithmic (Base 10) | 100% |
| Addition | 100% |

### 4.1.4 Will's Initial Results

| **Class Name** | **Code Coverage** |
|---|---|
| Natural Logarithm | 100% |
| Inverse Logarithm | 100% |
| AND | 93% |
| XOR | 90% |

## 4.2 Manual Coverage Results

To ensure that the code coverage results from the programs used are accurate, each member of the test team will select a single unit and perform manual code coverage calculations. The differences between the automated and manual code coverage calculations will be noted and discussed below:

### 4.2.1 George's Manual Results

The following calculations are for the `function(double x)` method in `factorial.java`

```
public double function( double x ){
        if(  x < 0 || Math.round( x ) - x != 0 ) ← 4 branches
             throw new ArithmeticException( "Factorial error" );
        else if( x == 0 ) ← 2 Branches
             return 1;
        else
             return x * function( x - 1 );
}
```

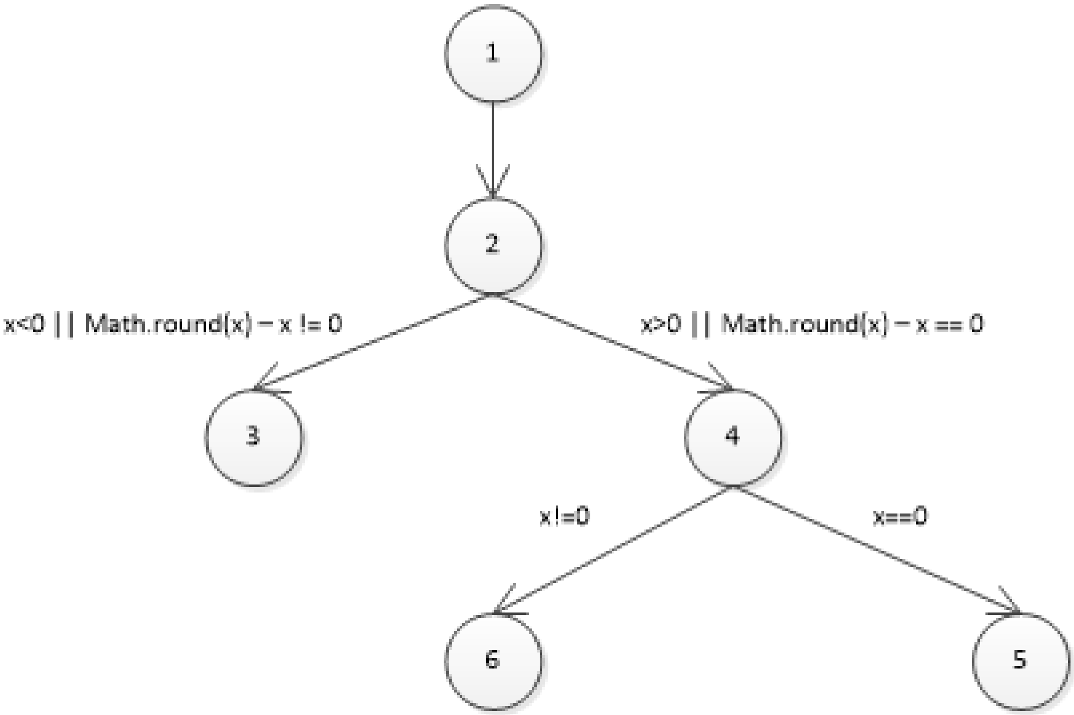


**Figure 1: Factorial Control Flow Graph**

**Condition Coverage:**

In the first line:

*if( x < 0 || Math.round( x ) - x != 0 )*

we have:

x<0 which has 2 outcomes, x<0 or x>0
we also have:
Math.round(x) - x != 0 or Math.round(x) - x == 0, this is another 2 outcomes for a total of 4 possibilities in the first line.
Next branch is:
*if(x==0)*
So either x==0 or x!=0, that's +2 outcomes for total branches.

In the end we have:
6 lines of code
2+2+2 = 6 branches

**Statement and Branch Coverage:**

- Test case 1 covers all branches, part 1(Testing with 5) covers nodes 1-2-4-6-5
- Part 2 (Testing with 0) covers nodes 1-2-4-5
- Part 4 (Testing 2.5) covers nodes 1-2-3, but this was a design fault by developer.
- Test case 2 covers nodes 1-2-3 by testing -5.

**Path Coverage:**
Those same tests used in statement coverage also accomplish path coverage for the entire function.

### 4.2.2 Yasir's Manual Results

The following calculations are for the `function(oobjectx)` method in `acos.java`

```
public OObject function( OObject x ){
             if( x instanceof Complex ){
                 Complex c = (Complex)x;
                 if( scale != 1 && StrictMath.abs( c.imaginary() ) > 1e-6 )
                          throw new RuntimeException( "Error" );
                 if( scale != 1 && StrictMath.abs( c.real() ) > 1 )
                          throw new RuntimeException( "Error" );
                 return c.acos().scale( iscale );
             } else {
                 return x.acos( angleType );
             }
}
```

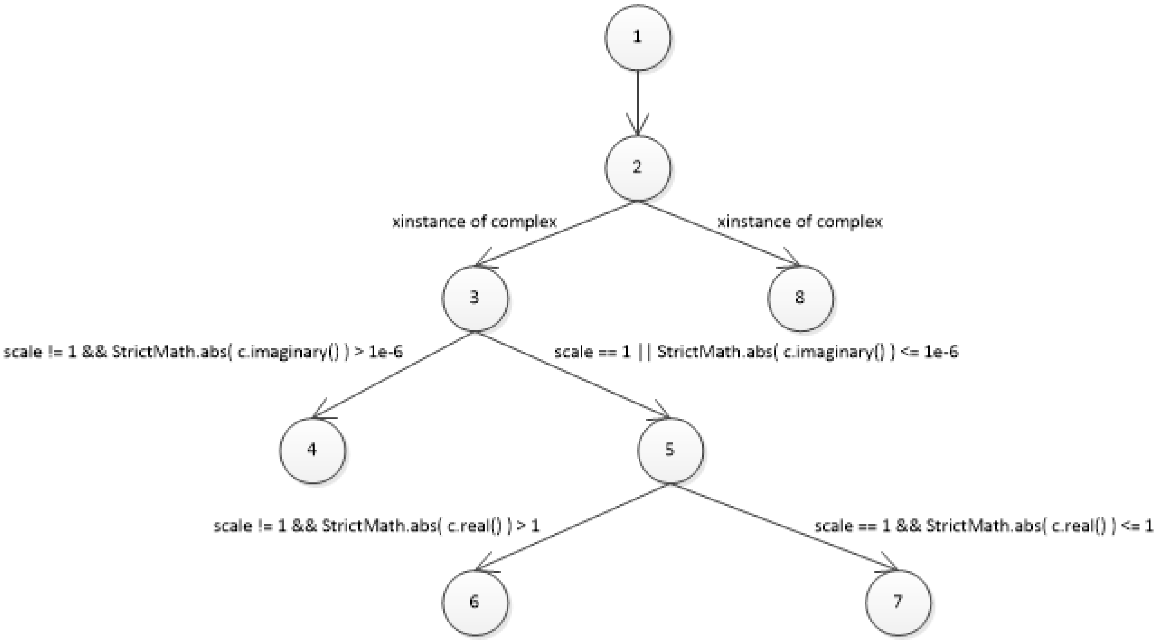

1
2
xinstance of complex
xinstance of complex
3
8
scale != 1 && StrictMath.abs( c.imaginary() ) > 1e-6
scale == 1 || StrictMath.abs( c.imaginary() ) <= 1e-6
4
5
scale != 1 && StrictMath.abs( c.real() ) > 1
scale == 1 && StrictMath.abs( c.real() ) <= 1
6
7

**Figure 2: Inverse Cosine Control Flow Graph**

### Condition Coverage:

- x instanceof Complex
- x ! instanceof Complex
- scale != 1 && StrictMath.abs( c.imaginary() ) > 1e-6
- scale == 1 || StrictMath.abs( c.imaginary() ) <= 1e-6
- scale != 1 && StrictMath.abs( c.real() ) > 1
- scale == 1 && StrictMath.abs( c.real() ) <= 1

All conditions are covered

### Path/Statement and Branch Coverage:

- testOObject() traverse nodes 1-2-8
- testComplexObject() traverses 1-2-3-5-7
- testErrorComplexObjectImagineryGreaterThan() traverses 1-2-3-4
- public void testErrorComplexObjectRealGreaterThan1() traverses 1-2-3-5-6

### 4.2.3 Cory's Manual Results

The following calculations are for the `function(double x, double y)` method in `combination.java`

```
public double function( double x, double y ){
             if(  x < 0 || Math.round( x ) - x != 0 )
                    throw new ArithmeticException( "Combination error" );
             if(  y < 0 || y > x || Math.round( y ) - y != 0 )
                    throw new ArithmeticException( "Combination error" );
             if( y == 0 )
                    return 1;
             else
                    return x  / y * function( x - 1, y - 1 );
}
```

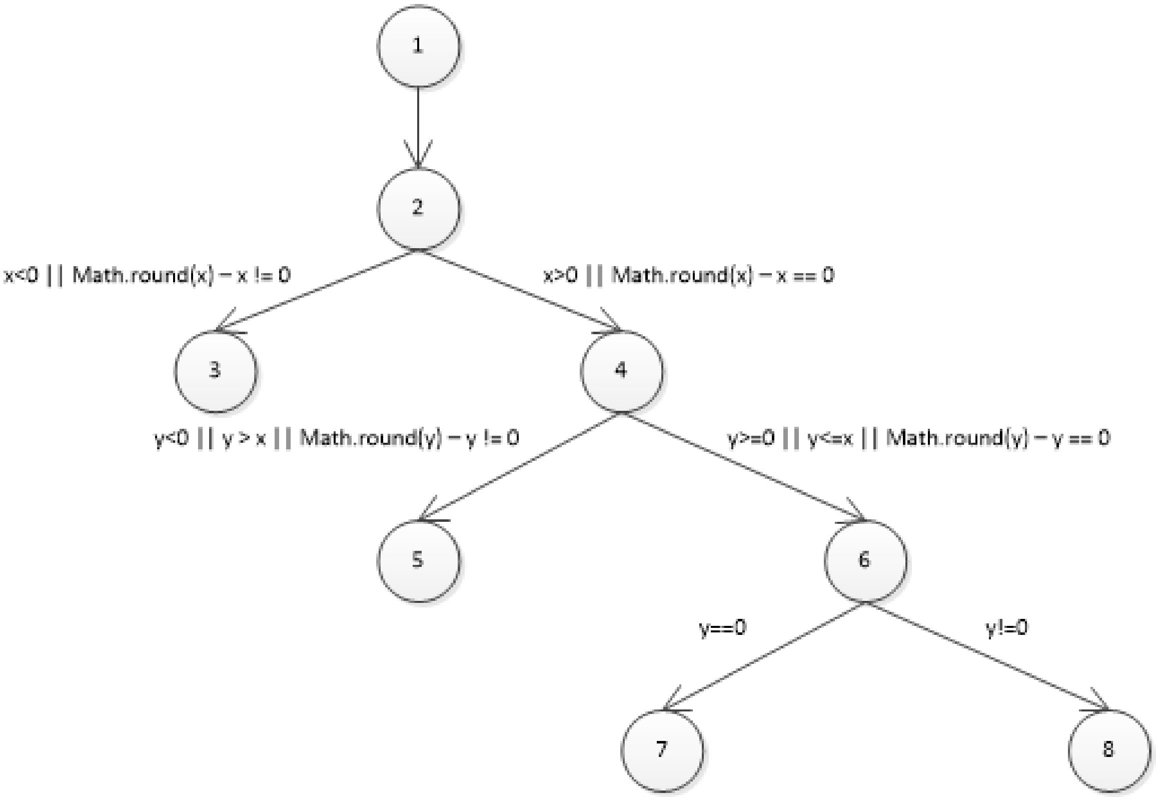

1
2
x<0 || Math.round(x) – x != 0
x>0 || Math.round(x) – x == 0
3
4
y<0 || y > x || Math.round(y) – y != 0
y>=0 || y<=x || Math.round(y) – y == 0
5
6
y==0
y!=0
7
8

Figure 3: Combination Control Flow Graph

**Condition Coverage:**

- x < 0
- x >= 0
- Math.round(x) –x != 0
- Math.round(x) –x == 0
- y < 0
- y >= 0

- y > X
- y < x
- Math.round(y) –y != 0
- Math.round(y) –y == 0
- y == 0
- y != 0

$$Condition\ Coverage = \frac{\#\ of\ condition\ cases\ covered}{(\#\ of\ all\ conditions - \#\ of\ unreachable)}$$

$$= \frac{12}{(12-0)} = 1$$

Therefore we have 100% condition coverage.

**Statement and Branch Coverage:**

- Test 1 traverses nodes 1-2-3
- Test 3 traverses nodes 1-2-4-5
- Test 5 traverses nodes 1-2-4-6-7
- Test 4 traverses nodes 1-2-4-6-8

All nodes and edges are covered; therefore we have 100% statement and branch coverage.

**Path Coverage:**

- Test 1 traverses nodes 1-2-3
- Test 3 traverses nodes 1-2-4-5
- Test 5 traverses nodes 1-2-4-6-7
- Test 4 traverses nodes 1-2-4-6-8

All paths leading from the initial to the final node are covered; therefore we have 100% path coverage.

### 4.2.4 Will’s Manual Results

The following calculations are for the `function(double x, double y)` method in `xor.java`

```
public double function( double x, double y ){
             if( Double.isNaN( x ) || Double.isNaN( y )
                 || Double.isInfinite( x )
                 || Double.isInfinite( y ) )
                 throw new RuntimeException( "Boolean Error" );
             if( Math.abs( y ) > Math.abs( x ) ){
                 double tmp = x;
                 x = y;
                 y = tmp;
             }
             long x_bits = Double.doubleToLongBits( x );
             boolean x_sign = (x_bits >> 63) == 0;
             int x_exponent = (int)((x_bits >> 52) & 0x7FFL);
             long x_significand = x_exponent == 0 ? (x_bits &
0xFFFFFFFFFFFFFL) << 1
                 : (x_bits & 0xFFFFFFFFFFFFFL) | 0x10000000000000L;
             long y_bits = Double.doubleToLongBits( y );
             boolean y_sign = (y_bits >> 63) == 0;
             int y_exponent = (int)((y_bits>>52) & 0x7FFL);
             long y_significand = y_exponent == 0 ? (y_bits &
0xFFFFFFFFFFFFFL) << 1
                 : (y_bits & 0xFFFFFFFFFFFFFL) | 0x10000000000000L;
             y_significand >>= (x_exponent - y_exponent);

             // actually carry out the operation
             x_significand ^= y_significand;

             // now reconstruct result
             if( x_exponent == 0 )
                 x_significand >>= 1;
             else {
                 if( x_significand == 0 ) return 0;
                 while( (x_significand & 0x10000000000000L) == 0 ){
                    x_significand <<= 1;
                    --x_exponent;
                    if( x_exponent == 0 ){
                        x_significand >>= 1;
                        break;
                    }
                 }
                 x_significand &= 0xFFFFFFFFFFFFFL;
             }

             x_bits = ((long)x_exponent) << 52;
             x_bits |= x_significand;

             double result = Double.longBitsToDouble( x_bits );

             // deal with signs
             if( x_sign ^ y_sign )
                 result =- result;
             return result;
}
```

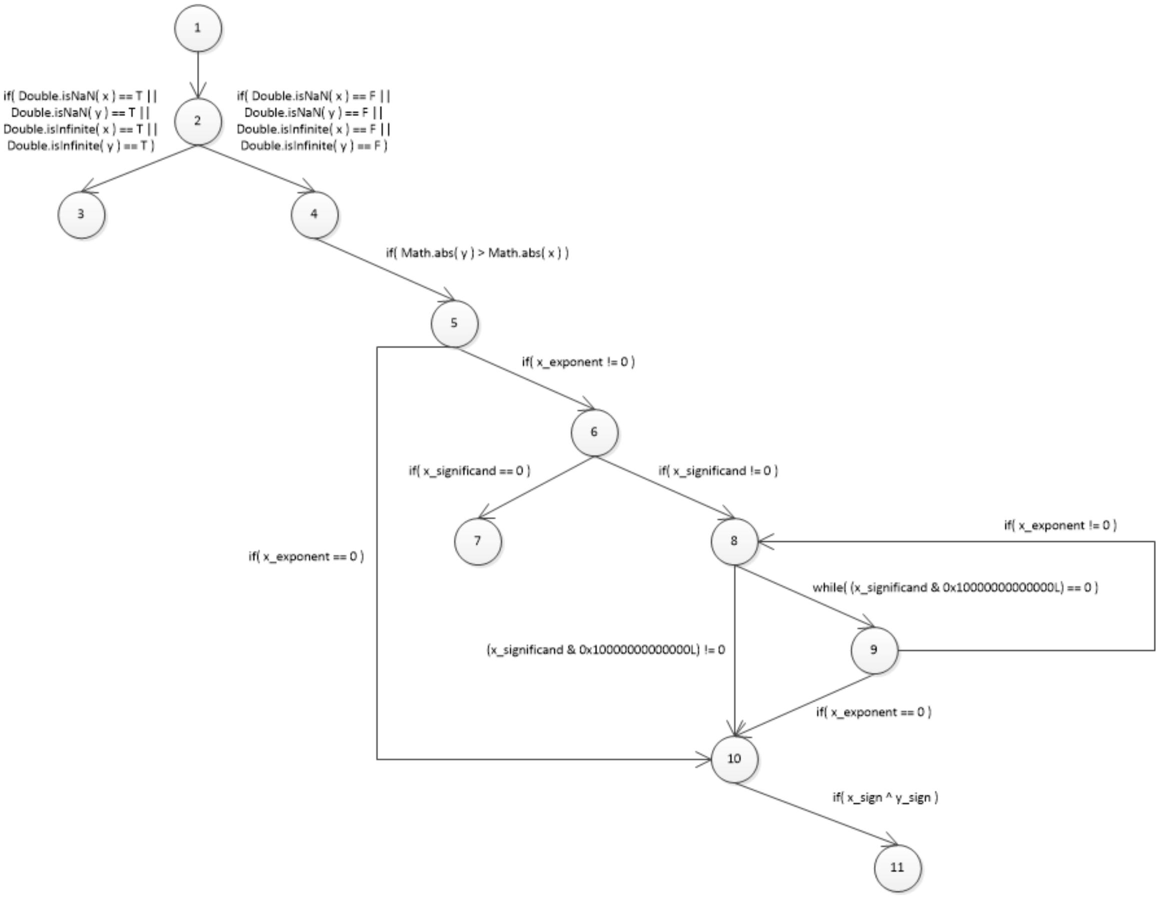

1
2
if( Double.isNaN( x ) == T ||
Double.isNaN( y ) == T ||
Double.isInfinite( x ) == T ||
Double.isInfinite( y ) == T )
if( Double.isNaN( x ) == F ||
Double.isNaN( y ) == F ||
Double.isInfinite( x ) == F ||
Double.isInfinite( y ) == F )
3
4
if( Math.abs( y ) > Math.abs( x ) )
5
if( x_exponent != 0 )
6
if( x_significand == 0 )
if( x_significand != 0 )
7
8
if( x_exponent != 0 )
if( x_exponent == 0 )
while( (x_significand & 0x10000000000000L) == 0 )
(x_significand & 0x10000000000000L) != 0
9
if( x_exponent == 0 )
10
if( x_sign ^ y_sign )
11

Figure 4: XOR Control Flow Graph

## Condition Coverage:

- Double.isNaN( x ) || Double.isNaN( y ) || Double.isInfinite( x ) || Double.isInfinite( y ) == T
- Double.isNaN( x ) || Double.isNaN( y ) || Double.isInfinite( x ) || Double.isInfinite( y ) == F
- Math.abs( y ) > Math.abs( x )
- x_exponent != 0
- x_exponent == 0
- x_significand == 0

- x_significand != 0
- x_significand & 0x10000000000000L) == 0
- x_significand & 0x10000000000000L) != 0
- x_exponent != 0
- x_exponent == 0
- x_sign ^ y_sign

$$Condition\ Coverage = \frac{\#\ of\ condition\ cases\ covered}{(\#\ of\ all\ conditions - \#\ of\ unreachable)}$$

$$= \frac{11}{(12-1)} = 1$$

The second instance of `x_exponent != 0` is infeasible due to it being highly unlikely and unreachable. This second statement (located on the path between node 9 to 8) should be obsolete because of the previous check for `x_exponent != 0` (located in the paths before nodes 7 and 8). Therefore we have 100% condition coverage.

**Statement and Branch Coverage:**

- Test 1 traverses nodes 1-2-4-5-10-11
- Test 3 traverses nodes 1-2-4-5-6-8-10-11
- Test 4 traverses nodes 1-2-4-5-6-7
- Test 10 traverses nodes 1-2-3
- Test 14 traverses nodes 1-2-4-5-6-8-9-10-11 **(This test was added to reach 100%, more about it in the next section)**

All nodes and edges are covered; therefore we have 100% statement and branch coverage.

**Path Coverage:**

Statement and Branch Coverage calculations showed that all except for one path was traverse. As mentioned in Condition Coverage, the path between node 9 and node 8 is infeasible and therefore should be removed. Without the infeasible path, we can conclude that we have 100% coverage.

### 4.3 Improved Code Coverage Attempts

Test member George and Cory did not have to perform any improvements with their test cases. Their results in Section 1 indicated that the selection of inputs reveals all possible errors with respect to the Test Selection Problem.

Code coverage and white-box testing showed Yasir, he had to add some extra test to test for methods inside the class that were previously not tested for. Some methods in the class were not being run and with the help of EclEMMA, these methods were targeted and special tests were created.

**Tests added:**

- public void testNameArray()
- public void testOObject()
- public void testComplexObject()
- public void testErrorComplexObject()
- public void testMain()

These tested the methods in the class that were previously not executed, and completed the 100% code coverage.

Results in Section 1 indicated that Will's set of inputs was incomplete for both ADD and XOR functions. After using Code Coverage tools, he was able to locate where in the code, his tests did not account for. Using this knowledge, he developed 3 new tests and was able to raise his coverage to 96.9% from 90%, in regards to the XOR function. This is presumably the highest coverage achievable due to an infeasible condition/path at line 79.

To improve the ADD function code coverage, the same process was used. Will developed 6 new tests and was about to raise his coverage to 97.1% from 93%. This is presumably the highest coverage achievable due to an infeasible condition/path; the ADD and XOR functions contain very similar code structure and this unit of the code was identical to the infeasible path in XOR.

### 4.3.1 Will's Improved Results

| **Class Name** | **Code Coverage** |
|---|---|
| Natural Logarithm | 100% |
| Inverse Logarithm | 100% |
| AND | 97.1% (+4.1%) |
| XOR | 96.9% (+6.9%) |

## 4.4 Conclusions and Recommendations

It is in the opinion of the test team that testing process is easier if Black-Box testing is done initially, then Code coverage and finally White-Box testing. Black-Box testing should be done first to find all obvious bugs; this will give testers a good initial guess of what the input selection should be with regards to the Test Selection problem. Code coverage should be done after to find any areas that Black-Box testing may have missed. Afterwards White-Box testing should be performed to eliminate those areas missed; the second and third steps should be repeated as much as possible in order to raise code coverage. The entire process should give the team a near complete selection of possible inputs to test with.

The team is also in agreement that EclEMMA is highly recommended in code coverage for Java programming. The eclipse plug-in offers ease of use and makes it easy to locate where testing is absent.

# 5. Mutation Testing

## 5.1 Initial Mutation Testing Results

Our test team consists of four members: George, Yasir, Cory and Will. The mutation-testing tool used was **PITClipse**, a bytecode based plug-in for Java.

The following tables are a summary of the results of automated mutation testing done by each respective member. For each table, there are corresponding collections of screenshots taken of raw data located in Appendix A.

### 5.1.1 George's Initial Results

| **Class Name** | **Mutation Coverage** |
|---|---|
| Cube | 100% |
| Square | 100% |
| Inverse | 100% |
| Factorial | 100% |

### 5.1.2 Yasir's Initial Results

| **Class Name** | **Mutation Coverage** |
|---|---|
| Inverse Cosine | 62% |
| Cosine | 60% |
| Sine | 60% |
| Tangent | 60% |

### 5.1.3 Cory's Initial Results

| **Class Name** | **Mutation Coverage** |
|---|---|
| Combination | 88% |
| Cube Root | 100% |
| Logarithmic (Base 10) | 100% |
| Addition | 100% |

### 5.1.4 Will's Initial Results

| **Class Name** | **Mutation Coverage** |
|---|---|
| Natural Logarithm | 100% |
| Inverse Logarithm | 100% |
| AND | 73% |
| XOR | 88% |

## 5.2 Process of Improving Test Effectiveness

### 5.2.1 Initial Stage

All test suites were prepared before testing; the prerequisites for mutation testing are:

- The maximum code coverage achieved
- All test suites must result in pass[1]

After all preparations are completed, faults (or mutations) will be automatically seeded into the course code, by PIT, and then the test suites will be ran. If the tests result in a **fail** then the mutation are "**killed**". But if the tests result in a **pass** then the mutation **lives.** The percentage of mutations killed are achieve from the following formula:

$$Mutation\ Coverage\colon \frac{\#\ of\ Mutations\ killed\ by\ Tests}{\#\ of\ Mutations\ introduced\ by\ PIT}$$

### 5.2.2 Analysis Stage

PIT will return several noteworthy items in its result report but the most important of these items are:

- Mutations and their respective status
  - Statuses include: Killed, Survive and No Coverage

  This item will indicate specifically which mutations survive; the ones our test suites failed to catch.
- Location of mutations

  This item, along with the previous item, will help the team determine if the mutation is even possible/applicable; if mutation is applicable, changes can be made to test in order to address and kill it.

  Applicability depends on the following requirements:
  - Reach ability – location must be feasible
  - Propagation – the incorrect state, caused by mutant, must propagate to output and must be checked by test suite

### 5.2.3 Improvement Stage

After a mutation is determined to be pertinent, the test can be revised by understanding which mutant operator type was used. Location of mutant can reveal which line infections were introduced; this information is also vital in revising test suites.

This process will be repeated until all mutations have been killed.

[1] Tests that fail cannot find the mutations introduced by PIT

## 5.3 In-Depth Analysis

### 5.3.1 Analysis of George's Results

**George** did not have to change anything with his test cases. The results showed that the test cases covered all mutations performed by Pit with passing results. All code was covered and it handled various forced errors that arisen during the mutation testing.

### 5.3.2 Analysis of Yasir' Results

**Yasir** had to implement a few new tests to cover some previously missed conditional checks. Pitclipse tested for negated conditions, which made some mutations, which survived. Three new tests had to be implemented to get near 100% mutation coverage. Most tests did not have 100% coverage due to GUI mutations surviving. This will be addressed during the next milestone.

The following table demonstrates the improvements made by Yasir:

| **Class Name** | **Mutation Coverage** |
|---|---|
| Inverse Cosine | 92% (+30%) |
| Cosine | 90% (+30%) |
| Sine | 90% (+30%) |
| Tangent | 80% (+20%) |

### 5.3.3 Analysis of Cory's Results

**Cory** did not have to perform any improvements with his test cases. Their results indicated that the selection of test cases reveals all possible errors with respect to the Test Selection Problem and Mutation testing; the only exception being the mutation testing dealing with Combination.

For the Combination mutation test, 17 mutations were introduced to the code and only 15 were killed leaving us with 88% mutation coverage. Two of the tests required for 100% coverage were commented out because they fail in jUnit and as such are unable to be used in mutation testing. This makes it impossible to achieve 100% mutation coverage with the Combination function.

### 5.3.4 Analysis of Will's Results

**Will** did not have to make any improvements on the logarithmic functions tested. Previously done white box testing revealed that those functions are simple functions that resulted in simple testing; this means the input selection were easier to obtain than the XOR and AND functions due to their respective complexity.

To improve the AND mutation results, Will came across the same problem as Cory. Some of the mutation testing required failed test cases in jUnit. To raise the mutation coverage, contradictory actions such as making those failed test suites passable with incorrect expected values. Therefore, in this case, mutation-testing objectives somewhat conflicts with white-box testing objectives.

In both XOR and AND cases, there was an occurrence of the same mutation that was not covered by the test suites. The mutation, on line 80 of both files, was ruled to be irrelevant and not applicable. This is because of the infeasible code that was revealed by the previously done white-box testing and thus, was not tested for.

All other mutations were deemed not applicable with regards to this program. This program must be programmed according to mathematical concepts. The mutations introduced can only occur in real world situations only if the programmer was not competent. Even if the programmer was not competent, faults and incorrectly implemented mathematical concepts should be picked up during white-box testing.

The following table demonstrates the improvements made by Will:

| **Class Name** | **Mutation Coverage** |
|---|---|
| Natural Logarithm | 100% |
| Inverse Logarithm | 100% |
| AND | 80% (+7%) |
| XOR | 83% |

### 5.3.5 Analysis of specific Mutants, or classes of Mutants

#### *5.3.5.1 Conditional Boundary Mutator*

| **Original Conditional** | **Mutated Conditional** |
|---|---|
| < | <= |
| <= | < |
| > | >= |
| >= | > |

The conditional boundary mutator becomes very important when testing test cases that deal with if statements. It would be very easy when creating a test case to use < when infact <= was intended or vice versa. While the majority of test cases would pass and it might appear that the test was working as intended it would fail on select test cases and execute code that was not intended. This would result in any number of unforeseen consequences including giving us false positives or false negatives for the test case.

#### *5.3.5.2 Math Mutator*

| **Original Operation** | **Mutated Operation** |
|---|---|
| x * x * x | x / x * x |
| x * x * x | x * x / x |
| 1 / x | 1 * x |
| x + x | x - x |

| | |
|---|---|
| x - x | x + x |

The math mutator is extremely important when testing to make that the values you receive from your functions have the correct values. It's not always obvious that something is wrong with your code, for example if you only check a value is greater than 0 but instead of saying x-x you are saying x+x, then you'll get weird behaviour in your program which can become a huge headache. Test case might pass with x+x but you require x-x, allowing a mutation to occur you can notice issues in your tests. The review of the mutations allows another form of debugging before finalizing software.

## 5.4 Conclusions and Recommendations

After examining the results of this phase in the testing process, mutation testing is somewhat effective in the testing process. There are many reasons for this such as complexity of functions. For two of our members, their functions were simple enough to achieve 100% coverage.

For Yasir, mutation testing revealed some missed input selections and thus increased the effectiveness of his test suite. On the other hand, mutation testing can be conflict with the objectives of white-box/black-box testing. PIT requires all test cases that fail cannot be used in mutation testing but those tests were used to reveal implementation problems. Those **same** tests may be required to achieve higher mutation coverage and thus, causes a dilemma for the testing team.

Mutation testing can reveal some unique inputs missed but again those flaws introduced by the testing process may not be realistically applicable. Most flaws introduced can only happen if the programmer implemented concepts and algorithms incorrectly; these flaws can be found and rectified during the white-box testing phase.

So in conclusion, only one of our members has increased their effectiveness of their tests significantly. The remainder of our team experienced little to no improvement to effectiveness to their tests. Therefore, mutation testing should only be used if there is time for it in the project development schedule or to increase the confidence in test input selection with regards to the test selection problem.

The team is also in agreement that PIT is highly recommended in code coverage for Java programming. The eclipse plugin Pitclipse offers ease of use and makes it easy to locate where test cases are not being performing as intended. It should be noted that PIT is very much still in development and has a potential of containing bugs and the eclipse plugin Pitclipse is more likely to contain bugs than the standalone PIT.

# 6. GUI and Integration Testing

## 6.1 GUI Testing Approach and Results

### 6.1.1 Rationale

Graphical User Interface (GUI) testing can be used in order to determine if an application has met functional and non-functional requirements outlined in the specifications. GUI testing can also be used as an improvised way to conduct basic integration testing since the GUI makes use of several of components in unison to perform tasks.

The approach in GUI testing for this application can be described as a focused, exploratory method. The team chose to do exploratory testing it can uncover more bugs that normal standard ways of testing may ignore. It also utilizes the team's experience and prior knowledge to adjust to test cases to cover more cases and scenarios; the ability to adjust accordingly during testing is one advantage that exploratory testing has over automated testing.

### 6.1.2 Approach

GUI testing and preparations were primarily based on Session Based Test Management Cycle (SBTM): Bug Classification, Test Objectives, Time Restrictions, Review and Debriefing.

#### *6.1.2.1 Define Bug Classification*

The team categorized bugs in four categories of severity/priority: critical, normal, minor or cosmetic. Critical bugs are ones that compromises and impedes the usability of application. Such types of bugs are: data loss, corruption, application crashes, and inability to save work. All other bugs are classified as major or minor according to tester experience and how severe they impede the user's ability to make use of the application. Cosmetic bugs only affect graphical appearance and do not affect the logic behind the program at all.

#### *6.1.2.2 Test Objectives*

The prime objective is very simple for this type of application: Determine if, under varying real-world environments, the components of the applications are working correctly underneath the GUI.

Test ideas should be the starting point for exploratory testing and should be based of prior black-box testing results.

#### *6.1.2.3 Time Restrictions*

Restrictions on test sessions encourage testers to react on response events from the system and prepare for the correct outcome. Testing was done in 90 minutes sessions with the following restrictions:

- There should be no interruptions in the 90 minutes session
- The sessions can be extended or reduced by 45 minutes

#### *6.1.2.4 Review*

After testing, the team will reconvene to review all outcomes. This review entails that the testing team will evaluate all bugs and learn where they are locate so that they can be fixed. A small analysis of coverage will be concluded as well.

#### *6.1.2.5 Debrief*

Once the review session is over, a compilation of all data obtained will happen. This compilation will be examined to see if all testing objectives are covered. Based off that, additional tests may be created in order to achieve any objectives that were not fulfilled.

#### *6.1.2.6 Other Testing Specifications*

Testing is to be conducted in pairs since one person may miss bugs that occur or paths to certain states that need to be traverse. Notes and test execution logs are required to track progression and program states. An example of this log can be seen in Appendix I.

### 6.1.3 Testing Results

For a more detailed account of results, please refer to Appendix G.

#### *6.1.3.1 Team A Results Summary*

| | |
|---|---|
| **Tests Attempted** | 33 |
| **Tests Passed** | 30 |
| **Bugs Found** | 3 |

#### *6.1.3.2 Team B Results Summary*

| | |
|---|---|
| **Tests Attempted** | 9 |
| **Tests Passed** | 8 |
| **Bugs Found** | 1 |

## 6.2 Integration Testing

### 6.2.1 Rationale

Integration testing is the testing of one or more components within a system. A component consists of more than one unit, which was tested during black- and white-box testing. This type of testing checks to see how units interact with each other in a typical environment and if the interfaces used to govern these units are working properly.

Integration testing was not done by the team due to the structure of the program. All units that were tested during black- and white-box testing are subclasses of abstract classes. These abstract classes cannot be tested due their inability to be instantiated. The class (Parser) that uses those abstract super classes (GObject/OObject) could have been tested but after reviewing the program structure, the team decided to test the perform GUI testing instead. The diagram bellows this structure in detail:

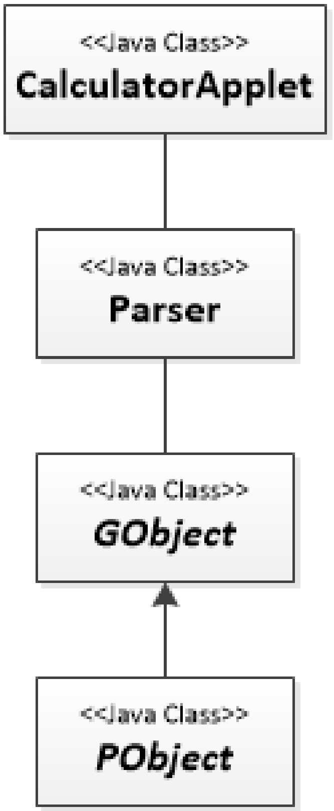


Figure 5: Diagram of Program Structure

By performing GUI testing (on CalculatorApplet), the team, in a way, has executed a form of integration testing. The GUI uses the Parser class, which uses those abstract super classes.  By testing the GUI, the team can see if those units are working together properly.

### 6.2.2 Log-Factorial Integration Testing

For this section we tested the logarithm and factorial calculator functions together. First a value would be put through the factorial function and that value would be directly sent to the logarithm function. Oddly enough the functions behave normally separately but the moment we put them together they gave completely the wrong answers. Wolframalpha was used as a comparison for actual values. Test results are in Appendix H.

### 6.2.3 Trigonometry and Logical Functions Integration Testing

Some integration tests for the other two classes were also written to make sure that they would work when they are put together. All the functionality of these two classes combined preformed correctly. Test results are in Appendix H.

## 6.3 Bug Reporting

### 6.3.1 Formatting

Bug reports will have the following format:

- **Bug Name –** one line summary of bug
- **Bug ID** – identifying number assigned by bug tracking tool
- **Area Path** – Buttons/Commands needed to get to state
- **Build Number**
- **Severity** – critical, major, minor or cosmetic
- **Priority** – critical, major, minor or cosmetic
- **Reported By**
- **Reported On**
- **Reason**
- **Status:** New/Open/Active
- **Environment –** OS Environment
- **Description –** More detailed summary of bug
- **Steps To Reproduce**
- **Expected Result**

### 6.3.2 Bugs Found during GUI Testing

Bugs uncovered during testing were similar to the ones uncovered during black-box testing. These bugs are mostly precision based and were deem minor. They were not deemed cosmetic because it may affect results in sequential calculations. No major or critical bugs were found during the exploratory testing.
The raw results are in Appendix G as well as summaries in Appendix H.

## 6.4 Conclusions and Recommendations

Exploratory GUI Testing as a methodology for find bugs is somewhat effective. It can be effective in finding typical human prone errors since testing environments are similar to real-world applications. It can also be effective, if the testers have a lot of experience with the application's background, since testers can easily see where potential problem spots are.

However, even with the many advantages that exploratory GUI testing does provide, there are many disadvantages as well. Although automation does exclude the ability to adjust tests accordingly to application behavior, it is more effective due to being repeated and faster. Exploratory testing requires a large amount of time in comparison. This is not to say that GUI testing should be fully automated, but it should be somewhere in between.

Although in-depth integration testing was done not performed for this application, the testing team can see the importance to do some integration testing within the testing cycle. Units that work correctly independent of each other **may** be defective when interacting with one another. That being said, the testing focus should primarily focus on black- and white-box testing to find the majority of the more probable bugs that can occur. Integration testing should be given more priority more than mutation testing but should be done only if time permits.

## Appendix A. – Schedule/Task List

| Task | Predecessor Task | Special Skills | Responsibility | Finish Date |
|---|---|---|---|---|
| 1. Test Preparation | None | Interpreting Code Documentation | Program Testing Team/Test Manager | January 30 |
| 2. Black-Box Testing | Task 1 | | Program Testing Team | February 13 |
| 3. White-Box Testing | Task 2 | | Program Testing Team | March 3 |
| 4. Code Coverage | Task 2 | | Program Testing Team | March 3 |
| 5. Mutation Testing | Task 3 | | Program Testing Team | March 20 |
| 6. GUI and Non-Functionality Testing | Task 4 | UI Experience | Program Testing Team | April 3 |
| 7. Presentation to Department Manager | Task 5 | | Program Testing Team/Test Manager | April 14 |

## Appendix B. – Black-box Testing Results

| Tester | Type Of Tests | Package Tested | Class Tested | Test Scenario | Results | Notes |
|---|---|---|---|---|---|---|
| George | Black-Box | jscicalc.pobject | Cube | Large Negative Value | Pass | |
| | | | | Small Negative Value | Pass | |
| | | | | Zero | Pass | |
| | | | | Small Positive Value | Pass | |
| | | | | Large Positive Value | Pass | |
| | | | | Positive Decimal | Pass | |
| | | | | Negative Decimal | Pass | |
| | | | Square | Large Negative Value | Pass | |
| | | | | Small Negative Value | Pass | |
| | | | | Zero | Pass | |
| | | | | Small Positive Value | Pass | |
| | | | | Large Positive Value | Pass | |
| | | | | Positive Decimal | Pass | |
| | | | | Negative Decimal | Pass | |
| | | | Inverse | Large Negative Value | Pass | |
| | | | | Small Negative Value | Pass | |
| | | | | Zero | Pass | |
| | | | | Small Positive Value | Pass | |
| | | | | Large Positive Value | Pass | |
| | | | | Positive Decimal | Pass | |

|  |  |  |  |
|---|---|---|---|
|  | Negative Decimal | Pass |  |
| Factorial | Large Negative Value | N/A |  |
|  | Small Negative Value | Pass | Only negative test done, rest are redundant, pass condition is an exception is thrown. |
|  | Zero | Pass |  |
|  | Small Positive Value | Pass |  |
|  | Large Positive Value | Pass |  |
|  | Positive Decimal | Fail | Using 2.5, Actual Result: Arithmetic Exception, Expected Result: 1.875 or 2 |
|  | Negative Decimal | N/A |  |

| Tester | Type Of Tests | Package Tested | Class Tested | Test Scenario | Results | Notes |
|---|---|---|---|---|---|---|
| Cory | Black-Box | jscicalc.pobject | Combination | x is negative, y is negative | Pass | |
| | | | | x is negative, y is positive | Pass | |
| | | | | x is positive, y negative | Pass | |
| | | | | x is zero, y is positive | Pass | |
| | | | | x is positive, y is zero | Fail | |
| | | | | x is zero, y is zero | Fail | |
| | | | | x is larger positive, y is smaller positive | Pass | |
| | | | | x is smaller positive, y is larger positive | Pass | |
| | | | | x is large positive, y is large positive | Fail | Java throws stack overflow. Manually testing an error message is displayed, the function tested does not handle this. |
| | | | | x is decimal < .5, y is non decimal positive | Pass | |
| | | | | x is decimal >= .5, y is non decimal | Pass | |

|  |  |  |  |
|---|---|---|---|
|  | positive |  |  |
|  | x is decimal < .5, y is decimal < .5 | Pass |  |
|  | x is decimal >= .5, y is decimal >= .5 | Pass |  |
|  | x is non decimal positive, y is decimal < .5 | Pass |  |
|  | x is non decimal, y is decimal >= .5 | Pass |  |
| Cube Root | Large Negative Value: -8000000 | Fail, uses complex numbers as answers. Says it uses real numbers | Expected: -200, Actual: NaN |
|  | Small Negative Value: -2 | Fail, uses complex numbers as answers. Says it uses real numbers | Expected: -1.259921, Actual: NaN |
|  | Zero | Pass |  |
|  | Small Positive Value | Pass |  |
|  | Large Positive Value | Pass |  |
|  | Positive Decimal | Pass |  |
|  | Negative Decimal: -576.78654 | Fail, uses complex numbers as answers. Says it uses real numbers | Expected: -8.324121, Actual: NaN |
| Logarithm (Base 10) | Large Negative Value: -8000000 | Fail, uses complex numbers as answers. Says it | Expected: Error Message, Actual: NaN |

| | | | |
|---|---|---|---|
| | | uses real numbers | |
| | Small Negative Value: -2 | Fail, uses complex numbers as answers. Says it uses real numbers | Expected: Error Message, Actual: NaN |
| | Zero | Pass | |
| | Small Positive Value | Pass | |
| | Large Positive Value | Pass | |
| | Positive Decimal | Pass | |
| | Negative Decimal: -576.78654 | Fail, uses complex numbers as answers. Says it uses real numbers | Expected: Error Message, Actual: NaN |
| Addition | Large Negative x: -8000000, Large Negative y: -8000000 | Pass | |
| | Small Negative x: -2, Large Negative y: -8000000 | Pass | |
| | Small Negative x: -2, Small Negative y: -2 | Pass | |
| | Large Negative x: -8000000, Small Negative y: -2 | Pass | |
| | x = Zero, non zero y = 5 | Pass | |
| | x = Zero, y = zero | Pass | |

| | |
|---|---|
| Non-zero x = 5, y = zero | Pass |
| Low Positive x: 50, Low Positive y: 60 | Pass |
| Large Positive x: 479001600, Small Positive y = 44 | Pass |
| Large Positive x: 479001600, Large Positive y = 479001600 | Pass |
| Small Positive x: 47, Large Positive y = 479001600 | Pass |
| Positive Decimal x: 56.29, Positive Decimal y: 56.29 | Pass |
| Positive Decimal x: 56.29, Negative Decimal y: -56.29 | Pass |
| Negative Decimal x: -56.29, Positive Decimal y: 56.29 | Pass |
| Negative Decimal x: -56.29, Negative Decimal y: -56.29 | Pass |

| Tester | Type Of Tests | Package Tested | Class Tested | Test Scenario | Results | Notes |
|---|---|---|---|---|---|---|
| Will | Black-Box | jscicalc.pobject | Natural Logarithm | Property: ln(1) = 0 | Pass | |
| | | | | Property: ln(x^y) = y*ln(x) | Pass | |
| | | | | Property: ln(e^y) = y | Pass | |
| | | | | Property: ln(x)+ln(y) = ln(x * y) | Pass | |
| | | | | Zero | Pass | |
| | | | | Negative Non-Zero: -1 | Pass | |
| | | | | Infinity | Pass | |
| | | | | Negative Zero | Pass | |
| | | | | Large Positive Value: 1000 | Pass | |
| | | | | Small Positive Value: 0.0001 | Pass | |
| | | | Inverse Logarithm | Zero | Pass | |
| | | | | Property: 10^(x+1)/10 = 10^x | Pass | |
| | | | | Property: 10^1 = 10 | Fail | Small Precision Difference |
| | | | | Negative Value: -1 | Fail | Small Precision Difference |
| | | | | Large Negative Value: -1000 | Fail | Small Precision Difference |
| | | | | Testing Negative Property Equality: [10^(2+1)]/10 = 10^2 | Fail | Small Precision Difference |
| | | | | Large Positive Value: 1000 | Pass | |
| | | | | Small Positive Value: 0.0001 | Pass | |

| | | | |
|---|---|---|---|
| AND | Property: 0 AND 0 | Pass | |
| | Property: 0 AND 1 | Pass | |
| | Property: 1 AND 0 | Pass | |
| | Property: 1 AND 1 | Pass | |
| | Large Positive Values: 1110 AND 1001 | Pass | |
| | Infinity AND Infinity | Pass | Pass Condition: Exception was Thrown |
| XOR | Property: 0 AND 0 | Pass | |
| | Property: 0 AND 1 | Pass | |
| | Property: 1 AND 0 | Pass | |
| | Property: 1 AND 1 | Pass | |
| | Large Positive Values: 1110 AND 1001 | Pass | |
| | Infinity AND Infinity | Pass | Pass Condition: Exception was Thrown |

| Tester | Type Of Tests | Package Tested | Class Tested | Test Scenario | Results | Notes |
|---|---|---|---|---|---|---|
| Yasir | Black-Box | jscicalc.pobject | Sine | DegreesUpperBound | Pass | |
| | | | | DegreesLowerBound | Pass | |
| | | | | DegreesNegativeUpperBound | Pass | |
| | | | | DegreesNegativeLowerBound | Pass | |
| | | | | DegreesZero | Pass | |
| | | | | DegreesFifty | Pass | |
| | | | | RadiansUpperBound | Fail | |
| | | | | RadiansLowerBound | Pass | |
| | | | | RadiansNegativeUpperBound | Fail | |
| | | | | RadiansNegativeLowerBound | Pass | |
| | | | | RadiansZero | Pass | |
| | | | | RadiansFifty | Pass | |
| | | | Cosine | DegreesUpperBound | Fail | |
| | | | | DegreesLowerBound | Pass | |
| | | | | DegreesNegativeUpperBound | Fail | |
| | | | | DegreesNegativeLowerBound | Pass | |
| | | | | DegreesZero | Pass | |
| | | | | DegreesFifty | Pass | |
| | | | | RadiansUpperBoun | Pass | |

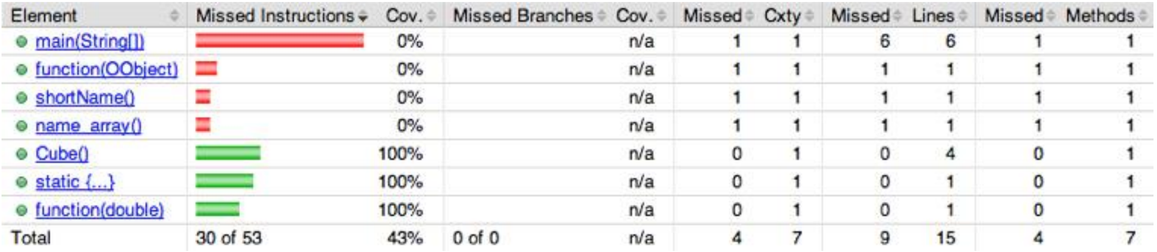
| | | |
|---|---|---|
| | d | |
| | RadiansLowerBound | Pass |
| | RadiansNegativeUpperBound | Pass |
| | RadiansNegativeLowerBound | Pass |
| | RadiansZero | Pass |
| | RadiansFifty | Pass |
| | DegreesUpperBound | Fail |
| | DegreesLowerBound | Pass |
| | DegreesNegativeUpperBound | Fail |
| | DegreesNegativeLowerBound | Pass |
| | DegreesZero | Pass |
| Tangent | DegreesFifty | Pass |
| | RadiansUpperBound | Pass |
| | RadiansLowerBound | Pass |
| | RadiansNegativeUpperBound | Pass |
| | RadiansNegativeLowerBound | Pass |
| | RadiansZero | Pass |
| | RadiansFifty | Pass |
| | DegreesUpperBound | Pass |
| Inverse Cosine | DegreesLowerBound | Pass |
| | DegreesNegativeUp | Pass |

| | |
|---|---|
| perBound | |
| DegreesNegativeLowerBound | Pass |
| DegreesZero | Pass |
| DegreesFifty | Pass |
| RadiansUpperBound | Pass |
| RadiansLowerBound | Pass |
| RadiansNegativeUpperBound | Pass |
| RadiansNegativeLowerBound | Pass |
| RadiansZero | Pass |
| RadiansFifty | Pass |

# Appendix C. – Initial Code Coverage Results

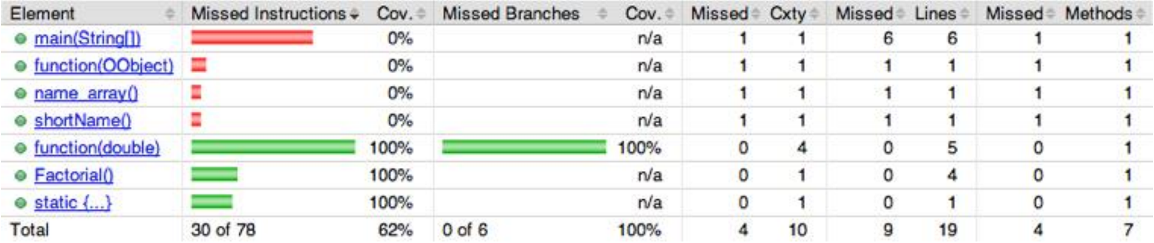
## C.1 George's Initial Code Coverage Results

### C.1.1 Cube



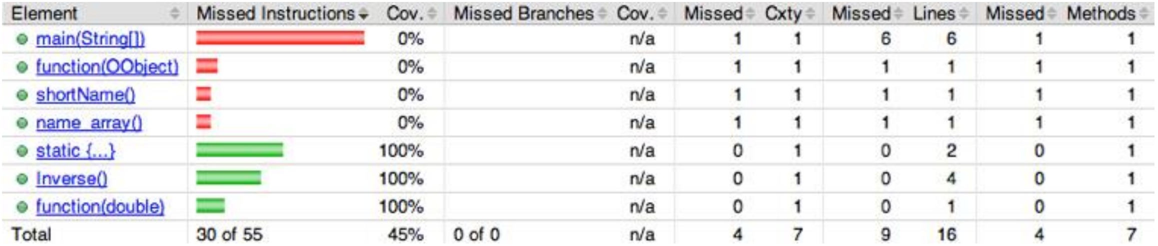
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| shortName() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| Cube() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| function(double) | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 30 of 53 | 43% | 0 of 0 | n/a | 4 | 7 | 9 | 15 | 4 | 7 |

### C.1.2 Factorial



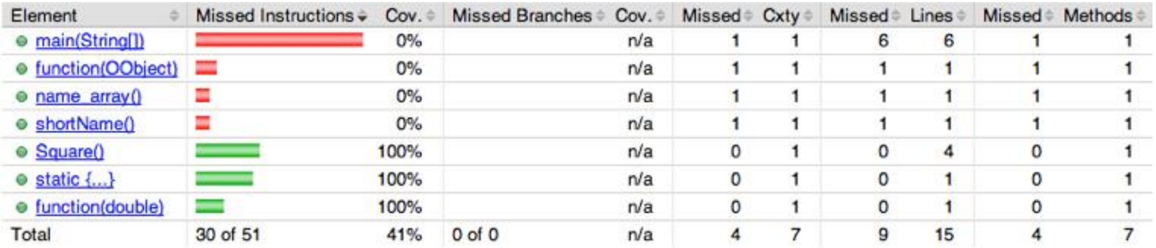
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| shortName() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| function(double) | 🟩 | 100% | 🟩 | 100% | 0 | 4 | 0 | 5 | 0 | 1 |
| Factorial() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 30 of 78 | 62% | 0 of 6 | 100% | 4 | 10 | 9 | 19 | 4 | 7 |

### C.1.3 Inverse



| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| shortName() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| Inverse() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| function(double) | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 30 of 55 | 45% | 0 of 0 | n/a | 4 | 7 | 9 | 16 | 4 | 7 |

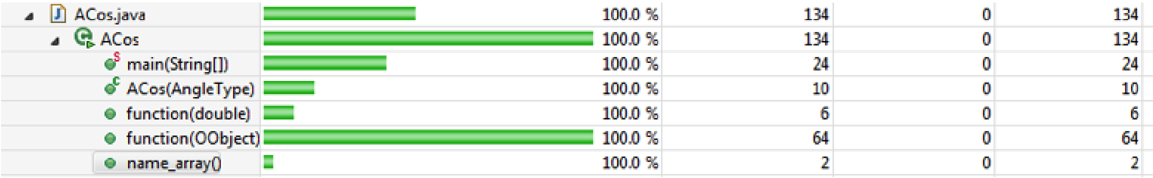
### C.1.4 Square



| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| shortName() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| Square() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| function(double) | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 30 of 51 | 41% | 0 of 0 | n/a | 4 | 7 | 9 | 15 | 4 | 7 |

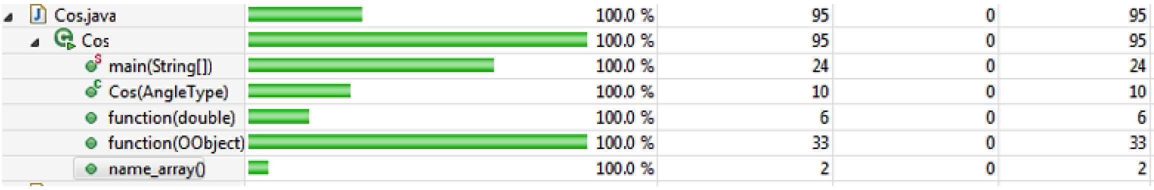
## C.2 Yasir’s Initial Code Coverage Results

### C.2.1 Inverse Cosine



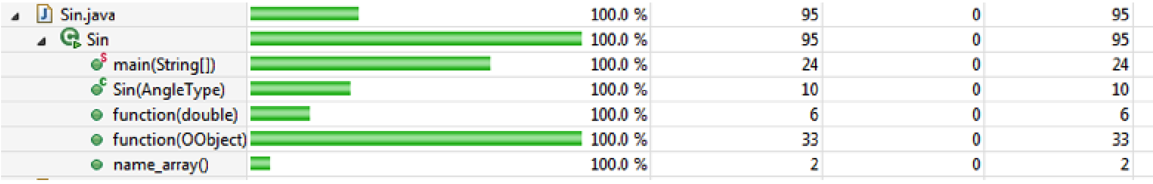
| | | | | |
|---|---|---|---|---|
| ACos.java | 100.0 % | 134 | 0 | 134 |
| ACos | 100.0 % | 134 | 0 | 134 |
| main(String[]) | 100.0 % | 24 | 0 | 24 |
| ACos(AngleType) | 100.0 % | 10 | 0 | 10 |
| function(double) | 100.0 % | 6 | 0 | 6 |
| function(OObject) | 100.0 % | 64 | 0 | 64 |
| name_array() | 100.0 % | 2 | 0 | 2 |

### C.2.2 Cosine



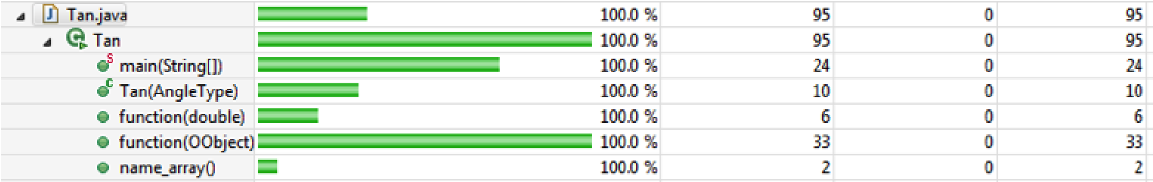
| | | | | |
|---|---|---|---|---|
| Cos.java | 100.0 % | 95 | 0 | 95 |
| Cos | 100.0 % | 95 | 0 | 95 |
| main(String[]) | 100.0 % | 24 | 0 | 24 |
| Cos(AngleType) | 100.0 % | 10 | 0 | 10 |
| function(double) | 100.0 % | 6 | 0 | 6 |
| function(OObject) | 100.0 % | 33 | 0 | 33 |
| name_array() | 100.0 % | 2 | 0 | 2 |

### C.2.3 Sine



| | | | | |
|---|---|---|---|---|
| Sin.java | 100.0 % | 95 | 0 | 95 |
| Sin | 100.0 % | 95 | 0 | 95 |
| main(String[]) | 100.0 % | 24 | 0 | 24 |
| Sin(AngleType) | 100.0 % | 10 | 0 | 10 |
| function(double) | 100.0 % | 6 | 0 | 6 |
| function(OObject) | 100.0 % | 33 | 0 | 33 |
| name_array() | 100.0 % | 2 | 0 | 2 |

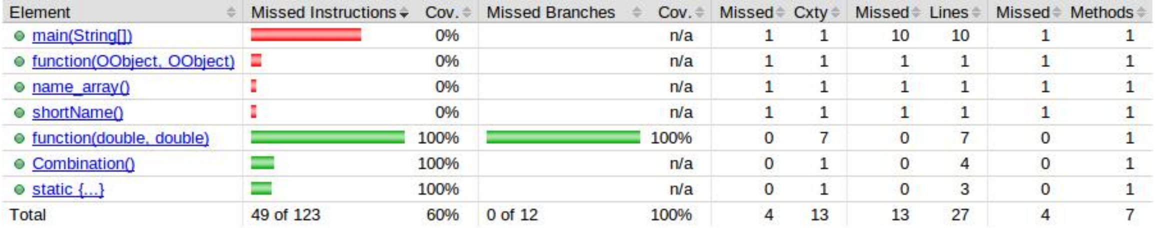
### C.2.4 Tangent



| | | | | |
|---|---|---|---|---|
| Tan.java | 100.0 % | 95 | 0 | 95 |
| Tan | 100.0 % | 95 | 0 | 95 |
| main(String[]) | 100.0 % | 24 | 0 | 24 |
| Tan(AngleType) | 100.0 % | 10 | 0 | 10 |
| function(double) | 100.0 % | 6 | 0 | 6 |
| function(OObject) | 100.0 % | 33 | 0 | 33 |
| name_array() | 100.0 % | 2 | 0 | 2 |

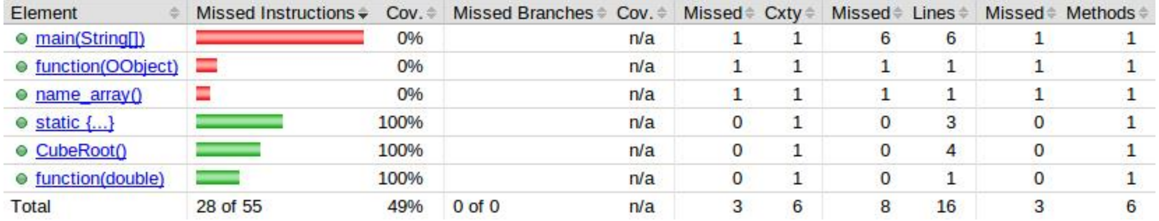
## C.3 Cory’s Initial Code Coverage Results

### C.3.1 Combination



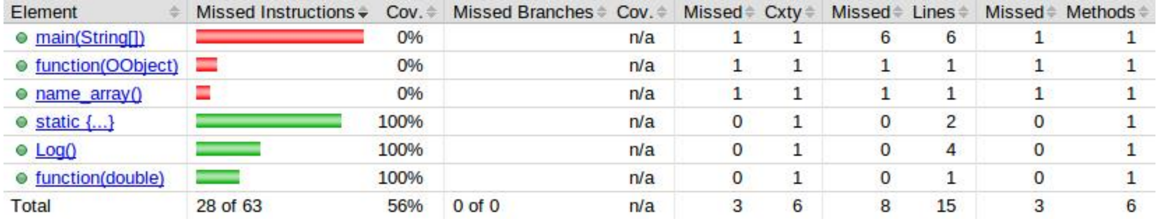
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 10 | 10 | 1 | 1 |
| function(OObject, OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| shortName() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| function(double, double) | 🟩 | 100% | 🟩 | 100% | 0 | 7 | 0 | 7 | 0 | 1 |
| Combination() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 3 | 0 | 1 |
| Total | 49 of 123 | 60% | 0 of 12 | 100% | 4 | 13 | 13 | 27 | 4 | 7 |

### C.3.2 Cube Root



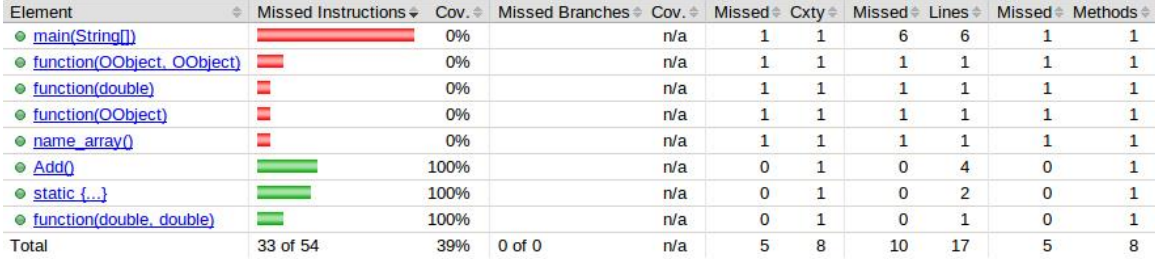
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 3 | 0 | 1 |
| CubeRoot() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| function(double) | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 28 of 55 | 49% | 0 of 0 | n/a | 3 | 6 | 8 | 16 | 3 | 6 |

### C.3.3 Logarithmic (Base 10)



| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| Log() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| function(double) | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 28 of 63 | 56% | 0 of 0 | n/a | 3 | 6 | 8 | 15 | 3 | 6 |

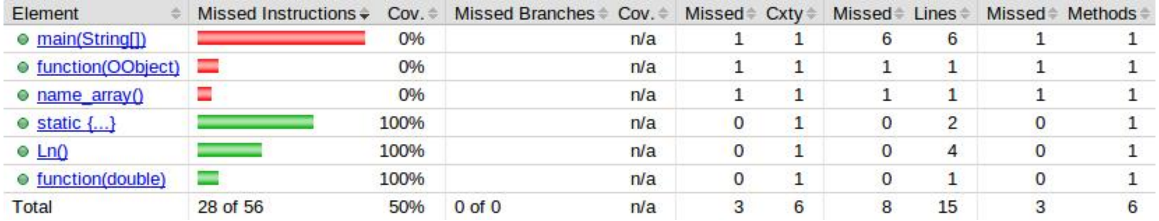
### C.3.4 Addition



| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject, OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| function(double) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| function(OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| Add() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| function(double, double) | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 33 of 54 | 39% | 0 of 0 | n/a | 5 | 8 | 10 | 17 | 5 | 8 |

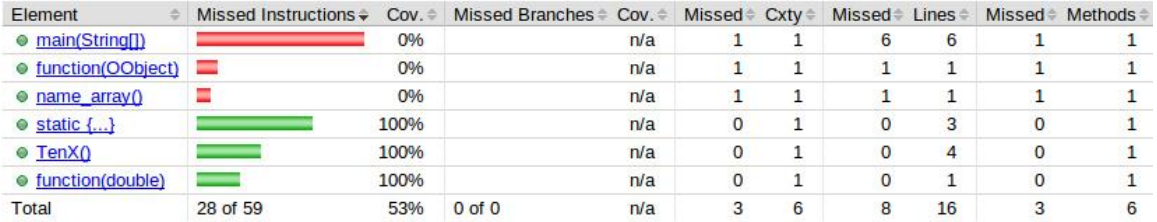
## C.4 Will’s Initial Code Coverage Results

### C.4.1 Natural Logarithmic



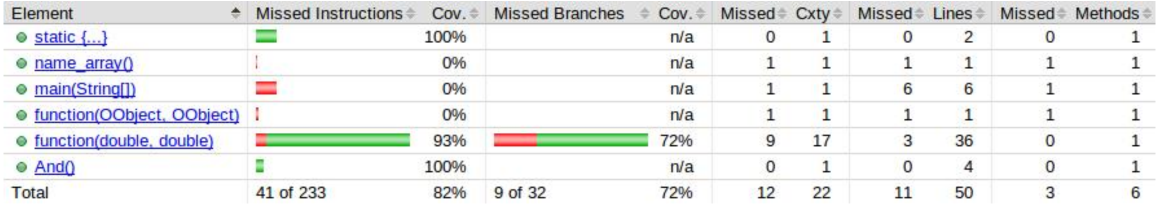
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| Ln() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| function(double) | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 28 of 56 | 50% | 0 of 0 | n/a | 3 | 6 | 8 | 15 | 3 | 6 |

### C.4.2 Inverse Logarithmic



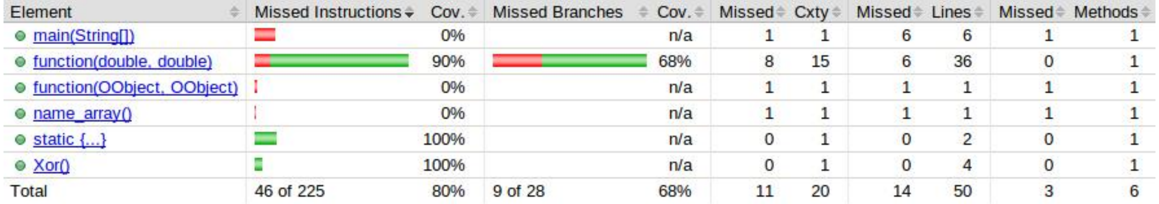
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 3 | 0 | 1 |
| TenX() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| function(double) | 🟩 | 100% | | n/a | 0 | 1 | 0 | 1 | 0 | 1 |
| Total | 28 of 59 | 53% | 0 of 0 | n/a | 3 | 6 | 8 | 16 | 3 | 6 |

### C.4.3 AND



| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(OObject, OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| function(double, double) | 🟥🟩 | 93% | 🟥🟩 | 72% | 9 | 17 | 3 | 36 | 0 | 1 |
| And() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| Total | 41 of 233 | 82% | 9 of 32 | 72% | 12 | 22 | 11 | 50 | 3 | 6 |

### C.4.4 XOR

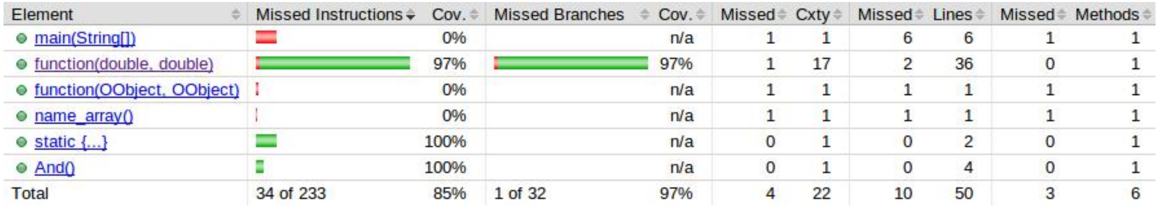


| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(double, double) | 🟥🟩 | 90% | 🟥🟩 | 68% | 8 | 15 | 6 | 36 | 0 | 1 |
| function(OObject, OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| Xor() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| Total | 46 of 225 | 80% | 9 of 28 | 68% | 11 | 20 | 14 | 50 | 3 | 6 |

# Appendix D. – Post-Improved Code Coverage Results

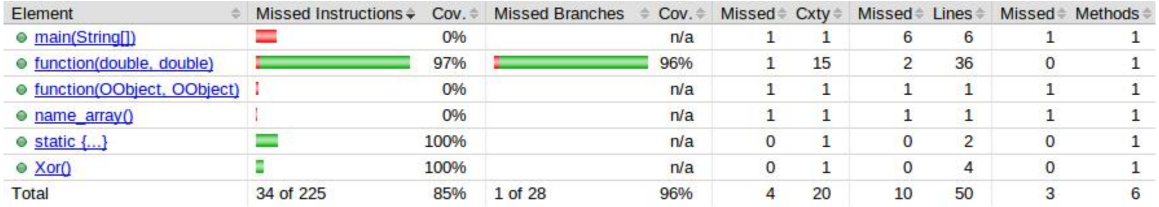
## D.1 Will’s Improved Code Coverage Results

### D.1.1 AND



| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(double, double) | 🟥🟩 | 97% | 🟥🟩 | 97% | 1 | 17 | 2 | 36 | 0 | 1 |
| function(OObject, OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| And() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| Total | 34 of 233 | 85% | 1 of 32 | 97% | 4 | 22 | 10 | 50 | 3 | 6 |

### D.1.2 XOR

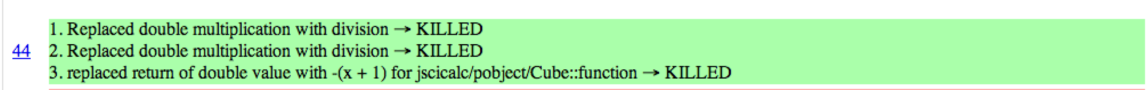


| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---|---|---|---|---|---|---|---|---|---|
| main(String[]) | 🟥 | 0% | | n/a | 1 | 1 | 6 | 6 | 1 | 1 |
| function(double, double) | 🟥🟩 | 97% | 🟥🟩 | 96% | 1 | 15 | 2 | 36 | 0 | 1 |
| function(OObject, OObject) | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| name_array() | 🟥 | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 |
| static {...} | 🟩 | 100% | | n/a | 0 | 1 | 0 | 2 | 0 | 1 |
| Xor() | 🟩 | 100% | | n/a | 0 | 1 | 0 | 4 | 0 | 1 |
| Total | 34 of 225 | 85% | 1 of 28 | 96% | 4 | 20 | 10 | 50 | 3 | 6 |

# Appendix E. – Initial Code Coverage Results

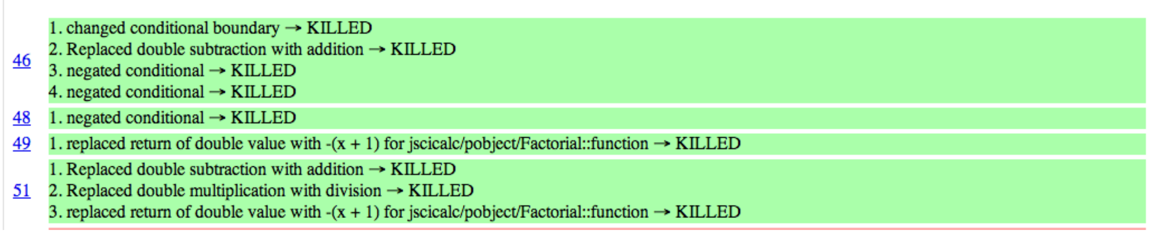
## E.1 George's Initial Mutation Results

### E.1.1 Cube



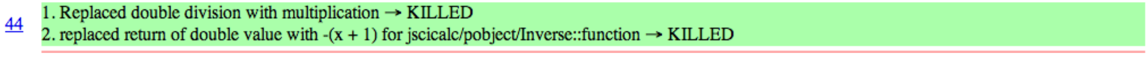
```
    1. Replaced double multiplication with division → KILLED
44  2. Replaced double multiplication with division → KILLED
    3. replaced return of double value with -(x + 1) for jscicalc/pobject/Cube::function → KILLED
```

### E.1.2 Factorial



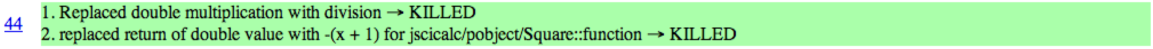
```
    1. changed conditional boundary → KILLED
46  2. Replaced double subtraction with addition → KILLED
    3. negated conditional → KILLED
    4. negated conditional → KILLED
48  1. negated conditional → KILLED
49  1. replaced return of double value with -(x + 1) for jscicalc/pobject/Factorial::function → KILLED
    1. Replaced double subtraction with addition → KILLED
51  2. Replaced double multiplication with division → KILLED
    3. replaced return of double value with -(x + 1) for jscicalc/pobject/Factorial::function → KILLED
```

### E.1.3 Inverse



```
44  1. Replaced double division with multiplication → KILLED
    2. replaced return of double value with -(x + 1) for jscicalc/pobject/Inverse::function → KILLED
```

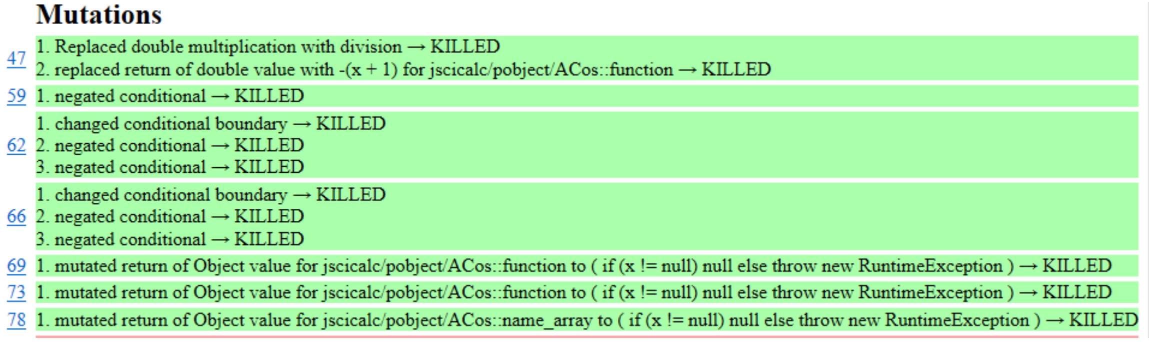
### E.1.4 Square



```
44  1. Replaced double multiplication with division → KILLED
    2. replaced return of double value with -(x + 1) for jscicalc/pobject/Square::function → KILLED
```

## E.2 Yasir’s Initial Mutation Results

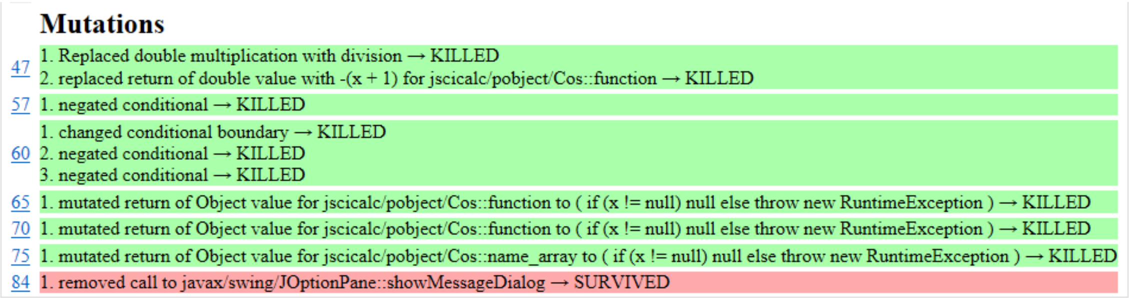
### E.2.1 Inverse Cosine



Mutations

| | |
|---|---|
| 47 | 1. Replaced double multiplication with division → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/ACos::function → KILLED |
| 59 | 1. negated conditional → KILLED |
| 62 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED<br>3. negated conditional → KILLED |
| 66 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED<br>3. negated conditional → KILLED |
| 69 | 1. mutated return of Object value for jscicalc/pobject/ACos::function to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 73 | 1. mutated return of Object value for jscicalc/pobject/ACos::function to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 78 | 1. mutated return of Object value for jscicalc/pobject/ACos::name_array to ( if (x != null) null else throw new RuntimeException ) → KILLED |

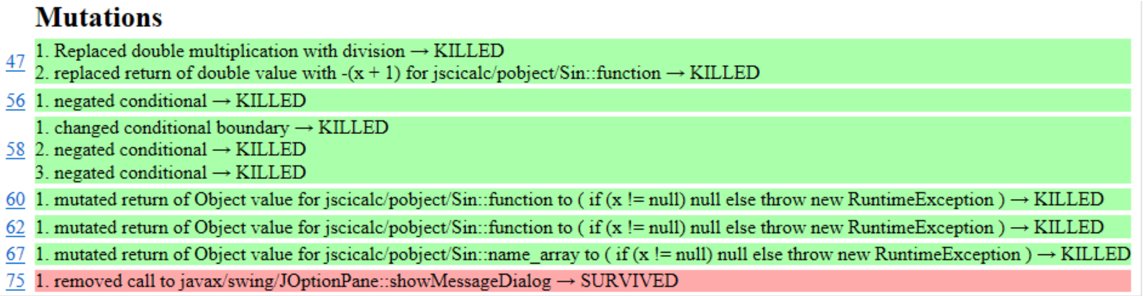
### E.2.2 Cosine



Mutations

| | |
|---|---|
| 47 | 1. Replaced double multiplication with division → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/Cos::function → KILLED |
| 57 | 1. negated conditional → KILLED |
| 60 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED<br>3. negated conditional → KILLED |
| 65 | 1. mutated return of Object value for jscicalc/pobject/Cos::function to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 70 | 1. mutated return of Object value for jscicalc/pobject/Cos::function to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 75 | 1. mutated return of Object value for jscicalc/pobject/Cos::name_array to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 84 | 1. removed call to javax/swing/JOptionPane::showMessageDialog → SURVIVED |

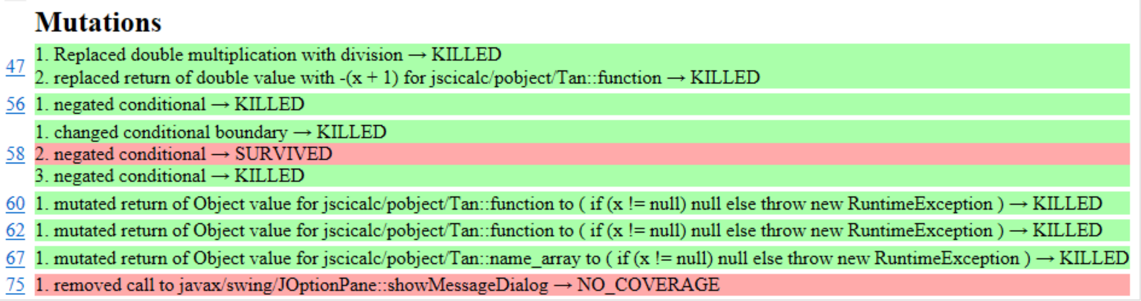
### E.2.3 Sine



Mutations

| | |
|---|---|
| 47 | 1. Replaced double multiplication with division → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/Sin::function → KILLED |
| 56 | 1. negated conditional → KILLED |
| 58 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED<br>3. negated conditional → KILLED |
| 60 | 1. mutated return of Object value for jscicalc/pobject/Sin::function to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 62 | 1. mutated return of Object value for jscicalc/pobject/Sin::function to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 67 | 1. mutated return of Object value for jscicalc/pobject/Sin::name_array to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 75 | 1. removed call to javax/swing/JOptionPane::showMessageDialog → SURVIVED |

### E.2.4 Tangent

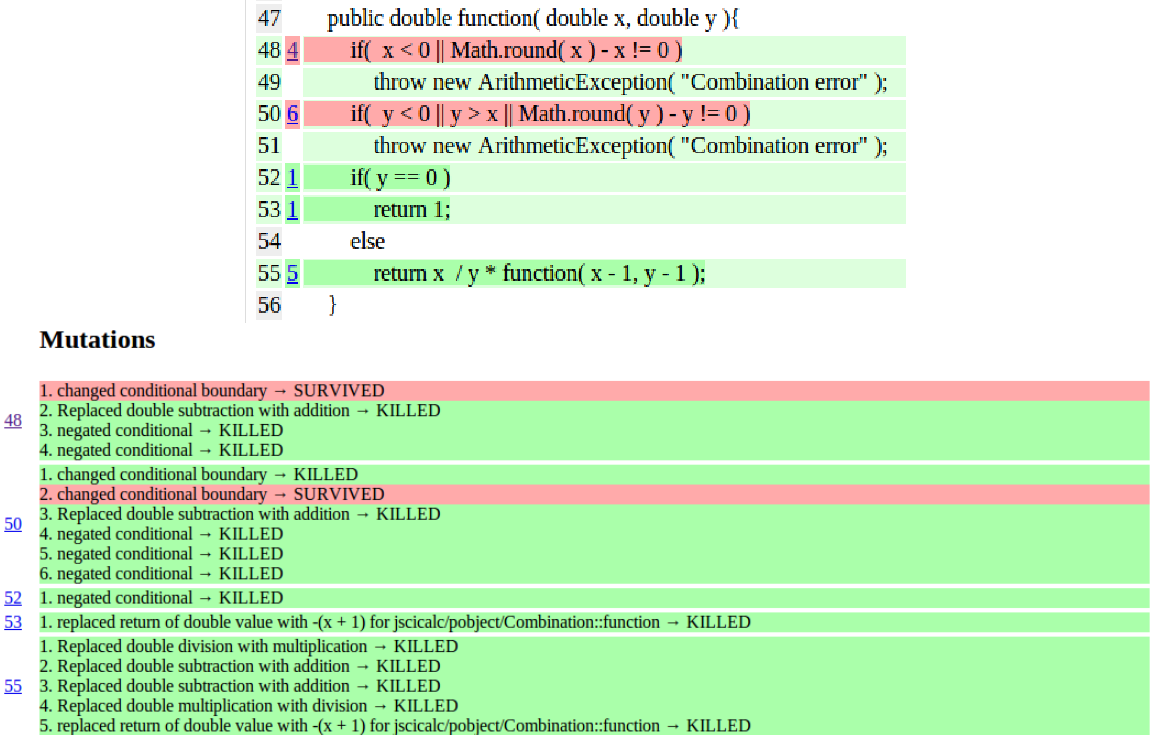


Mutations

| | |
|---|---|
| 47 | 1. Replaced double multiplication with division → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/Tan::function → KILLED |
| 56 | 1. negated conditional → KILLED |
| 58 | 1. changed conditional boundary → KILLED<br>2. negated conditional → SURVIVED<br>3. negated conditional → KILLED |
| 60 | 1. mutated return of Object value for jscicalc/pobject/Tan::function to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 62 | 1. mutated return of Object value for jscicalc/pobject/Tan::function to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 67 | 1. mutated return of Object value for jscicalc/pobject/Tan::name_array to ( if (x != null) null else throw new RuntimeException ) → KILLED |
| 75 | 1. removed call to javax/swing/JOptionPane::showMessageDialog → NO_COVERAGE |

## E.3 Cory’s Initial Mutation Results

### E.3.1 Combination

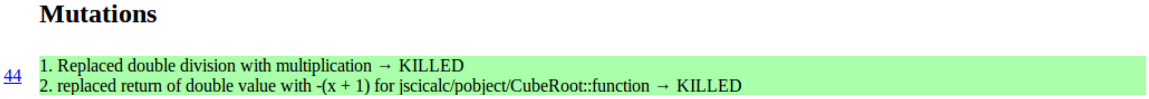


```
47      public double function( double x, double y ){
48 4      if(  x < 0 || Math.round( x ) - x != 0 )
49          throw new ArithmeticException( "Combination error" );
50 6      if(  y < 0 || y > x || Math.round( y ) - y != 0 )
51          throw new ArithmeticException( "Combination error" );
52 1      if( y == 0 )
53 1          return 1;
54        else
55 5          return x  / y * function( x - 1, y - 1 );
56      }
```

**Mutations**

| | |
|---|---|
| 48 | 1. changed conditional boundary → SURVIVED<br>2. Replaced double subtraction with addition → KILLED<br>3. negated conditional → KILLED<br>4. negated conditional → KILLED |
| 50 | 1. changed conditional boundary → KILLED<br>2. changed conditional boundary → SURVIVED<br>3. Replaced double subtraction with addition → KILLED<br>4. negated conditional → KILLED<br>5. negated conditional → KILLED<br>6. negated conditional → KILLED |
| 52 | 1. negated conditional → KILLED |
| 53 | 1. replaced return of double value with -(x + 1) for jscicalc/pobject/Combination::function → KILLED |
| 55 | 1. Replaced double division with multiplication → KILLED<br>2. Replaced double subtraction with addition → KILLED<br>3. Replaced double subtraction with addition → KILLED<br>4. Replaced double multiplication with division → KILLED<br>5. replaced return of double value with -(x + 1) for jscicalc/pobject/Combination::function → KILLED |

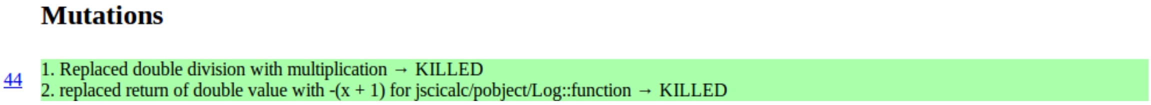
### E.3.2 Cube Root



**Mutations**

| | |
|---|---|
| 44 | 1. Replaced double division with multiplication → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/CubeRoot::function → KILLED |

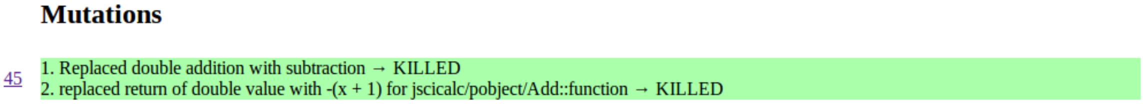
### E.3.3 Logarithmic (Base 10)



**Mutations**

| | |
|---|---|
| 44 | 1. Replaced double division with multiplication → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/Log::function → KILLED |

### E.3.4 Addition

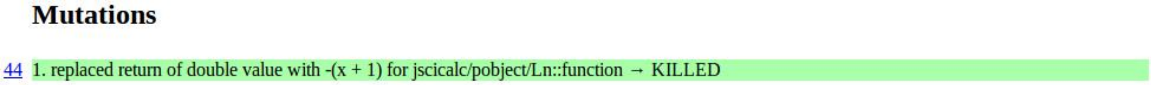


**Mutations**

| | |
|---|---|
| 45 | 1. Replaced double addition with subtraction → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/Add::function → KILLED |

## E.4 Will's Initial Mutation Results

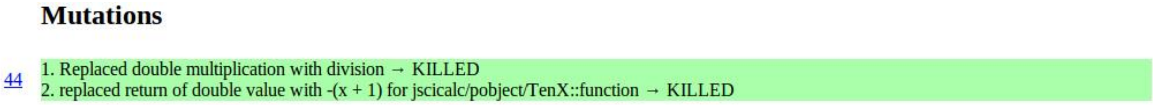
### E.4.1 Natural Logarithmic



**Mutations**

| | |
|---|---|
| 44 | 1. replaced return of double value with -(x + 1) for jscicalc/pobject/Ln::function → KILLED |

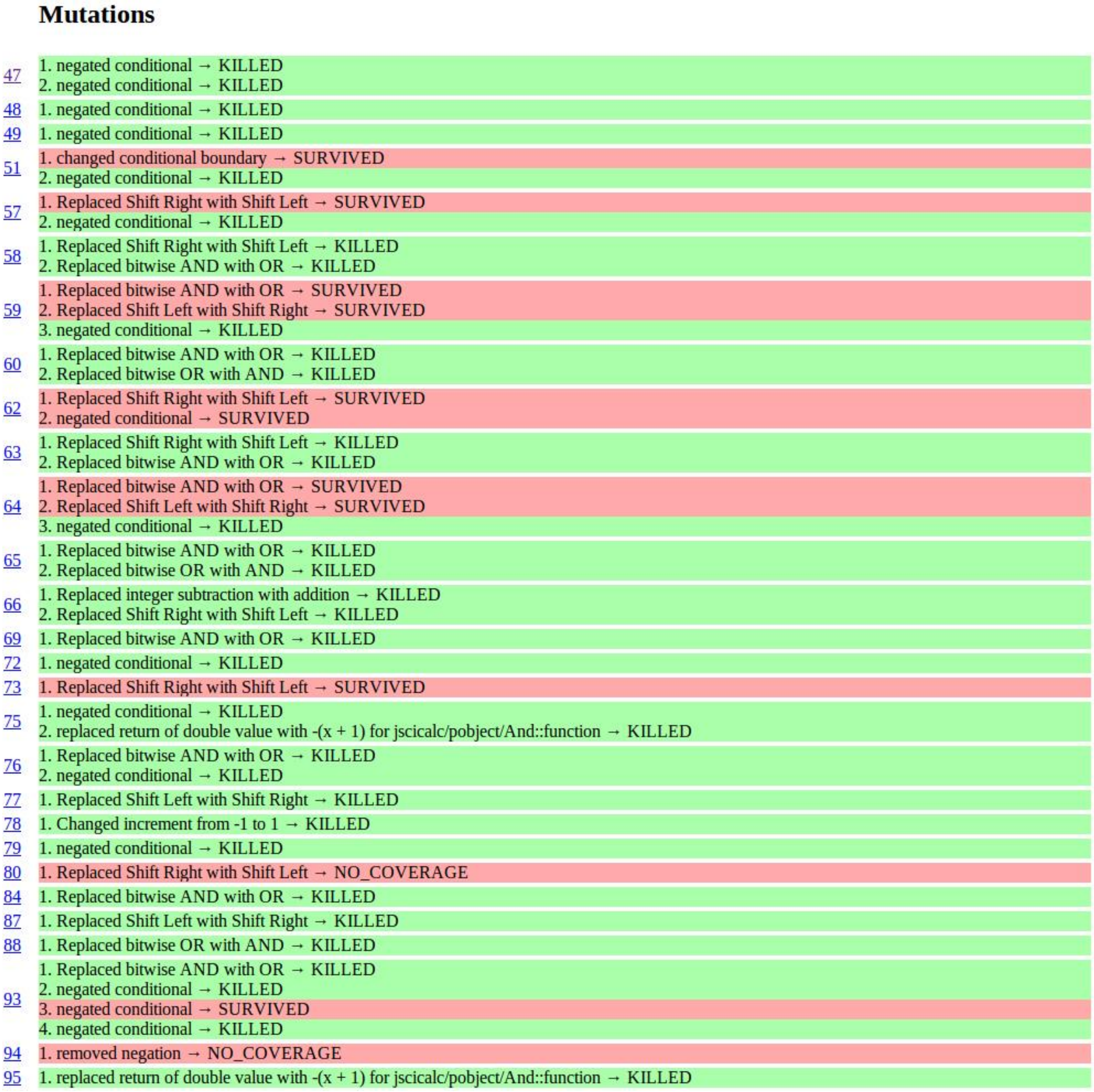
### E.4.2 Inverse Logarithmic



**Mutations**

| | |
|---|---|
| 44 | 1. Replaced double multiplication with division → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/TenX::function → KILLED |

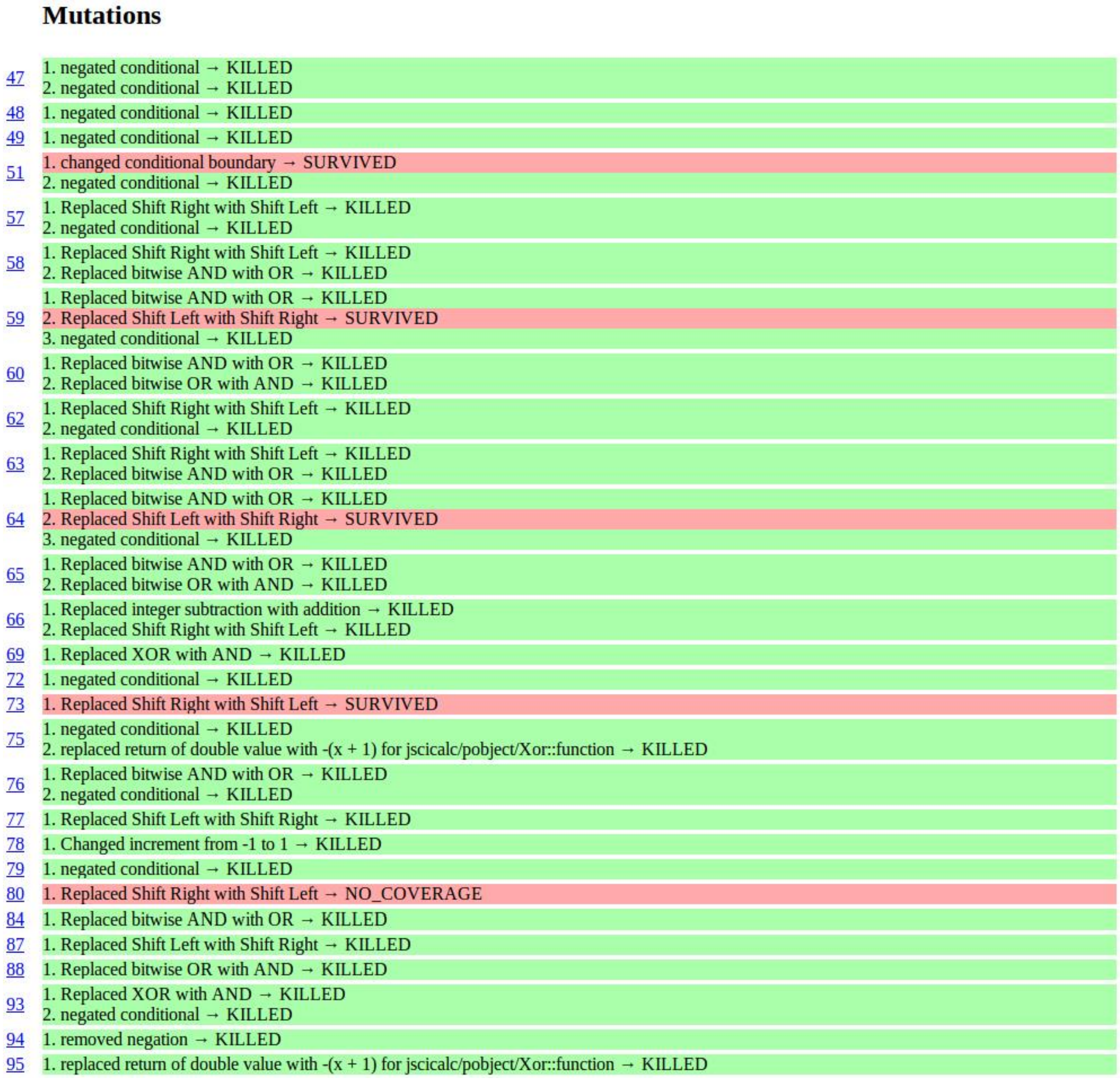
### E.4.3 AND



**Mutations**

| | |
|---|---|
| 47 | 1. negated conditional → KILLED<br>2. negated conditional → KILLED |
| 48 | 1. negated conditional → KILLED |
| 49 | 1. negated conditional → KILLED |
| 51 | 1. changed conditional boundary → SURVIVED<br>2. negated conditional → KILLED |
| 57 | 1. Replaced Shift Right with Shift Left → SURVIVED<br>2. negated conditional → KILLED |
| 58 | 1. Replaced Shift Right with Shift Left → KILLED<br>2. Replaced bitwise AND with OR → KILLED |
| 59 | 1. Replaced bitwise AND with OR → SURVIVED<br>2. Replaced Shift Left with Shift Right → SURVIVED<br>3. negated conditional → KILLED |
| 60 | 1. Replaced bitwise AND with OR → KILLED<br>2. Replaced bitwise OR with AND → KILLED |
| 62 | 1. Replaced Shift Right with Shift Left → SURVIVED<br>2. negated conditional → SURVIVED |
| 63 | 1. Replaced Shift Right with Shift Left → KILLED<br>2. Replaced bitwise AND with OR → KILLED |
| 64 | 1. Replaced bitwise AND with OR → SURVIVED<br>2. Replaced Shift Left with Shift Right → SURVIVED<br>3. negated conditional → KILLED |
| 65 | 1. Replaced bitwise AND with OR → KILLED<br>2. Replaced bitwise OR with AND → KILLED |
| 66 | 1. Replaced integer subtraction with addition → KILLED<br>2. Replaced Shift Right with Shift Left → KILLED |
| 69 | 1. Replaced bitwise AND with OR → KILLED |
| 72 | 1. negated conditional → KILLED |
| 73 | 1. Replaced Shift Right with Shift Left → SURVIVED |
| 75 | 1. negated conditional → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/And::function → KILLED |
| 76 | 1. Replaced bitwise AND with OR → KILLED<br>2. negated conditional → KILLED |
| 77 | 1. Replaced Shift Left with Shift Right → KILLED |
| 78 | 1. Changed increment from -1 to 1 → KILLED |
| 79 | 1. negated conditional → KILLED |
| 80 | 1. Replaced Shift Right with Shift Left → NO_COVERAGE |
| 84 | 1. Replaced bitwise AND with OR → KILLED |
| 87 | 1. Replaced Shift Left with Shift Right → KILLED |
| 88 | 1. Replaced bitwise OR with AND → KILLED |
| 93 | 1. Replaced bitwise AND with OR → KILLED<br>2. negated conditional → KILLED<br>3. negated conditional → SURVIVED<br>4. negated conditional → KILLED |
| 94 | 1. removed negation → NO_COVERAGE |
| 95 | 1. replaced return of double value with -(x + 1) for jscicalc/pobject/And::function → KILLED |

## E.4.4 XOR



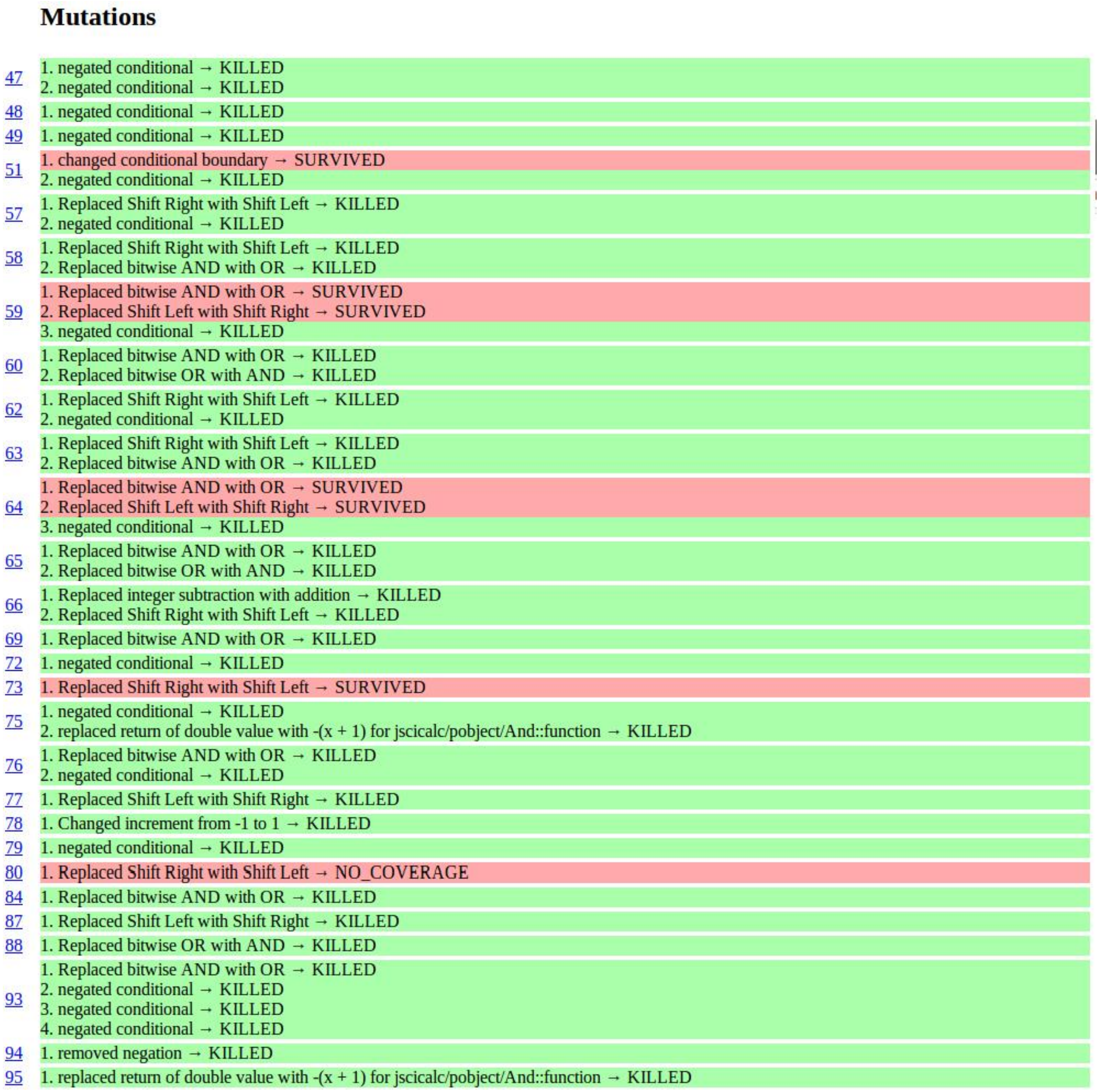
### Mutations

| | |
|---|---|
| 47 | 1. negated conditional → KILLED<br>2. negated conditional → KILLED |
| 48 | 1. negated conditional → KILLED |
| 49 | 1. negated conditional → KILLED |
| 51 | 1. changed conditional boundary → SURVIVED<br>2. negated conditional → KILLED |
| 57 | 1. Replaced Shift Right with Shift Left → KILLED<br>2. negated conditional → KILLED |
| 58 | 1. Replaced Shift Right with Shift Left → KILLED<br>2. Replaced bitwise AND with OR → KILLED |
| 59 | 1. Replaced bitwise AND with OR → KILLED<br>2. Replaced Shift Left with Shift Right → SURVIVED<br>3. negated conditional → KILLED |
| 60 | 1. Replaced bitwise AND with OR → KILLED<br>2. Replaced bitwise OR with AND → KILLED |
| 62 | 1. Replaced Shift Right with Shift Left → KILLED<br>2. negated conditional → KILLED |
| 63 | 1. Replaced Shift Right with Shift Left → KILLED<br>2. Replaced bitwise AND with OR → KILLED |
| 64 | 1. Replaced bitwise AND with OR → KILLED<br>2. Replaced Shift Left with Shift Right → SURVIVED<br>3. negated conditional → KILLED |
| 65 | 1. Replaced bitwise AND with OR → KILLED<br>2. Replaced bitwise OR with AND → KILLED |
| 66 | 1. Replaced integer subtraction with addition → KILLED<br>2. Replaced Shift Right with Shift Left → KILLED |
| 69 | 1. Replaced XOR with AND → KILLED |
| 72 | 1. negated conditional → KILLED |
| 73 | 1. Replaced Shift Right with Shift Left → SURVIVED |
| 75 | 1. negated conditional → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/Xor::function → KILLED |
| 76 | 1. Replaced bitwise AND with OR → KILLED<br>2. negated conditional → KILLED |
| 77 | 1. Replaced Shift Left with Shift Right → KILLED |
| 78 | 1. Changed increment from -1 to 1 → KILLED |
| 79 | 1. negated conditional → KILLED |
| 80 | 1. Replaced Shift Right with Shift Left → NO_COVERAGE |
| 84 | 1. Replaced bitwise AND with OR → KILLED |
| 87 | 1. Replaced Shift Left with Shift Right → KILLED |
| 88 | 1. Replaced bitwise OR with AND → KILLED |
| 93 | 1. Replaced XOR with AND → KILLED<br>2. negated conditional → KILLED |
| 94 | 1. removed negation → KILLED |
| 95 | 1. replaced return of double value with -(x + 1) for jscicalc/pobject/Xor::function → KILLED |

# Appendix F. – Post-Improved Code Coverage Results

## F.1 Will’s Improved Code Coverage Results

### F.1.1 AND



**Mutations**

| | |
|---|---|
| 47 | 1. negated conditional → KILLED<br>2. negated conditional → KILLED |
| 48 | 1. negated conditional → KILLED |
| 49 | 1. negated conditional → KILLED |
| 51 | 1. changed conditional boundary → SURVIVED<br>2. negated conditional → KILLED |
| 57 | 1. Replaced Shift Right with Shift Left → KILLED<br>2. negated conditional → KILLED |
| 58 | 1. Replaced Shift Right with Shift Left → KILLED<br>2. Replaced bitwise AND with OR → KILLED |
| 59 | 1. Replaced bitwise AND with OR → SURVIVED<br>2. Replaced Shift Left with Shift Right → SURVIVED<br>3. negated conditional → KILLED |
| 60 | 1. Replaced bitwise AND with OR → KILLED<br>2. Replaced bitwise OR with AND → KILLED |
| 62 | 1. Replaced Shift Right with Shift Left → KILLED<br>2. negated conditional → KILLED |
| 63 | 1. Replaced Shift Right with Shift Left → KILLED<br>2. Replaced bitwise AND with OR → KILLED |
| 64 | 1. Replaced bitwise AND with OR → SURVIVED<br>2. Replaced Shift Left with Shift Right → SURVIVED<br>3. negated conditional → KILLED |
| 65 | 1. Replaced bitwise AND with OR → KILLED<br>2. Replaced bitwise OR with AND → KILLED |
| 66 | 1. Replaced integer subtraction with addition → KILLED<br>2. Replaced Shift Right with Shift Left → KILLED |
| 69 | 1. Replaced bitwise AND with OR → KILLED |
| 72 | 1. negated conditional → KILLED |
| 73 | 1. Replaced Shift Right with Shift Left → SURVIVED |
| 75 | 1. negated conditional → KILLED<br>2. replaced return of double value with -(x + 1) for jscicalc/pobject/And::function → KILLED |
| 76 | 1. Replaced bitwise AND with OR → KILLED<br>2. negated conditional → KILLED |
| 77 | 1. Replaced Shift Left with Shift Right → KILLED |
| 78 | 1. Changed increment from -1 to 1 → KILLED |
| 79 | 1. negated conditional → KILLED |
| 80 | 1. Replaced Shift Right with Shift Left → NO_COVERAGE |
| 84 | 1. Replaced bitwise AND with OR → KILLED |
| 87 | 1. Replaced Shift Left with Shift Right → KILLED |
| 88 | 1. Replaced bitwise OR with AND → KILLED |
| 93 | 1. Replaced bitwise AND with OR → KILLED<br>2. negated conditional → KILLED<br>3. negated conditional → KILLED<br>4. negated conditional → KILLED |
| 94 | 1. removed negation → KILLED |
| 95 | 1. replaced return of double value with -(x + 1) for jscicalc/pobject/And::function → KILLED |

# Appendix G. – GUI Testing Results

## G.1 Team A Results

| Test Description | Mac OSX<br>Results | Windows<br>Results |
|---|---|---|
| Adding | Pass | Pass |
| Subbing | Pass | Pass |
| Multiplying | Pass | Pass |
| Dividing | Pass | Pass |
| Clearing Values | Pass | Pass |
| Deleting values | Pass | Pass |
| Powering Calculation On/Off | Pass | Pass |
| $x^{-1}$ | Pass | Pass |
| $x^2$ | Pass | Pass |
| Root | Pass | Pass |
| Sin | Pass | Pass |
| Cos | Fail | Fail |
| Tan | Fail | Fail |
| Log | Pass | Pass |
| Ln | Pass | Pass |
| ^ | Pass | Pass |
| nCr | Pass | Pass |
| i | Fail | Fail |
| Pi | Pass | Pass |
| Mode | Pass | Pass |
| Shift | Pass | Pass |
| STO | Pass | Pass |
| RCL | Pass | Pass |
| M+ | Pass | Pass |
| Braces | Pass | Pass |
| Up arrow | Pass | Pass |
| Down Arrow | Pass | Pass |
| Left Arrow | Pass | Pass |
| Right Arrow | Pass | Pass |
| ? | Pass | Pass |
| Numbers | Pass | Pass |
| . | Pass | Pass |
| e | Pass | Pass |

### G.2 Team B Results

| Test Description | Linux Results |
|---|---|
| $x^3$ | Pass |
| $\sqrt[3]{n}$ | Pass |
| sin$^{-1}$ | Fail |
| cos$^{-1}$ | Pass |
| tan-1 | Pass |
| 10x | Pass |
| exp | Pass |
| nCr | Pass |

# Appendix H. – Integration Testing Results

### *Results Summary for section 6.2.2*

| Test Case (log(x!)) | Pass/Fail | Results (as needed) |
|---|---|---|
| X=5 | Fail | java.lang.AssertionError: Log-Factorial Test: 5 expected:<4.78749> but was:<2.0791812460476247> |
| X=0 | Pass | N/A |
| X=20 | Fail | java.lang.AssertionError: Log-Factorial Test: 20 expected:<42.33561> but was:<18.386124616877712> |
| X=2.5 | Fail | java.lang.AssertionError: Arithmetic Exception thrown during factorial test of 2.5 |
| X=-5 | Pass | N/A |

### *Results Summary for section 6.2.3*

| Test Case | Expected | Actual | Pass/Fail |
|---|---|---|---|
| AND(0 , Tan(180)) | 0 | 0 | Pass |
| XOR(0, SIN(90)) | 1 | 1 | Pass |
| LN(COS(0)) | 0 | 0 | Pass |
| TENX(ACOS(1)) | 1 | 1 | Pass |

# Appendix I. – Partial Bug Report

## I.1 Cosine Bug

| | |
|---|---|
| **Bug Name** | Cosine of 90 - Incorrect Answer |
| **Bug ID** | - |
| **Area Path** | - |
| **Build Number** | 2:0.5 |
| **Severity** | Minor |
| **Priority** | Minor |
| **Reported By** | George |
| **Reported On** | March 30, 2014 |
| **Reason** | Unexpected result |
| **State** | Active |
| **Environment** | Mac OSX Mavericks 10.9.2/Windows 8.1 |
| **Description** | Cos(90) ~=0 on the calculator, it shows long decimal value: 0.0000000061; |

| | |
|---|---|
| | should just be 0 |
| **Steps to Reproduce** | cos →9→0→= |
| **Expected Result** | 0 |

### I.2 Tangent Bug

| | |
|---|---|
| **Bug Name** | Tangent of 180 - Incorrect Answer |
| **Bug ID** | - |
| **Area Path** | - |
| **Build Number** | 2:0.5 |
| **Severity** | Minor |
| **Priority** | Minor |
| **Reported By** | George/Yasir |
| **Reported On** | March 30, 2014 |
| **Reason** | Unexpected result |
| **State** | Active |
| **Environment** | Mac OSX Mavericks 10.9.2/Windows 8.1 |
| **Description** | Tan returns small negative values on Tan(180) & Tan(360), they should be just 0 |
| **Steps to Reproduce** | tan →1→8→0→= |
| **Expected Result** | 0 |

## I.3 Combination Bug

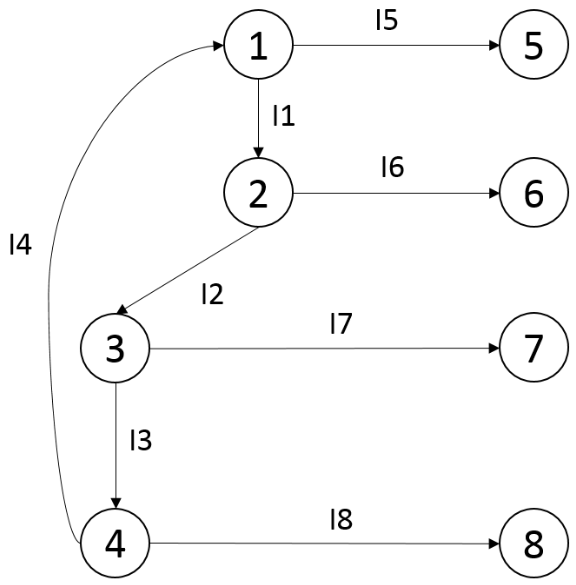
| | |
|---|---|
| **Bug Name** | Sequential use of Combination and i produces Incorrect Answer Error |
| **Bug ID** | - |
| **Area Path** | - |
| **Build Number** | 2:0.5 |
| **Severity** | Minor |
| **Priority** | Minor |
| **Reported By** | George |
| **Reported On** | March 30, 2014 |
| **Reason** | Unexpected behaviour |
| **State** | Active |
| **Environment** | Mac OSX Mavericks 10.9.2/Windows 8.1 |
| **Description** | 5C-3 produces an error, and the usage of i sequentially produced the wrong answer (25i) |
| **Steps to Reproduce** | 5→ nCr→-3→=→5→i→= |
| **Expected Result** | Error; 5i |

## I.4 Inverse Sine Bug

| | |
|---|---|
| **Bug Name** | Inverse Sine Error Answer |
| **Bug ID** | - |
| **Area Path** | SHIFT |
| **Build Number** | 2:0.5 |
| **Severity** | Minor |
| **Priority** | Minor |
| **Reported By** | Cory |
| **Reported On** | March 30, 2014 |
| **Reason** | Unexpected behavior and result |
| **State** | Active |
| **Environment** | Linux Mint 16 |
| **Description** | $\text{Sin}^{-1}$(30) produces an error for an answer. If the equal key is pressed again, the answer of 30 will be produced. |
| **Steps to Reproduce** | SHIFT → $\sin^{-1}$→3→0→=→= |
| **Expected Result** | NaN |

# Appendix J. – Example of Test Execution Log

## J.1 Combination State Diagram



Where I1 – I8 are inputs, 1-8 are states of the GUI

| Input # | Input | Expected Output | Actual Output | Notes |
|---|---|---|---|---|
| I1 | Click 3 button | 3 should be displayed in calculator output | 3 is displayed in calculator output | |
| I2 | Click **nCr** button | 3C Should be displayed | 3C Is displayed | |
| I3 | Click **2** button | 3C2 should be displayed | 3C2 is displayed | |
| I4 | Click **=** button | 3 should be displayed as the answer | 3 is displayed as the answer | |
| I5 | Click = button when no new input has been entered | Displays previous answer | Displays previous answer | |
| I6 | Click = button without adding nCr | Display 3 as answer | Display 3 as answer | |
| I7 | Click a button that is not a valid input for the | Error message is displayed | Error message is displayed | |

| | | | | |
|---|---|---|---|---|
| | combination function | | | |
| I8 | Add additional inputs | Displays additional outputs | Displays additional outputs | You can create equations in the calculator with an unlimited amount of possibilities |