Title: Test Tool Evaluation, Software Process Quality Assignment 2

Name: Aniket S. Angwalkar

Student ID: 23269100

Program: M.Sc. Computing (Secure Software Engineering)

Module: CA650 Software Process Quality

Email: aniket.angwalkar2@mail.dcu.ie

I. INTRODUCTION

This report is a demonstration and discussion of the Mutation Testing with an example program. The tool I will be utilizing is widely known as PIT short for Pit Mutation Testing. PIT is an open-source mutation testing tool for Java based programs. This report will be testing a Binary Search based algorithm implementation in Java. Further we will be adding a few simple test cases. We will be discussing the concepts of mutation testing with the example and understand the advantages and disadvantages of mutation testing over the usual unit testing techniques.

II. Mutation Testing

Mutation is the basic but significant alteration to some form or nature of an object. Utilizing this concept, Mutation testing is a technique that evaluates the ability of software tests to detect changes in the code by adding small changes, or mutations, and seeing if the tests detect them. The purpose of mutation testing is to mimic code errors, or mutations, and see if the test suite can detect these errors.

The usual ways of Unit Testing a program works by detecting different behaviors of software programs with the help of failed test cases. Mutation testing on the other hand introduces failures or defects to the program during the test execution. If the mutation or the defect is identified, it means that the mutation has been "killed" implying that the test suite has effectively identified the change and the code will very likely similar such changes or behaviors in the code.

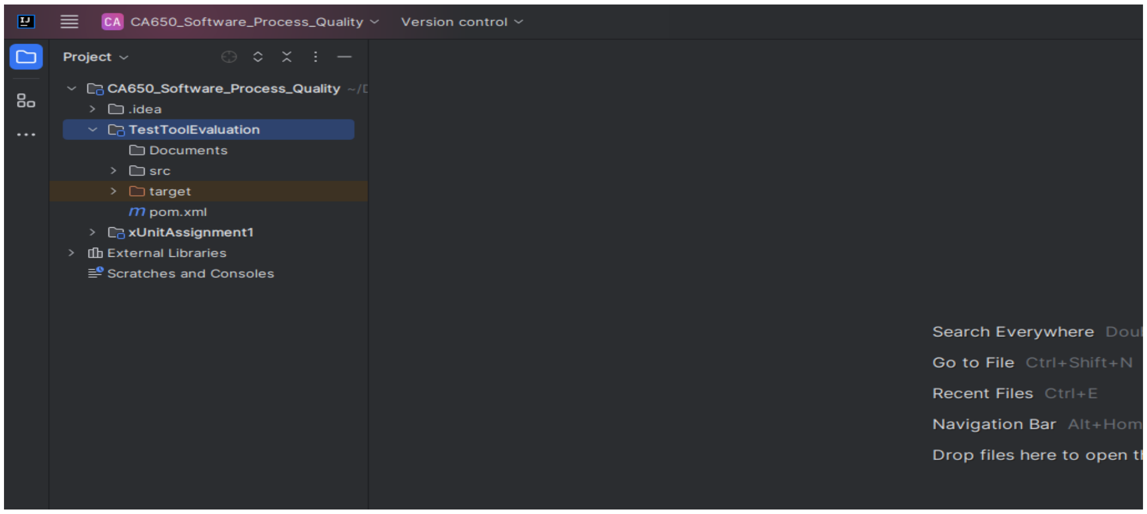
III. PITest

PIT is an open source, easy to setup and use tool for testing Java mutations. The tool supports both Maven and Gradle projects and can also be implemented in standalone java programs for testing using IDEs such as IntelliJ. While it lacks some of the features of some of the more well-known tools for research, like muJava and Major, it sufficiently makes itself it more appealing because it is actively maintained as well as integrates well with development tools like Ant or Maven. It can be invoked via the command line. PIT offers several benefits, including scalability and support for mutant operators that meet the most recent advancements in mutation research.

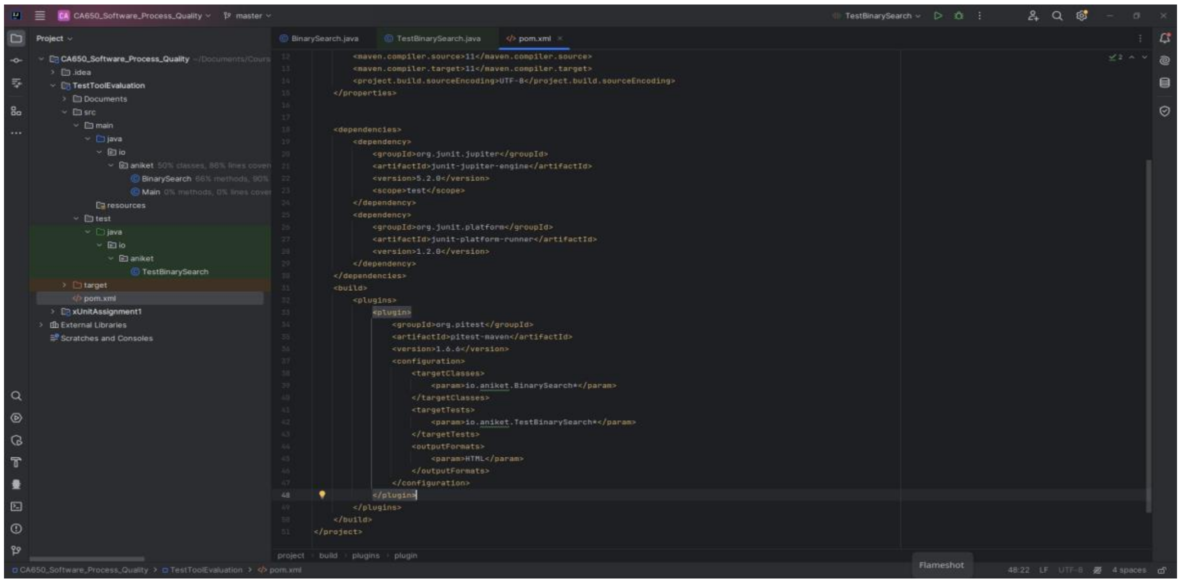
IV. Setup

PIT is fairly easy to setup in a Java project environment. For this demonstration and the test cases, I have added a init function that executes some steps before running the test cases to create multiple arrays to cover multiple scenarios. I've outlined each of the steps for the setup of the project and test environment with screenshots. The steps are as follows:

1. We first created a Maven based Java project named TestToolEvaluation using IntelliJ Idea.

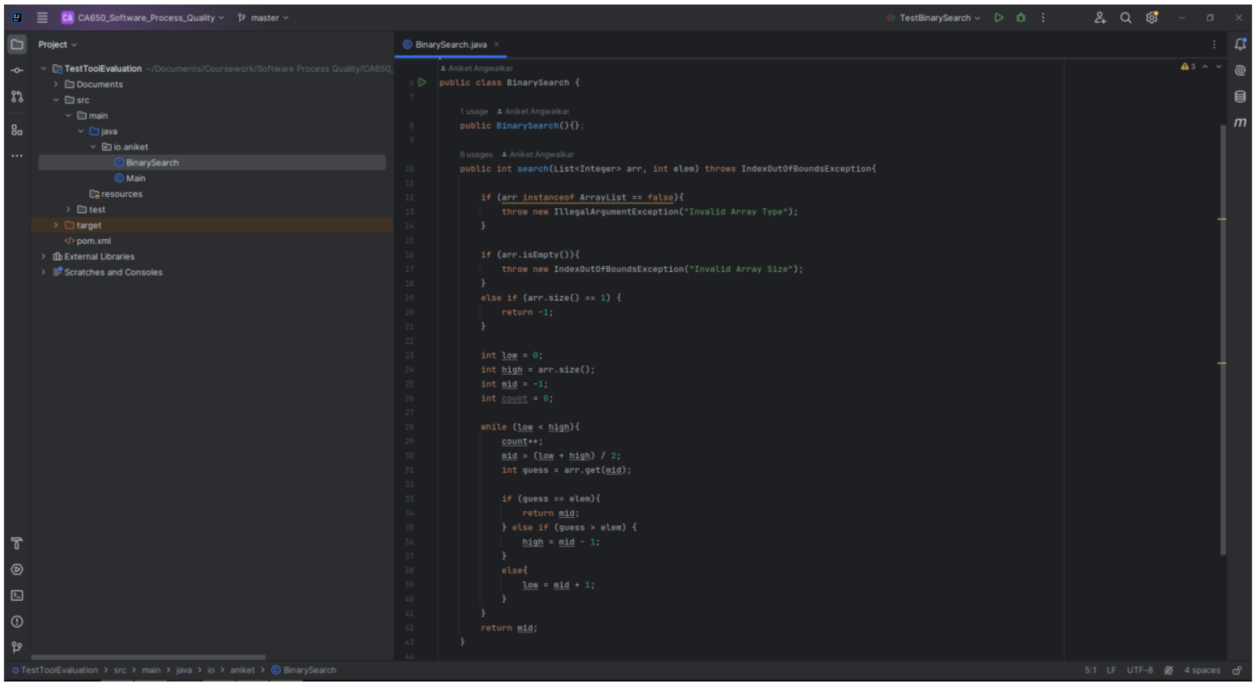


2. Notice that we now have a pom.xml file. This is where we will add our project dependencies for the PITest tool and JUnit testing tool.



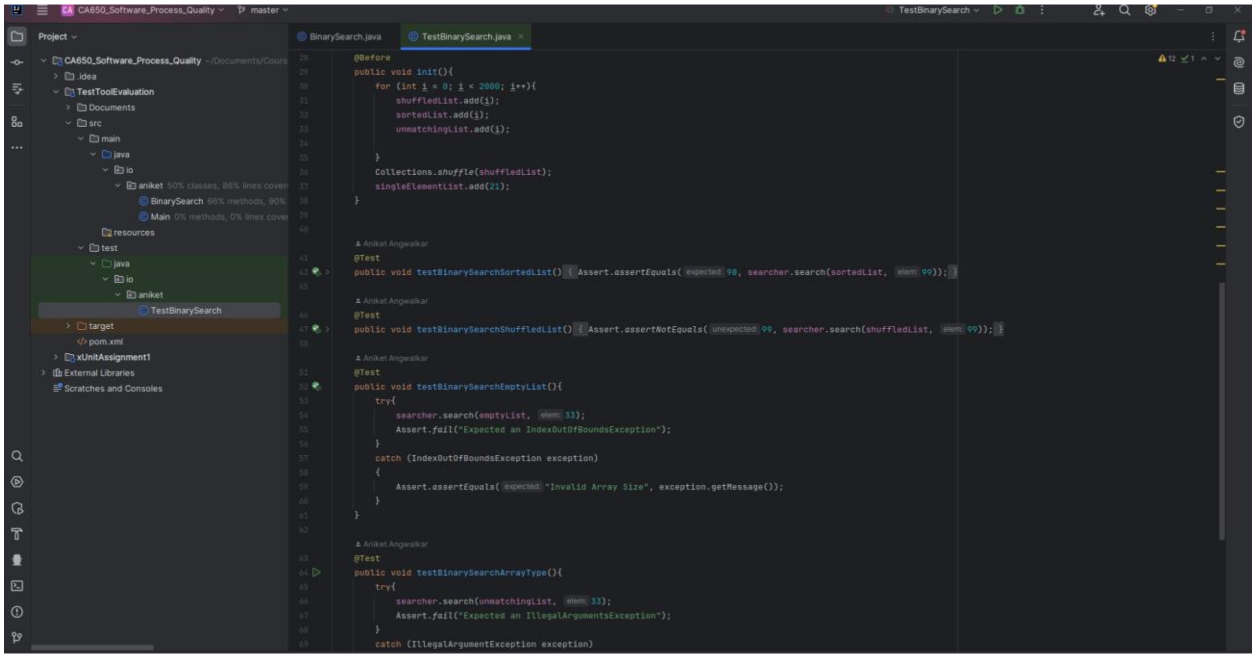
```
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>


    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.2.0</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-runner</artifactId>
            <version>1.2.0</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.pitest</groupId>
                <artifactId>pitest-maven</artifactId>
                <version>1.6.6</version>
                <configuration>
                    <targetClasses>
                        <param>io.aniket.BinarySearch*</param>
                    </targetClasses>
                    <targetTests>
                        <param>io.aniket.TestBinarySearch*</param>
                    </targetTests>
                    <outputFormats>
                        <param>HTML</param>
                    </outputFormats>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

3. In this step, I have added some code to that I will further test. This code is the Binary Search algorithm as follows:



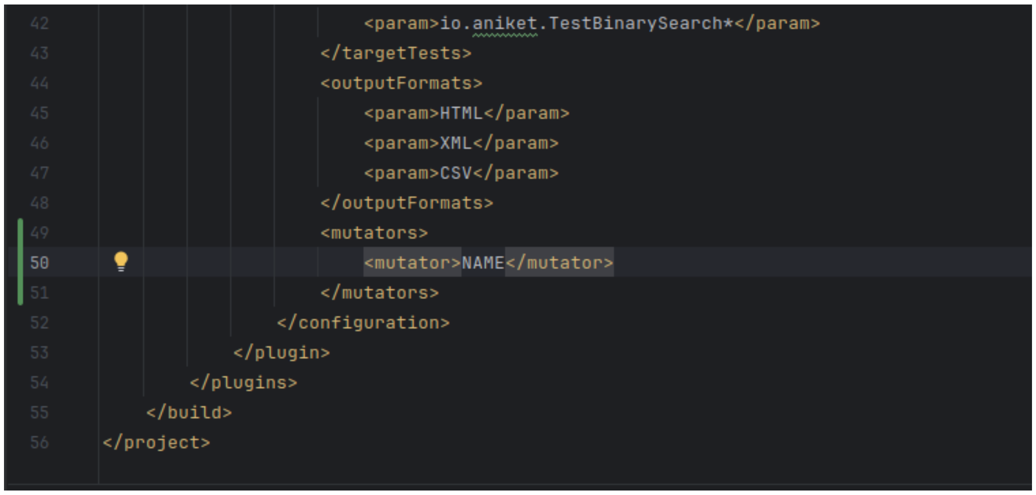
```java
public class BinarySearch {

    public BinarySearch(){};

    public int search(List<Integer> arr, int elem) throws IndexOutOfBoundsException{

        if (arr instanceof ArrayList == false){
            throw new IllegalArgumentException("Invalid Array Type");
        }

        if (arr.isEmpty()){
            throw new IndexOutOfBoundsException("Invalid Array Size");
        }
        else if (arr.size() == 1) {
            return -1;
        }

        int low = 0;
        int high = arr.size();
        int mid = -1;
        int count = 0;

        while (low < high){
            count++;
            mid = (low + high) / 2;
            int guess = arr.get(mid);

            if (guess == elem){
                return mid;
            } else if (guess > elem) {
                high = mid - 1;
            }
            else{
                low = mid + 1;
            }
        }
        return mid;
    }
```

4. Now we add some test cases for testing the program and various scenarios



```java
    @Before
    public void init(){
        for (int i = 0; i < 2000; i++){
            shuffledList.add(i);
            sortedList.add(i);
            unmatchingList.add(i);

        }
        Collections.shuffle(shuffledList);
        singleElementList.add(21);
    }


    @Test
    public void testBinarySearchSortedList() { Assert.assertEquals(98, searcher.search(sortedList, 99)); }

    @Test
    public void testBinarySearchShuffledList() { Assert.assertNotEquals(99, searcher.search(shuffledList, 99)); }

    @Test
    public void testBinarySearchEmptyList(){
        try{
            searcher.search(emptyList, 33);
            Assert.fail("Expected an IndexOutOfBoundsException");
        }
        catch (IndexOutOfBoundsException exception)
        {
            Assert.assertEquals("Invalid Array Size", exception.getMessage());
        }
    }

    @Test
    public void testBinarySearchArrayType(){
        try{
            searcher.search(unmatchingList, 33);
            Assert.fail("Expected an IllegalArgumentsException");
        }
        catch (IllegalArgumentException exception)
```

5. It is very easy to add / remove any specific mutators that we need, we can do so in the PIT plugin in the pom.xml file as follows:



```
                        <param>io.aniket.TestBinarySearch*</param>
                    </targetTests>
                    <outputFormats>
                        <param>HTML</param>
                        <param>XML</param>
                        <param>CSV</param>
                    </outputFormats>
                    <mutators>
                        <mutator>NAME</mutator>
                    </mutators>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

V. PITest Mutation Specifics

The process and steps involved in using PITest for Mutation Testing are summarized as follows:

1. Code Instrumentation:

The source code to be tested is first instrumented by PITest. By instrumentation, it means that to track which sections of the code are run during testing, it adds extra code to the original source code.

2. Mutation Operators

Mutation operators are rules that specify how changes are added to the source code. For instance, method call removal, negating conditions, and replacement of arithmetic operations.

3. Generation of Mutations

PITest uses mutation operators to create mutated versions of the code after it has been instrumented. Every mutant is a possible flaw in the original code.

4. Test Execution

Subsequently, PITest applies the current test suite to the mutated versions. It tracks where the tests are failing (i.e., mutants are killed) and which tests are successful (i.e., mutants are not killed).

5. Analysis of Mutations

PITest computes mutation scores based on test results. The percentage of mutants eliminated by the test suite is shown by these scores. A low score denotes test flaws, whereas a high mutation score indicates that the tests are successful in finding errors.

6. Report Generation

Ultimately, PITest produces comprehensive reports that include coverage data, identified mutations, and mutation score metrics. These reports aid developers in prioritizing their testing efforts and identifying areas in which their test suite needs to be improved.

VI. Usage

The program we will be testing is a Binary Search algorithm. This program takes two arguments, an array of integers to search elements from, and an integer to be searched for in the array. This program consists of one function called search which processes the search and returns the result.

The test cases are as follows:

```
testBinarySearchSortedList
```

```
testBinarySearchShuffledList
```

```
testBinarySearchEmptyList
```

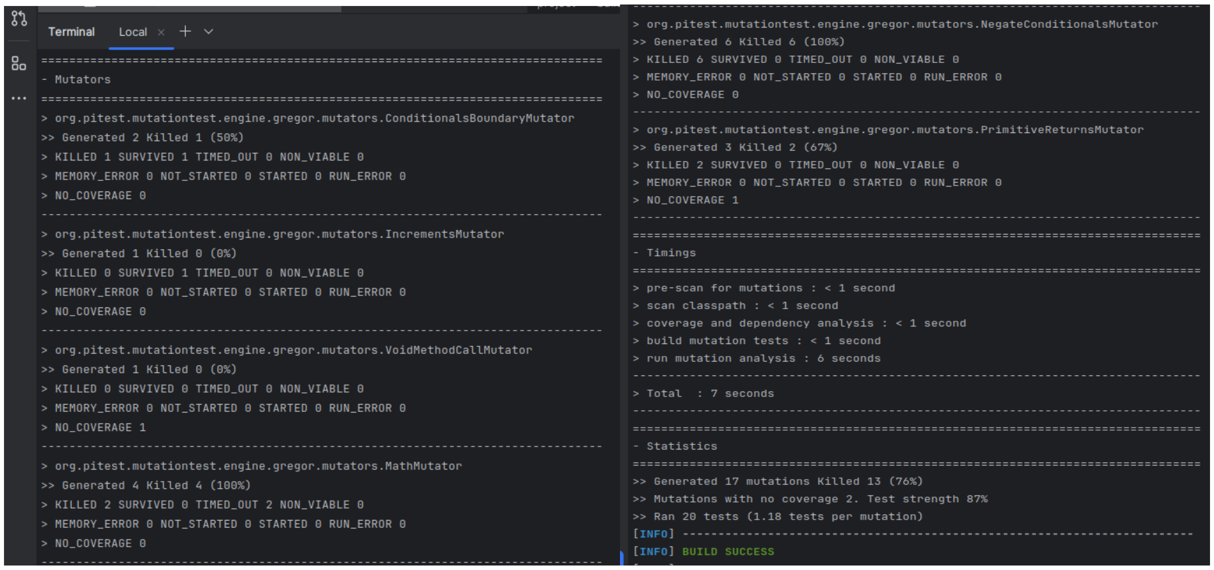
```
testBinarySearchArrayType
```

```
testBinarySearchSingleElementList
```

The test cases make use of jUnit test framework as PIT makes the use of jUnit as the basis for it’s tests and then generates the reports.

In order to execute the tests, we execute the following command in the same folder as the pom.xml file. The command executes the test cases using maven compilation features and generates mutators to be tested as follows:

mvn clean test org.pitest:pitest-maven:mutationCoverage

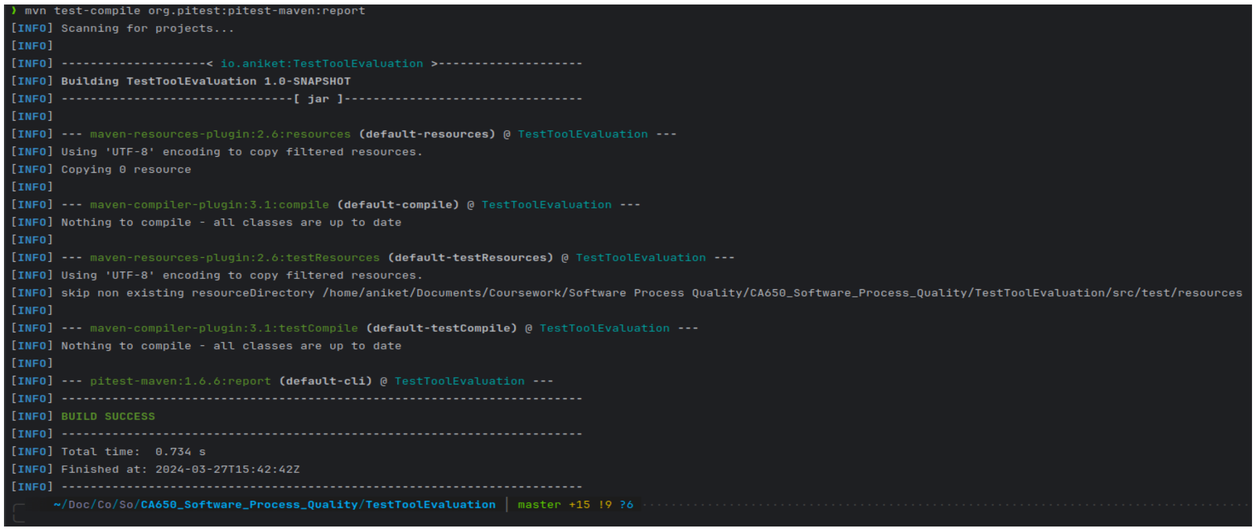


```
================================================================================
- Mutators
================================================================================
> org.pitest.mutationtest.engine.gregor.mutators.ConditionalsBoundaryMutator
>> Generated 2 Killed 1 (50%)
> KILLED 1 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.IncrementsMutator
>> Generated 1 Killed 0 (0%)
> KILLED 0 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.VoidMethodCallMutator
>> Generated 1 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 1
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.MathMutator
>> Generated 4 Killed 4 (100%)
> KILLED 2 SURVIVED 0 TIMED_OUT 2 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.NegateConditionalsMutator
>> Generated 6 Killed 6 (100%)
> KILLED 6 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
--------------------------------------------------------------------------------
> org.pitest.mutationtest.engine.gregor.mutators.PrimitiveReturnsMutator
>> Generated 3 Killed 2 (67%)
> KILLED 2 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 1
--------------------------------------------------------------------------------
================================================================================
- Timings
================================================================================
> pre-scan for mutations : < 1 second
> scan classpath : < 1 second
> coverage and dependency analysis : < 1 second
> build mutation tests : < 1 second
> run mutation analysis : 6 seconds
--------------------------------------------------------------------------------
> Total  : 7 seconds
--------------------------------------------------------------------------------
================================================================================
- Statistics
================================================================================
>> Generated 17 mutations Killed 13 (76%)
>> Mutations with no coverage 2. Test strength 87%
>> Ran 20 tests (1.18 tests per mutation)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

After the successful execution as shown in the screenshots above, we execute the following commands to generate reports from the tests:

mvn test-compile org.pitest:pitest-maven:report

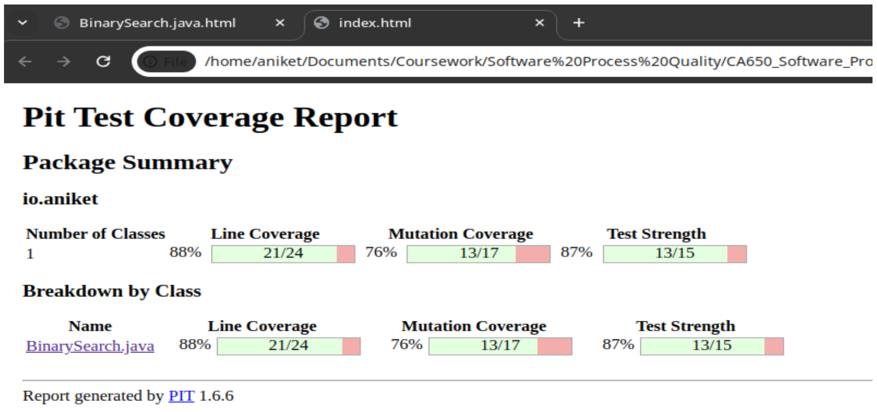


```
❯ mvn test-compile org.pitest:pitest-maven:report
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< io.aniket:TestToolEvaluation >--------------------
[INFO] Building TestToolEvaluation 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ TestToolEvaluation ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ TestToolEvaluation ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ TestToolEvaluation ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/aniket/Documents/Coursework/Software Process Quality/CA650_Software_Process_Quality/TestToolEvaluation/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ TestToolEvaluation ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- pitest-maven:1.6.6:report (default-cli) @ TestToolEvaluation ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.734 s
[INFO] Finished at: 2024-03-27T15:42:42Z
[INFO] ------------------------------------------------------------------------
   ~/Doc/Co/So/CA650_Software_Process_Quality/TestToolEvaluation │ master +15 !9 ?6
```

VII. Results

1. Coverage:

- PITest generates multiple types of reports and we'll be utilizing HTML reports.



BinarySearch.java.html | index.html

/home/aniket/Documents/Coursework/Software%20Process%20Quality/CA650_Software_Pro

Pit Test Coverage Report

Package Summary

io.aniket

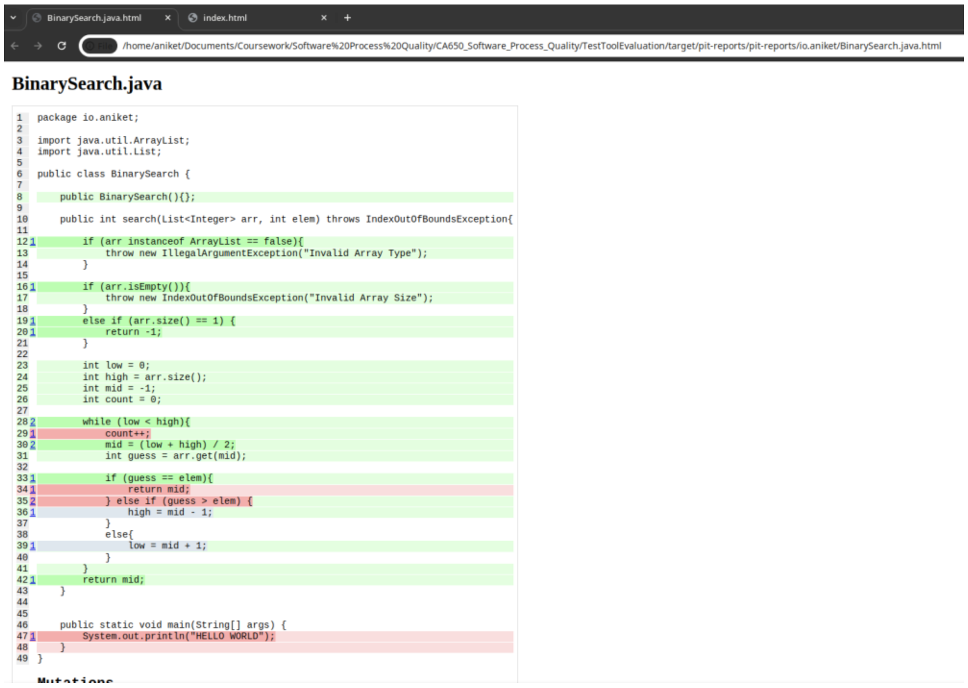
Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	88% 21/24	76% 13/17	87% 13/15

Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
BinarySearch.java	88% 21/24	76% 13/17	87% 13/15

Report generated by PIT 1.6.6

- The coverage report shows the Line and Mutation coverage of the test cases and shows the test strength. The classname is highlighted and on clicking the name, it shows a detailed report as follows:



BinarySearch.java.html | index.html

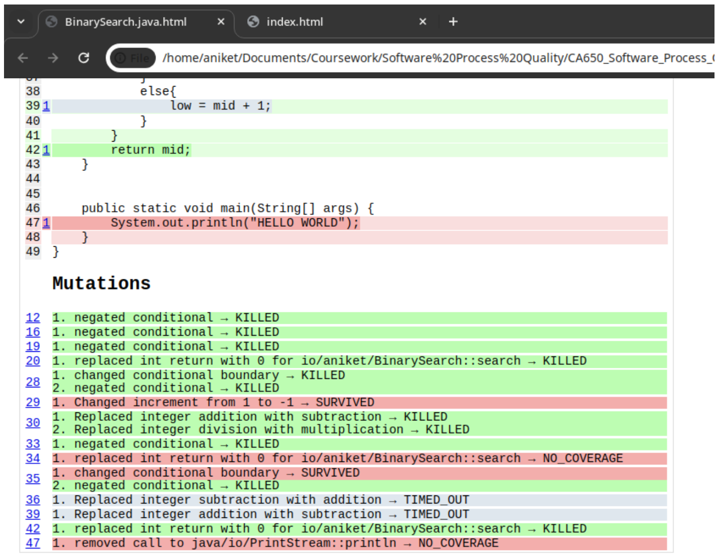
/home/aniket/Documents/Coursework/Software%20Process%20Quality/CA650_Software_Process_Quality/TestToolEvaluation/target/pit-reports/pit-reports/io.aniket/BinarySearch.java.html

BinarySearch.java

```java
package io.aniket;

import java.util.ArrayList;
import java.util.List;

public class BinarySearch {

    public BinarySearch(){};

    public int search(List<Integer> arr, int elem) throws IndexOutOfBoundsException{

        if (arr instanceof ArrayList == false){
            throw new IllegalArgumentException("Invalid Array Type");
        }

        if (arr.isEmpty()){
            throw new IndexOutOfBoundsException("Invalid Array Size");
        }
        else if (arr.size() == 1) {
            return -1;
        }

        int low = 0;
        int high = arr.size();
        int mid = -1;
        int count = 0;

        while (low < high){
            count++;
            mid = (low + high) / 2;
            int guess = arr.get(mid);

            if (guess == elem){
                return mid;
            } else if (guess > elem) {
                high = mid - 1;
            }
            else{
                low = mid + 1;
            }
        }
        return mid;
    }


    public static void main(String[] args) {
        System.out.println("HELLO WORLD");
    }
}
```

Mutations

- The screenshot above shows the Line coverage for the class BinarySearch.

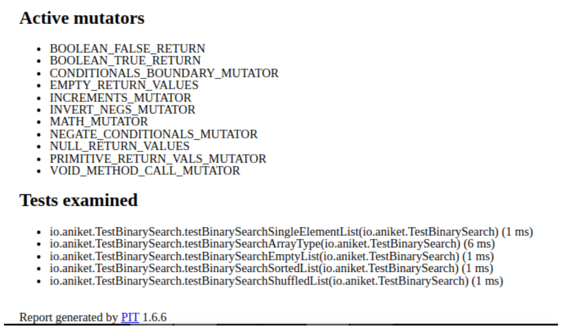

BinarySearch.java.html
index.html
/home/aniket/Documents/Coursework/Software%20Process%20Quality/CA650_Software_Process_

```
            else{
                low = mid + 1;
            }
        }
        return mid;
    }


    public static void main(String[] args) {
        System.out.println("HELLO WORLD");
    }
}
```

Mutations

```
12  1. negated conditional → KILLED
16  1. negated conditional → KILLED
19  1. negated conditional → KILLED
20  1. replaced int return with 0 for io/aniket/BinarySearch::search → KILLED
28  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
29  1. Changed increment from 1 to -1 → SURVIVED
30  1. Replaced integer addition with subtraction → KILLED
    2. Replaced integer division with multiplication → KILLED
33  1. negated conditional → KILLED
34  1. replaced int return with 0 for io/aniket/BinarySearch::search → NO_COVERAGE
35  1. changed conditional boundary → SURVIVED
    2. negated conditional → KILLED
36  1. Replaced integer subtraction with addition → TIMED_OUT
39  1. Replaced integer addition with subtraction → TIMED_OUT
42  1. replaced int return with 0 for io/aniket/BinarySearch::search → KILLED
47  1. removed call to java/io/PrintStream::println → NO_COVERAGE
```

- The screenshot above shows the number of mutations generated for different lines in the code. For instance, lines 12, 16, 19, and 20 have one mutation each whereas line 28 and 30 have 2 mutations for them. The mutations list shows red, white and green background for lines. The green lines are the mutations that were executed successfully, the red lines are failed mutations, and the white lines are mutations that were never executed.



Active mutators

- BOOLEAN_FALSE_RETURN
- BOOLEAN_TRUE_RETURN
- CONDITIONALS_BOUNDARY_MUTATOR
- EMPTY_RETURN_VALUES
- INCREMENTS_MUTATOR
- INVERT_NEGS_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- NULL_RETURN_VALUES
- PRIMITIVE_RETURN_VALS_MUTATOR
- VOID_METHOD_CALL_MUTATOR

Tests examined

- io.aniket.TestBinarySearch.testBinarySearchSingleElementList(io.aniket.TestBinarySearch) (1 ms)
- io.aniket.TestBinarySearch.testBinarySearchArrayType(io.aniket.TestBinarySearch) (6 ms)
- io.aniket.TestBinarySearch.testBinarySearchEmptyList(io.aniket.TestBinarySearch) (1 ms)
- io.aniket.TestBinarySearch.testBinarySearchSortedList(io.aniket.TestBinarySearch) (1 ms)
- io.aniket.TestBinarySearch.testBinarySearchShuffledList(io.aniket.TestBinarySearch) (1 ms)

Report generated by PIT 1.6.6

- This image shows the active mutators that the test utilized and the list of test cases that were examined in the test.

VIII. Conclusion

To conclude this demonstration and report, I would say i found PITest to be very efficient, fast and easier to use. I was able to get a good insight into the code and how there are cases that can still be covered for testing. Though this tool does not explicitly show each mutation unlike other mutation tools such as muJava, the tools efficacy and robustness help with writing good test cases. The tool also makes it a bit difficult to cover some parts of the code as it generates mutations for some unreachable lines of code. This behavior of the tool may sometimes hamper the overall coverage and percentage of the test cases. However, it does enforce a necessity to rethink your test cases so that they can be improved. Alongside this, the consistent support from its developers and their concentration on the development to keep the tool up to date is an added advantage over numerous other tools which are no longer supported / updated. The setup and usage of the tool is easy though even for such a small piece of code, the process is a bit longer than unit testing and other types of testing.