Software Testing Project

CSE 731

Title - Mutation Testing On Library Management System

Repository Link - Github Link

Instructor: Prof. Meenakshi D Souza

Team Members-

1. **Manasi Purkar (MT2023158)**
2. **Pracheti Bhale (MT2023155)**

What is mutation testing?

Mutation Testing is a white box software testing technique used to evaluate the quality of test cases. It works by intentionally introducing small changes, called **mutants**, into the source code to simulate potential bugs. The goal is to check if the existing test cases can detect these changes.

Purpose of Mutation Testing

- **Evaluate Test Case Effectiveness**: Ensures that test cases can detect subtle errors.
- **Identify Weak Test Cases**: Points out test cases that fail to detect certain code modifications.
- **Improve Code Quality**: Encourages the creation of robust and comprehensive test cases.

Need for Mutation Testing

- **Uncover Hidden Bugs**: Simulates real-world errors that might occur due to mistakes in the code.
- **Validate Test Suite Robustness**: Ensures that the test suite catches logical and structural issues.
- **Complementary Approach**: Works alongside other testing techniques to strengthen quality assurance.

Mutation Operators

Mutation operators are rules used to create mutants. They simulate common programming mistakes. Examples:

- **Arithmetic Operator Replacement**: Change `+` to `-`, `*` to `/`, etc.
- **Logical Operator Replacement**: Change `&&` to `||`, `!` to a direct boolean.
- **Relational Operator Replacement**: Modify `>` to `<`, `>=` to `<=`, etc.
- **Statement Deletion**: Remove a line or a block of code.
- **Variable Replacement**: Substitute one variable with another in the same scope.

Steps in mutation testing -

1. Generate Mutants

- **What It Means**:
  Mutants are slightly altered versions of the original program, created by applying **mutation operators**. These operators simulate common coding errors (e.g., replacing `>` with `<` or `+` with `-`).
- **Purpose**:
  To mimic potential bugs that developers might introduce, ensuring that the test suite can detect and handle them effectively.
- **Example**:
  - Original Code: `if (x > 10)`
  - Mutant: `if (x >= 10)`
- **Mutation Operators**:
  - **Arithmetic Operator Replacement**: Replace `+` with `-`, `/` with `*`, etc.
  - **Relational Operator Replacement**: Change `>` to `<`, `>=` to `<=`, etc.
  - **Logical Operator Replacement**: Replace `&&` with `||`, `!` with `true` or `false`.
  - **Variable Replacement**: Replace one variable with another in the same scope.

2. Run Tests

- **What It Means**:
  Execute the existing test suite against both the original program and all generated mutants. This step evaluates whether the test cases can detect the intentional changes introduced in the mutants.
- **Key Idea**:
  If a mutant leads to a test failure (different output than the original), it is considered **killed**. If it passes all the tests, it **survives**.
- **Example Execution**:
  - Test Case Input: `x = 10`
  - Original Code Output: `false`
  - Mutant Output: `true`
  - Result: The test case **kills** the mutant by detecting the difference.

3. Analyze Results

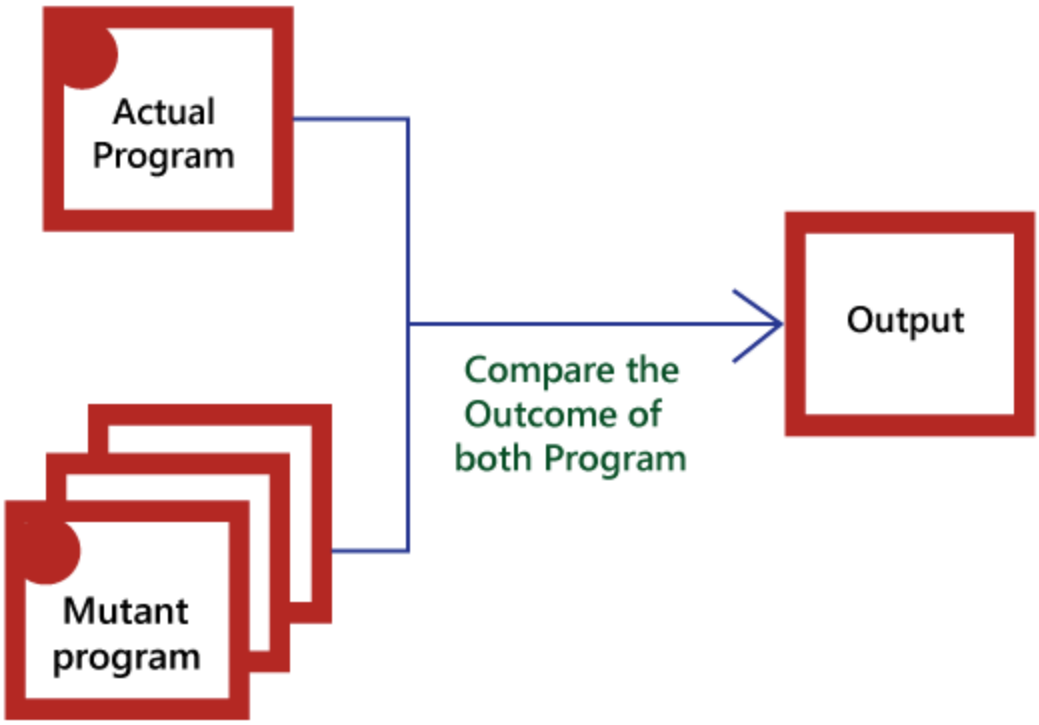
The test outcomes are analyzed to determine the effectiveness of the test suite in identifying errors.

Killed Mutants

- **Definition**: Mutants that are detected by the test cases.
- **Significance**:
    - Indicates that the test suite is robust enough to catch the introduced changes.
    - A high number of killed mutants shows good coverage and fault-detection capability.
- **Example**: If the mutant changes `>` to `<` and a test case detects the error by failing, this mutant is "killed."

Surviving Mutants

- **Definition**: Mutants that pass all the test cases undetected.
- **Significance**:
    - Highlights gaps or weaknesses in the test suite.
    - Indicates the need to create or improve test cases to cover the missed scenarios.
- **Example**: If no test case checks for the edge condition where `>=` behaves differently from `>`, the mutant survives undetected.



Source- https://www.javatpoint.com/mutation-testing

Outcome of Analysis

- **Mutation Score**:
  A metric to evaluate the quality of the test suite:

$$\text{Mutation Score} = \frac{\text{Number of Killed Mutants}}{\text{Total Number of Mutants}} \times 100$$

  A **high mutation score** indicates an effective test suite.

  - Surviving mutants suggest areas where test coverage needs improvement.

Unit-Level Mutation Testing

- **Focus**: Tests individual units (e.g., functions, methods).
- **Purpose**: Ensure correctness of isolated code segments and evaluate unit test effectiveness.
- **Scope**: Mutates only a specific unit.
- **Example**:
  Original: def add(a, b): return a + b
  Mutant: def add(a, b): return a - b

Integration-Level Mutation Testing

- **Focus**: Tests interactions between multiple units or components.
- **Purpose**: Ensure seamless integration and identify bugs in unit interactions.
- **Scope**: Mutates code related to data flow, API calls, or shared resources.
- **Example**:
  Original: Component A calls B(x);
  Mutant: Component A calls B(x+1).

Tools Used -

JUnit

- **Type**: Testing Framework for Java.
- **Purpose**: Used for unit testing Java applications.
- **Features**:
  - Provides annotations like `@Test`, `@Before`, and `@After`.
  - Supports assertion methods (e.g., `assertEquals`, `assertTrue`) to validate outcomes.
  - Helps automate testing and integrates well with build tools like Maven/Gradle.
- **Use Case**: Writing and running unit tests for individual classes or methods.

PIT (Pitest)

- **Type**: Mutation Testing Tool for Java.
- **Purpose**: Evaluates the effectiveness of test cases by generating and executing mutants.

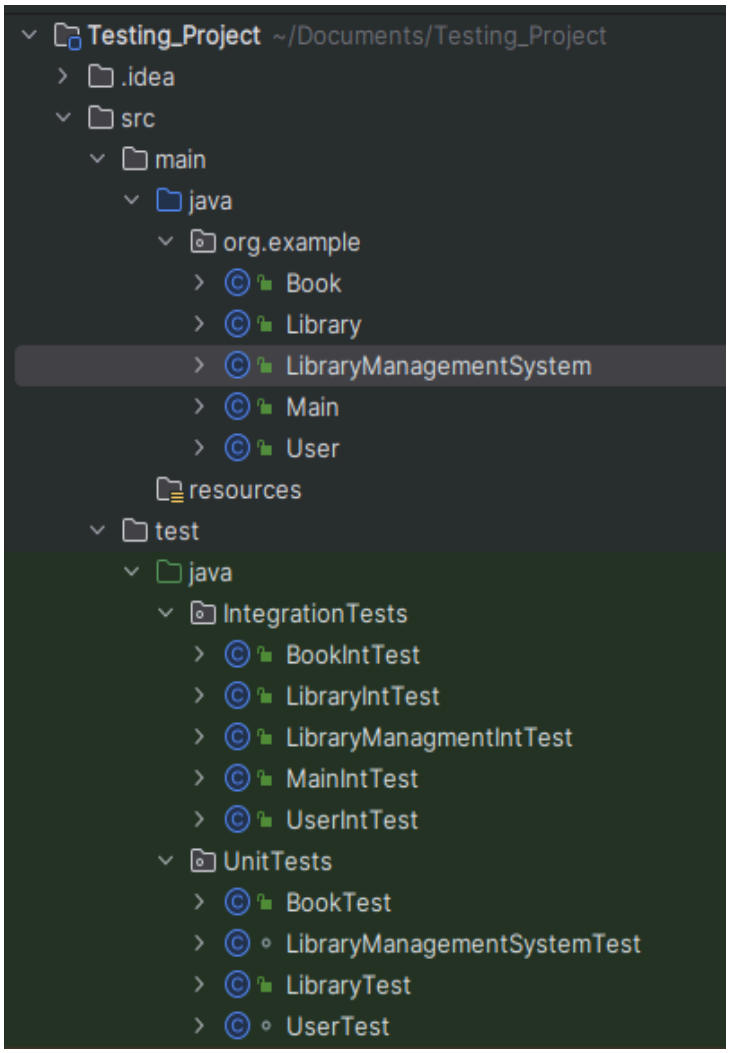
- **Features**:
    - Works with JUnit to assess test case quality.
    - Provides mutation operators for common programming errors.
    - Generates a **mutation score** to measure test suite effectiveness.
- **Use Case**: Ensuring the robustness of test cases by detecting missed bugs.

Problem Statement-

Library Management System

The Library Management System is a software application that enables efficient management of books and users in a library. It allows administrators to add new books and users, and manage book borrowing and returning processes. The system provides functionalities to find books by author, display all books and users, and track which users have borrowed a specific book. The system's user-friendly interface offers options for managing library operations, ensuring a seamless and organized experience for both administrators and users.

Project Structure -



Functions Involved -

addNewBook(): Adds a new book to the library with validation for non-empty title and author.

addNewUser(): Adds a new user to the system by entering user ID and name.

borrowBook(): Allows a user to borrow a book if the user and book are valid.

returnBook(): Allows a user to return a borrowed book, checking if the book exists.

displayAllBooksAndUsers(): Displays all books and users in the library.

findBooksByAuthor(): Finds and displays books by a specified author along with their availability.

findUsersWhoBorrowedBook(): Finds and lists users who have borrowed a specified book.

run(): Main loop that presents a menu to interact with the library system.

```
Library Management System
1. Add a new book
2. Add a new user
3. Borrow a book
4. Return a book
5. Display all books and users
6. Find books by author
7. Find users who borrowed a book
8. Exit
Choose an option:
```

```
Choose an option: 1
Enter Book ID: 1
Enter Book Title: Alchemist
Enter Book Author: Paulo Coelho
"Alchemist" has been added to the library.
```

```
Choose an option: 2
Enter User ID: 1
Enter User Name: ABC
ABC has been added as a user.
```

```
Choose an option: 5
Library Books:
Book ID: 1, Title: Alchemist, Author: Paulo Coelho, Available: Yes
Library Users:
ABC (User ID: 1)
```

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

   <modelVersion>4.0.0</modelVersion>

   <groupId>org.example</groupId>

   <artifactId>Testing_Project</artifactId>

   <version>1.0-SNAPSHOT</version>

   <properties>

       <maven.compiler.source>17</maven.compiler.source>

       <maven.compiler.target>17</maven.compiler.target>

       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

   </properties>


   <dependencies>

       <dependency>

           <groupId>org.junit.jupiter</groupId>

           <artifactId>junit-jupiter</artifactId>

           <version>5.10.2</version>

           <scope>test</scope>

       </dependency>


       <dependency>

           <groupId>org.pitest</groupId>
```

```
            <artifactId>pitest-junit5-plugin</artifactId>
            <version>1.0.0</version>
            <scope>test</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/junit/junit -->

        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
            <version>5.5.0</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-junit-jupiter</artifactId>
            <version>5.5.0</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-surefire-plugin</artifactId>
```

```
            <version>3.1.2</version>
            <configuration>
                <includes>
                    <include>**/UnitTests/*Test.java</include>
                    <include>**/IntegrationTests/*Test.java</include>
                </includes>
            </configuration>
        </plugin>

        <!-- Maven Compiler Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.11.0</version>
            <configuration>
                <source>${maven.compiler.source}</source>
                <target>${maven.compiler.target}</target>
            </configuration>
        </plugin>

        <!-- PIT Maven Plugin -->
        <plugin>
            <groupId>org.pitest</groupId>
            <artifactId>pitest-maven</artifactId>
            <version>1.9.11</version>
            <configuration>
```

```
                <!-- Specify which classes to mutate -->
                <targetClasses>
                    <param>org.example.*</param>
                </targetClasses>
                <!-- Specify which tests should run for mutation testing -->
                <targetTests>
                    <param>UnitTests.*Test</param>
                    <param>IntegrationTests.*Test</param>
                </targetTests>

                <!-- Use a stronger mutation set -->
                <mutators>
                    <mutator>STRONGER</mutator>
                </mutators>

                <!-- Enable HTML report generation -->
                <outputFormats>
                    <param>HTML</param>
                </outputFormats>
            </configuration>
        </plugin>
    </plugins>
  </build>
</project>
```

Test Cases -

Command -

mvn clean test

```
@Test
void testMain_AddNewBookOption() {
    String simulatedInput = "1\n101\nBook Title\nAuthor Name\n8\n"; // Add book and exit
    ByteArrayInputStream inputStream = new ByteArrayInputStream(simulatedInput.getBytes());
    System.setIn(inputStream);

    // Capture system output
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outputStream));

    // Run Main class
    Main.main(new String[]{});

    // Validate output
    String output = outputStream.toString();
    assertTrue(output.contains("Library Management System"), message: "Main menu should be displayed.");
    assertTrue(output.contains("1. Add a new book"), message: "Add book option should be displayed.");
    assertTrue(output.contains("Enter Book ID:"));
    assertTrue(output.contains("Enter Book Title:"));
    assertTrue(output.contains("Enter Book Author:"));
    assertTrue(output.contains("Exiting the system..."), message: "Exit message should be displayed.");
}
```

Integration level testing of adding book by running main method

```java
@Test
public void testAddBook() {

    Book book4 = new Book( bookId: 4,  title: "Brave New World",  author: "Aldous Huxley");
    Book book5 = new Book( bookId: 5,  title: "To Kill a Mockingbird",  author: "Harper Lee");

    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outputStream));
    library.addBook(book4);
    library.addBook(book5);

    List<Book> books = library.getBooks();
    assertEquals( expected: 5, books.size());
    assertTrue(books.contains(book4));
    assertTrue(books.contains(book5));
    String output = outputStream.toString();
    assertTrue(output.contains(" has been added to the library."));

}
```

Unit level test case of adding new book

```
[INFO]
[INFO] Tests run: 83, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  8.047 s
[INFO] Finished at: 2024-11-26T18:07:01+05:30
[INFO] ------------------------------------------------------------------------
```

Tests run successfully

PIT Report -

Pit Command-

mvn org.pitest:pitest-maven:mutationCoverage

Pit Test Coverage Report

Package Summary

org.example

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
5	99%	221/223	89%	117/131	90%	117/130

Breakdown by Class

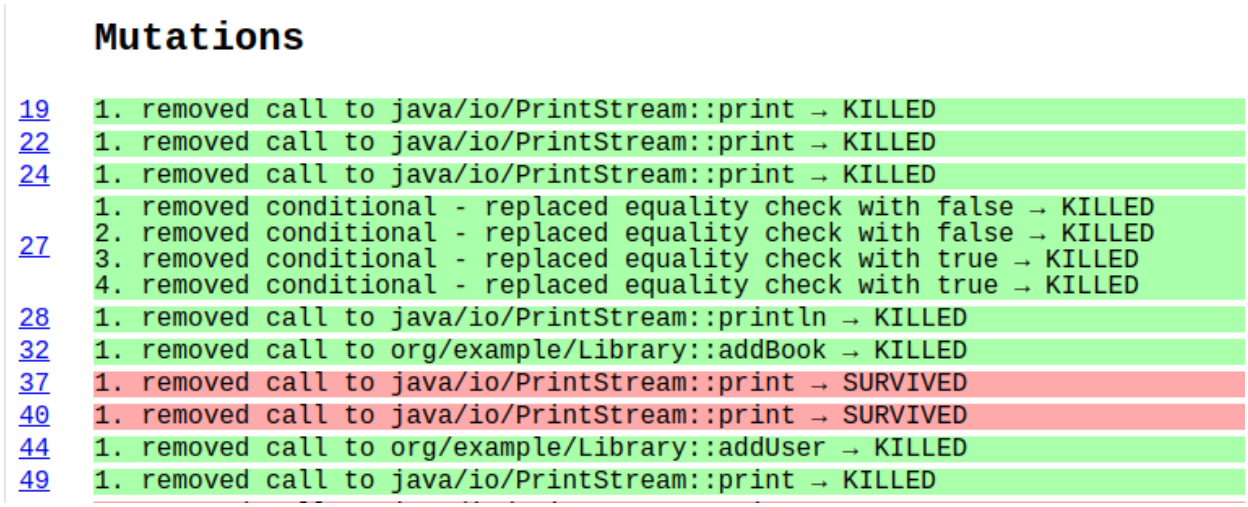
Name	Line Coverage		Mutation Coverage		Test Strength	
Book.java	100%	20/20	88%	7/8	88%	7/8
Library.java	100%	52/52	88%	22/25	88%	22/25
LibraryManagementSystem.java	99%	115/116	89%	68/76	91%	68/75
Main.java	75%	3/4	100%	1/1	100%	1/1
User.java	100%	31/31	90%	19/21	90%	19/21

Report generated by PIT 1.9.11

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- EXPERIMENTAL_SWITCH
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NULL_RETURNS
- PRIMITIVE_RETURNS
- REMOVE_CONDITIONALS_EQUAL_ELSE
- REMOVE_CONDITIONALS_EQUAL_IF
- REMOVE_CONDITIONALS_ORDER_ELSE
- REMOVE_CONDITIONALS_ORDER_IF
- TRUE_RETURNS
- VOID_METHOD_CALLS

LibraryManagementSystem.java



LibraryManagementSystem.java

Below is a brief description of each operator:

1. **CONDITIONALS_BOUNDARY**: Alters relational operators (e.g., `>` to `>=` or `<` to `<=`) to test edge cases.
2. **EMPTY_RETURNS**: Replaces return values with an empty or null return.
3. **EXPERIMENTAL_SWITCH**: Modifies switch statements to test handling of unexpected cases.
4. **FALSE_RETURNS**: Forces a method to always return `false`, regardless of logic.
5. **INCREMENTS**: Alters increment and decrement operators (e.g., `i++` becomes `i--`).
6. **INVERT_NEGS**: Negates numeric values (e.g., `-5` becomes `5`).
7. **MATH**: Changes mathematical operators (e.g., `+` to `-` or `*` to `/`).
8. **NULL_RETURNS**: Forces methods to return `null` instead of an actual value.
9. **PRIMITIVE_RETURNS**: Modifies methods to return primitive default values (e.g., `0`, `false`).
10. **REMOVE_CONDITIONALS_EQUAL_ELSE**: Removes the `else` branch in conditional statements where the condition evaluates to `true`.
11. **REMOVE_CONDITIONALS_EQUAL_IF**: Removes the `if` branch in conditional statements where the condition evaluates to `true`.

12. **REMOVE_CONDITIONALS_ORDER_ELSE**: Deletes the `else` branch while reordering nested conditionals.
13. **REMOVE_CONDITIONALS_ORDER_IF**: Deletes the `if` branch while reordering nested conditionals.
14. **TRUE_RETURNS**: Forces methods to always return `true`.
15. **VOID_METHOD_CALLS**: Removes method calls that return `void`, testing dependency on such calls.

Contribution -

1. Manasi Purkar (MT2023158) - Installation of pit, creating codebase of library management system, written unit level test cases for library and library management classes, written integration test cases for main class, library management class and book class

2. Pracheti Bhale (MT2023155) - Installation of pit, creating codebase of library management system, written unit test cases for classes Book and User, written integration test cases for user class and library class

References -

1. https://www.javatpoint.com/mutation-testing
2. https://www.youtube.com/watch?v=kHlysr-yPYs
3. https://medium.com/wearewaes/the-mutation-test-7fe94ee2e5e8
4. https://medium.com/@dmaioni/java-project-and-junit-test-3b62b28a98b