Mutation Testing

Milestone 4

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

ABSTRACT

Sometimes tests don't fail correctly with the code they test: this could be caused by a mistaken assumption, or maybe because of a slight oversight. Mutation testing is an effective, automated way of enforcing that tests fail correctly. The tests used in Milestone 3 will undergo mutation testing and the results and recommendations are documented in this paper.

Table of Contents

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 plugin 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.

1.1 George's Initial Results

Class Name	Mutation Coverage
Cube	100%
Square	100%
Inverse	100%
Factorial	100%

1.2 Yasir's Initial Results

Class Name	Mutation Coverage
Inverse Cosine	62%
Cosine	60%
Sine	60%
Tangent	60%

1.3 Cory's Initial Results

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

1.4 Will's Initial Results

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

2. Process of Improving Test Effectiveness

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:

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

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:
    - Reachability – location must be feasible
    - Propagation – the incorrect state, caused by mutant, must propagate to output and must be checked by test suite

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

3. In-Depth Analysis

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.

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%)

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.

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 Yasir:

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

3.5 Analysis of specific Mutants, or classes of Mutants

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.

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.

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.

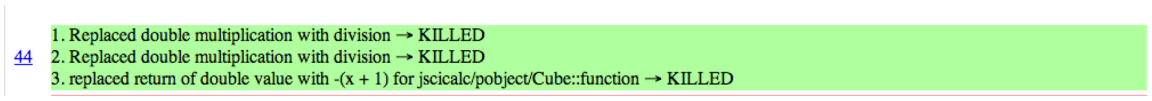
So in conclusion, only one of our members has increased their effectiveness of their tests significantly. The remainder of our team experienced little to no improvement to effectiveness to their tests. Therefore, mutation testing should only be used if there is time for it in the project development 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.

Appendix A. – Initial Code Coverage Results

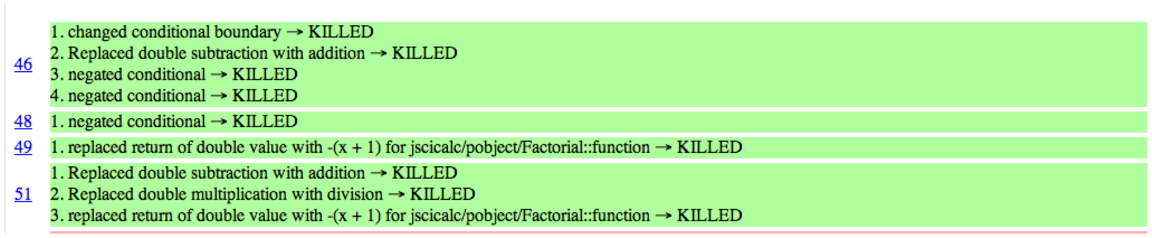
A.1 George's Initial Mutation Results

A.1.1 Cube



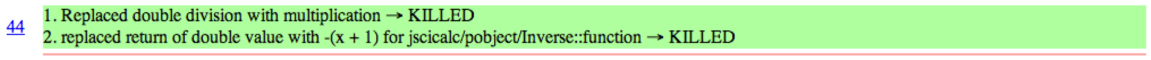
```
44  1. Replaced double multiplication with division → KILLED
    2. Replaced double multiplication with division → KILLED
    3. replaced return of double value with -(x + 1) for jscicalc/pobject/Cube::function → KILLED
```

A.1.2 Factorial



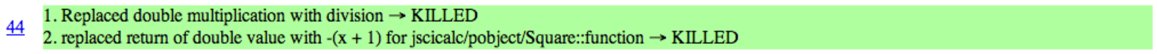
```
46  1. changed conditional boundary → KILLED
    2. Replaced double subtraction with addition → KILLED
    3. negated conditional → KILLED
    4. negated conditional → KILLED
48  1. negated conditional → KILLED
49  1. replaced return of double value with -(x + 1) for jscicalc/pobject/Factorial::function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

A.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
```

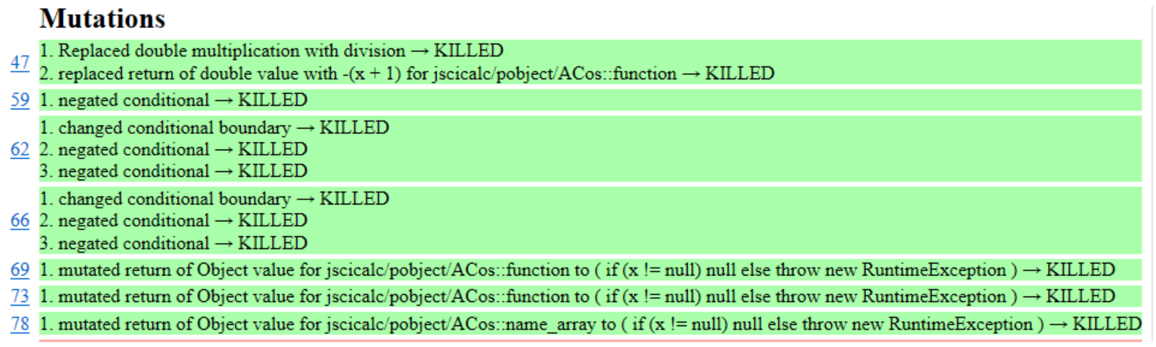
A.1.4 Square



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

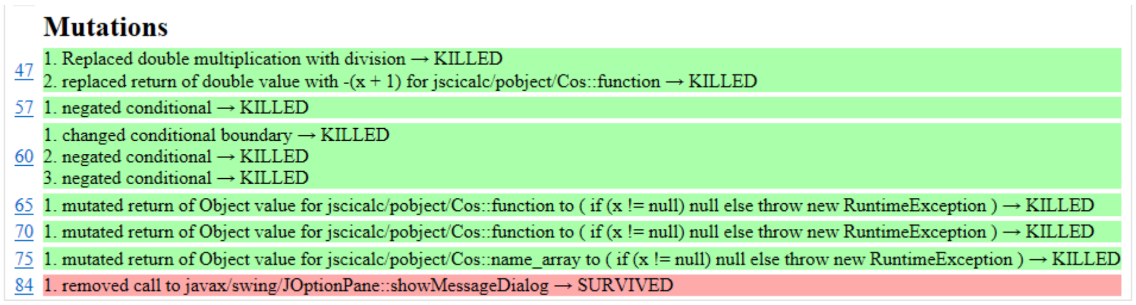
A.2 Yasir's Initial Mutation Results

A.2.1 Inverse Cosine



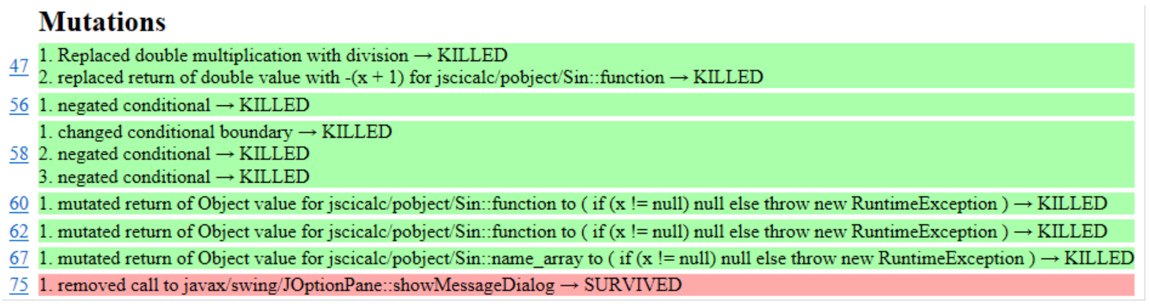
	Mutations
47	1. Replaced double multiplication with division → KILLED 2. replaced return of double value with -(x + 1) for jscicalc/pobject/ACos::function → KILLED
59	1. negated conditional → KILLED
62	1. changed conditional boundary → KILLED 2. negated conditional → KILLED 3. negated conditional → KILLED
66	1. changed conditional boundary → KILLED 2. negated conditional → KILLED 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

A.2.2 Cosine



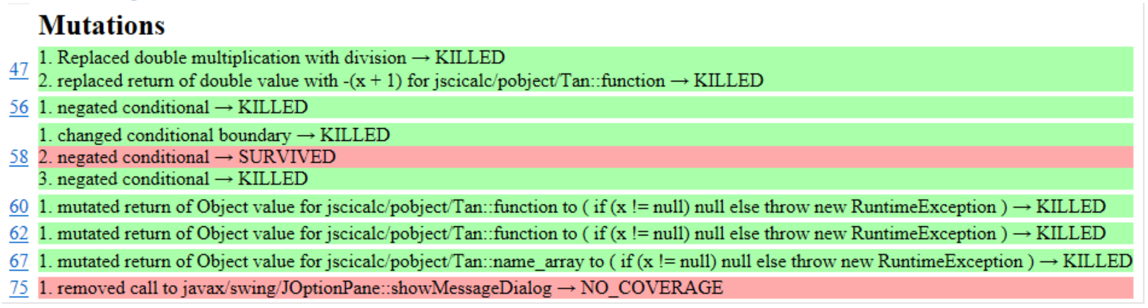
	Mutations
47	1. Replaced double multiplication with division → KILLED 2. replaced return of double value with -(x + 1) for jscicalc/pobject/Cos::function → KILLED
57	1. negated conditional → KILLED
60	1. changed conditional boundary → KILLED 2. negated conditional → KILLED 3. negated conditional → KILLED
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

A.2.3 Sine



	Mutations
47	1. Replaced double multiplication with division → KILLED 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 2. negated conditional → KILLED 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

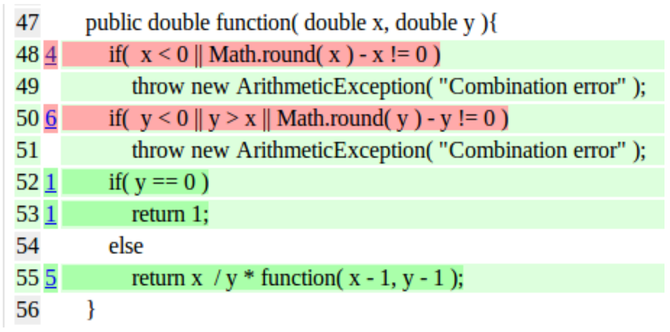
A.2.4 Tangent



	Mutations
47	1. Replaced double multiplication with division → KILLED 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 2. negated conditional → SURVIVED 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

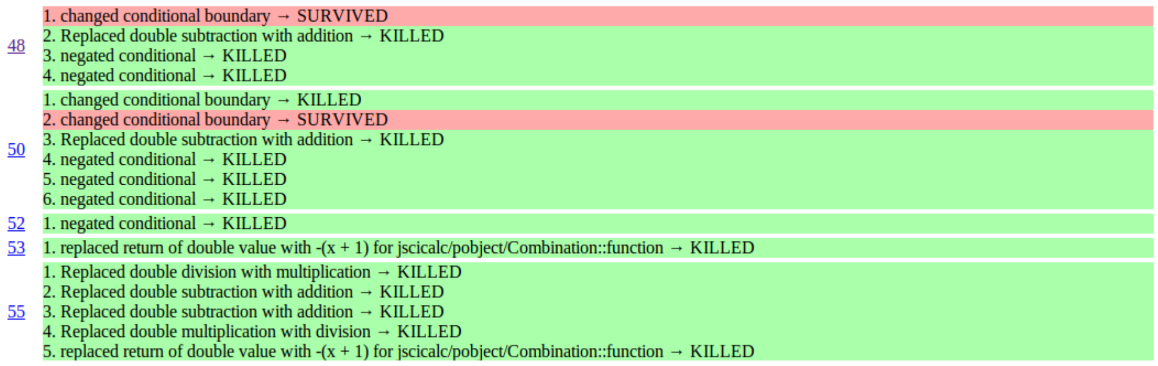
A.3 Cory's Initial Mutation Results

A.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      }
```

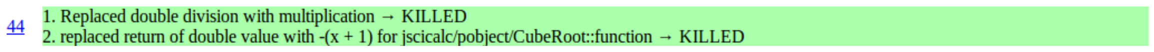
Mutations



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

A.3.2 Cube Root

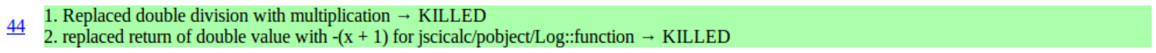
Mutations



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

A.3.3 Logarithmic (Base 10)

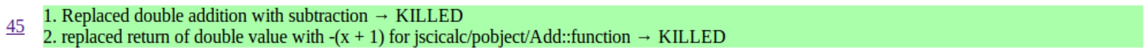
Mutations



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

A.3.4 Addition

Mutations



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

A.4 Will’s Initial Mutation Results

A.4.1 Natural Logarithmic

Mutations

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

A.4.2 Inverse Logarithmic

Mutations

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

A.4.3 AND

Mutations

47	1. negated conditional → KILLED 2. negated conditional → KILLED
48	1. negated conditional → KILLED
49	1. negated conditional → KILLED
51	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
57	1. Replaced Shift Right with Shift Left → SURVIVED 2. negated conditional → KILLED
58	1. Replaced Shift Right with Shift Left → KILLED 2. Replaced bitwise AND with OR → KILLED
59	1. Replaced bitwise AND with OR → SURVIVED 2. Replaced Shift Left with Shift Right → SURVIVED 3. negated conditional → KILLED
60	1. Replaced bitwise AND with OR → KILLED 2. Replaced bitwise OR with AND → KILLED
62	1. Replaced Shift Right with Shift Left → SURVIVED 2. negated conditional → SURVIVED
63	1. Replaced Shift Right with Shift Left → KILLED 2. Replaced bitwise AND with OR → KILLED
64	1. Replaced bitwise AND with OR → SURVIVED 2. Replaced Shift Left with Shift Right → SURVIVED 3. negated conditional → KILLED
65	1. Replaced bitwise AND with OR → KILLED 2. Replaced bitwise OR with AND → KILLED
66	1. Replaced integer subtraction with addition → KILLED 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 2. replaced return of double value with -(x + 1) for jscicalc/pobject/And::function → KILLED
76	1. Replaced bitwise AND with OR → KILLED 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 2. negated conditional → KILLED 3. negated conditional → SURVIVED 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

A.4.4 XOR

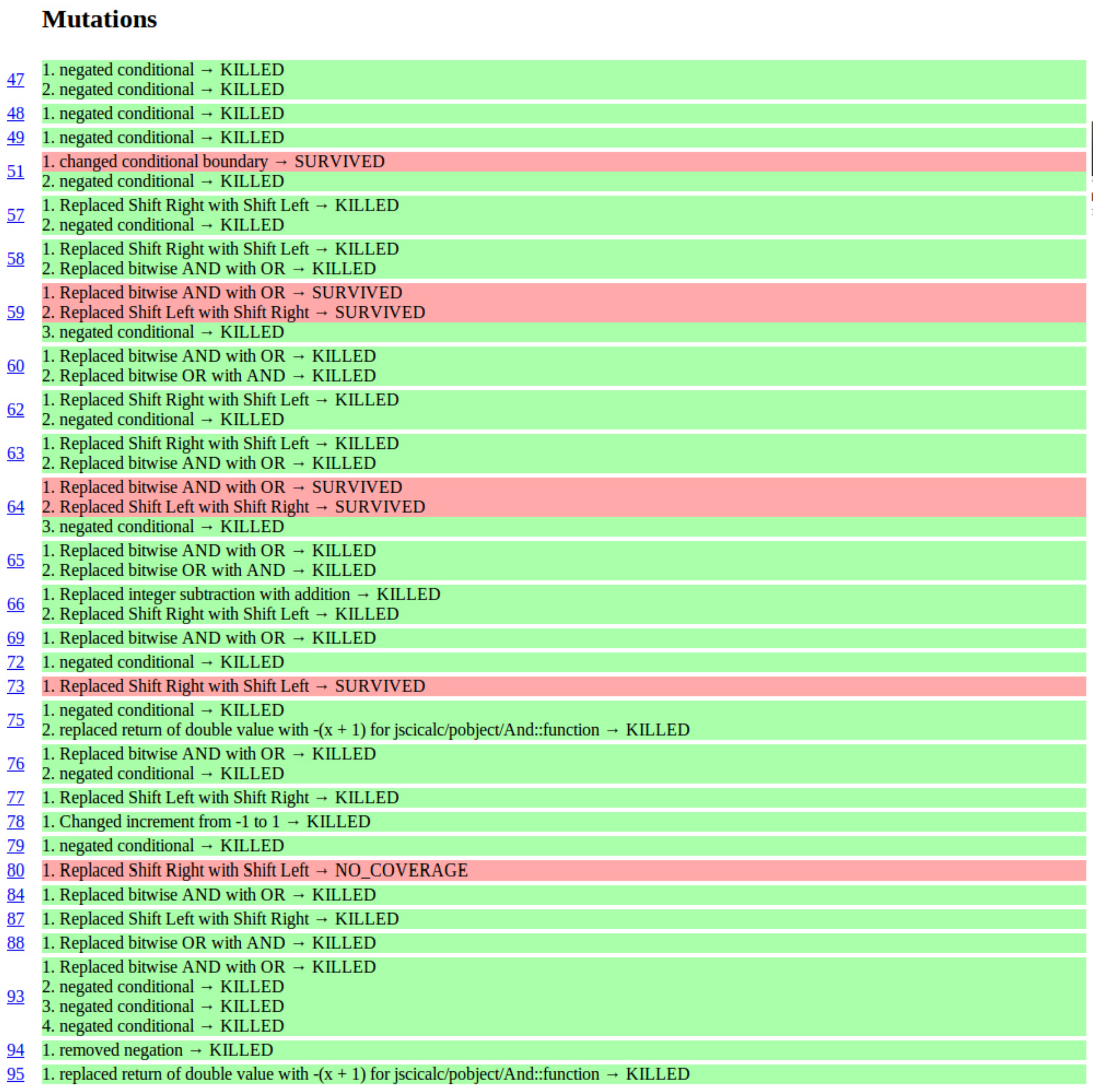
Mutations

47	1. negated conditional → KILLED 2. negated conditional → KILLED
48	1. negated conditional → KILLED
49	1. negated conditional → KILLED
51	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
57	1. Replaced Shift Right with Shift Left → KILLED 2. negated conditional → KILLED
58	1. Replaced Shift Right with Shift Left → KILLED 2. Replaced bitwise AND with OR → KILLED
59	1. Replaced bitwise AND with OR → KILLED 2. Replaced Shift Left with Shift Right → SURVIVED 3. negated conditional → KILLED
60	1. Replaced bitwise AND with OR → KILLED 2. Replaced bitwise OR with AND → KILLED
62	1. Replaced Shift Right with Shift Left → KILLED 2. negated conditional → KILLED
63	1. Replaced Shift Right with Shift Left → KILLED 2. Replaced bitwise AND with OR → KILLED
64	1. Replaced bitwise AND with OR → KILLED 2. Replaced Shift Left with Shift Right → SURVIVED 3. negated conditional → KILLED
65	1. Replaced bitwise AND with OR → KILLED 2. Replaced bitwise OR with AND → KILLED
66	1. Replaced integer subtraction with addition → KILLED 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 2. replaced return of double value with -(x + 1) for jscicalc/pobject/Xor::function → KILLED
76	1. Replaced bitwise AND with OR → KILLED 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 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 B. – Post-Improved Code Coverage Results

B.1 Will's Improved Code Coverage Results

B.1.1 AND



Mutations

47	1. negated conditional → KILLED 2. negated conditional → KILLED
48	1. negated conditional → KILLED
49	1. negated conditional → KILLED
51	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
57	1. Replaced Shift Right with Shift Left → KILLED 2. negated conditional → KILLED
58	1. Replaced Shift Right with Shift Left → KILLED 2. Replaced bitwise AND with OR → KILLED
59	1. Replaced bitwise AND with OR → SURVIVED 2. Replaced Shift Left with Shift Right → SURVIVED 3. negated conditional → KILLED
60	1. Replaced bitwise AND with OR → KILLED 2. Replaced bitwise OR with AND → KILLED
62	1. Replaced Shift Right with Shift Left → KILLED 2. negated conditional → KILLED
63	1. Replaced Shift Right with Shift Left → KILLED 2. Replaced bitwise AND with OR → KILLED
64	1. Replaced bitwise AND with OR → SURVIVED 2. Replaced Shift Left with Shift Right → SURVIVED 3. negated conditional → KILLED
65	1. Replaced bitwise AND with OR → KILLED 2. Replaced bitwise OR with AND → KILLED
66	1. Replaced integer subtraction with addition → KILLED 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 2. replaced return of double value with -(x + 1) for jscicalc/pobject/And::function → KILLED
76	1. Replaced bitwise AND with OR → KILLED 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 2. negated conditional → KILLED 3. negated conditional → KILLED 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