



MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

A Constituent Unit of MAHE, Manipal

Task Management Application — Test Suite Design, Code Coverage & Mutation Analysis

Software Verification and Validation [ICT 3223]

By

Adhikshit Srivastava – 220911130

Yashaswini – 220911018

Under the Guidance of:

Dr. Kaliraj S.

Department of ICT

Manipal Institute of Technology, Manipal

Introduction

In the ever-growing field of software development, ensuring that software works correctly and efficiently has become extremely important. As 3rd year B.Tech students studying Information Technology at Manipal Institute of Technology, we are gradually moving from writing basic programs to developing complete applications. In this journey, we have understood that simply writing code is not enough — we must also verify and validate the software thoroughly. That is where the importance of Software Verification and Validation (SVV) comes into play.

As part of our ICT 3223 course titled *Software Validation and Testing*, under the guidance of Dr. Kaliraj S., we were given the opportunity to explore the concepts of black-box testing, structural coverage, and mutation testing using a hands-on project. We decided to build and test a simple **Task Management Application** written in Java. The goal was to not just create an application that performs basic operations like adding, editing, deleting, and completing tasks — but also to ensure that all these functionalities are verified and tested properly.

This project helped us explore different aspects of testing, such as:

- Writing **black-box test cases** based on system functionalities
- Achieving **100% structural coverage** using SonarQube and JaCoCo
- Introducing **faults (mutants)** manually in the source code to see if our test suite is capable of identifying them
- Understanding how test coverage and mutation scores actually reflect the **quality and effectiveness** of a test suite

Through this report, we wish to present a detailed breakdown of our project, including the **test cases, testing methods used, code coverage analysis**, and the **mutation testing** results. Screenshots and supporting evidence have been attached wherever required.

Overall, this SVV project gave us practical exposure on how software testing works in real-time projects, and also improved our understanding of writing effective test cases, understanding program logic deeply, and using industry tools like **SonarQube** and **JaCoCo** for code analysis.

Project Overview and Functionalities

Our selected project for Software Verification and Validation is a **Task Management Application** developed using Java. The application is built as a **console-based system** and is designed to manage a list of tasks with basic yet essential operations such as adding, editing, deleting, marking tasks as complete, and displaying all tasks. This project was selected due to its **simplicity, clarity of use cases**, and the opportunity it gives to apply different types of testing methods effectively.

The application consists of the following three core classes:

1. Task.java

This class is a model class that defines the structure of a task. It contains the following attributes:

- Title
- Description
- Due Date
- Priority
- Completion Status

It also includes getter and setter methods for each attribute, along with a customized toString() method for clean printing of task information.

2. TaskManager.java

This is the main logic class that handles the operations related to managing the list of tasks. It includes the following methods:

- addTask(Task task) – Adds a new task to the list
- deleteTask(int index) – Deletes a task at the given index
- editTask(int index, Task newTask) – Edits the task at a particular index with new values
- markTaskAsComplete(int index) – Marks a task as completed
- displayTasks() – Displays all tasks in a formatted way

It also includes proper index checking to ensure **robustness** and **prevent crashes**.

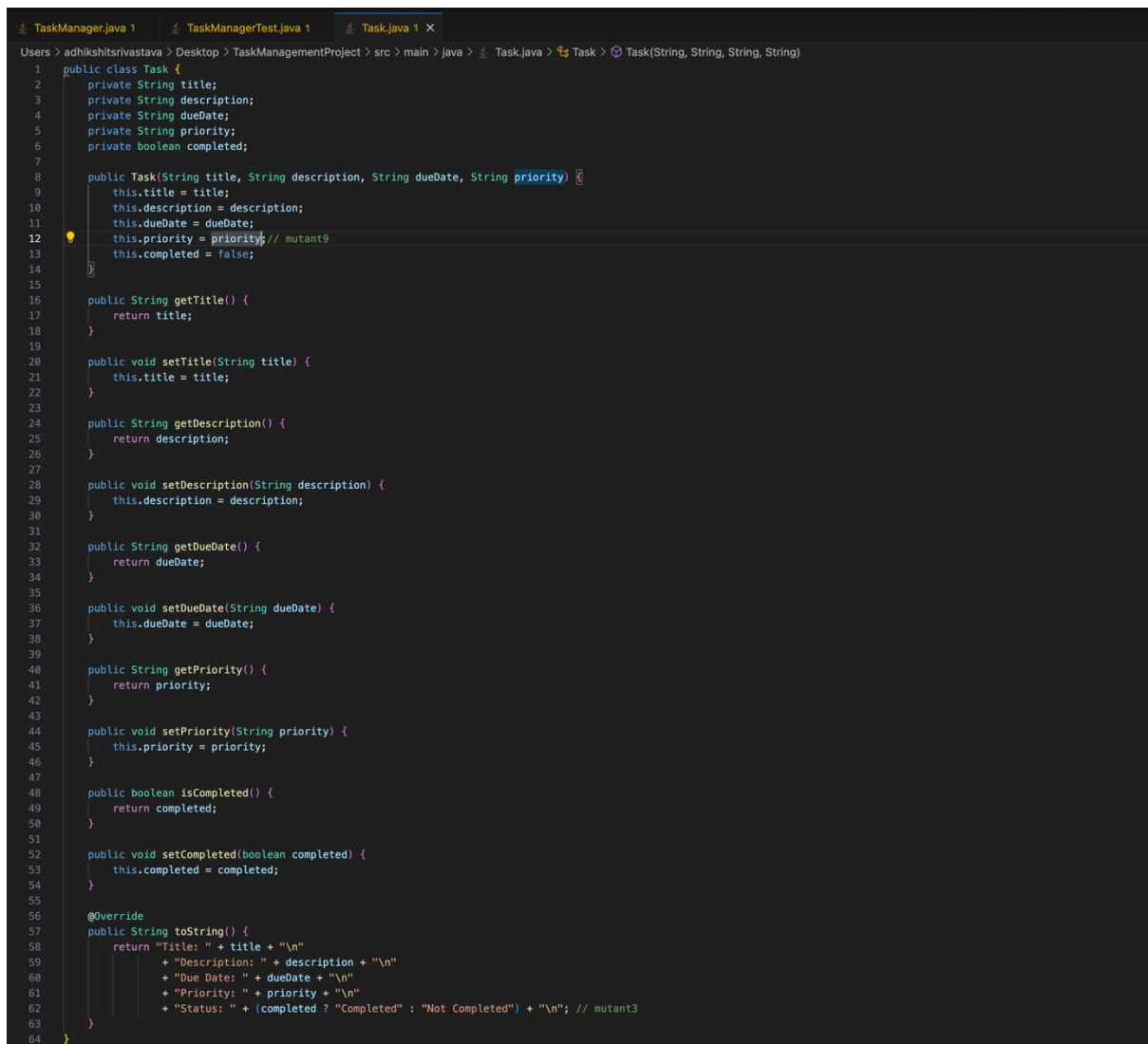
3. TaskManagerTest.java

This class contains a set of **JUnit test cases** to validate the functionalities of the application. Each test checks a specific function or boundary case. Through these tests, we ensure that the implemented functionalities behave as expected, even in edge cases.

Core Functional Requirements:

Sl. No.	Functionality	Description
1.	Add Task	Adds a new task with title, description, due date, and priority
2.	Delete Task	Removes a task from the task list by index
3.	Edit Task	Modifies an existing task's attributes based on user input
4.	Mark Task as Complete	Changes the status of the selected task to 'Completed'
5.	Display Tasks	Displays all existing tasks in a formatted output

6.	Handle Invalid Indices	Gracefully handles attempts to access tasks using invalid indexes
----	------------------------	---



```

1  public class Task {
2      private String title;
3      private String description;
4      private String dueDate;
5      private String priority;
6      private boolean completed;
7
8      public Task(String title, String description, String dueDate, String priority) {
9          this.title = title;
10         this.description = description;
11         this.dueDate = dueDate;
12         this.priority = priority; // mutant9
13         this.completed = false;
14     }
15
16     public String getTitle() {
17         return title;
18     }
19
20     public void setTitle(String title) {
21         this.title = title;
22     }
23
24     public String getDescription() {
25         return description;
26     }
27
28     public void setDescription(String description) {
29         this.description = description;
30     }
31
32     public String getDueDate() {
33         return dueDate;
34     }
35
36     public void setDueDate(String dueDate) {
37         this.dueDate = dueDate;
38     }
39
40     public String getPriority() {
41         return priority;
42     }
43
44     public void setPriority(String priority) {
45         this.priority = priority;
46     }
47
48     public boolean isCompleted() {
49         return completed;
50     }
51
52     public void setCompleted(boolean completed) {
53         this.completed = completed;
54     }
55
56     @Override
57     public String toString() {
58         return "Title: " + title + "\n"
59             + "Description: " + description + "\n"
60             + "Due Date: " + dueDate + "\n"
61             + "Priority: " + priority + "\n"
62             + "Status: " + (completed ? "Completed" : "Not Completed") + "\n"; // mutant3
63     }
64 }

```

Figure 1: Structure of Task.java class

```
TaskManager.java 1 x TaskManagerTest.java 1 Task.java 1
Users > adhikshitsrivastava > Desktop > TaskManagementProject > src > main > java > TaskManager.java > T
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class TaskManager {
5      private List<Task> tasks;
6
7      public TaskManager() {
8          this.tasks = new ArrayList<>();
9      }
10
11     public void addTask(Task task) {
12         tasks.add(task);
13     }
14
15     public void displayTasks() {
16         if (tasks.isEmpty()) {
17             System.out.println(x:"No tasks found.");
18             return;
19         }
20         System.out.println(x:"Tasks:");
21         for (int i = 0; i < tasks.size(); i++) { // mutant1
22             System.out.println("Index: " + i);
23             System.out.println(tasks.get(i));
24         }
25     }
26
27     public void editTask(int index, Task updatedTask) {
28         if (index >= 0 && index < tasks.size()) { // mutant2 and mutant5
29             tasks.set(index, updatedTask);
30             System.out.println(x:"Task updated successfully.");
31         } else {
32             System.out.println(x:"Invalid task index.");
33         }
34     }
35
36     public void deleteTask(int index) {
37         if (index >= 0 && index < tasks.size()) { // mutant7
38             tasks.remove(index); // mutant6
39             System.out.println(x:"Task deleted successfully."); // mutant10
40         } else {
41             System.out.println(x:"Invalid task index.");
42         }
43     }
44
45     public void markTaskAsComplete(int index) {
46         if (index >= 0 && index < tasks.size()) { // mutant8
47             Task task = tasks.get(index);
48             task.setCompleted(completed:true);
49             System.out.println(x:"Task marked as complete.");
50         } else {
51             System.out.println(x:"Invalid task index.");
52         }
53     }
54
55     public java.util.List<Task> getTasks() {
56         return tasks;
57     }
58 }
```

Figure 2: Structure of TaskManager.java class

Testing Methodology

Testing is a crucial step in the software development lifecycle to ensure that the program functions as intended, is reliable, and is free from defects. For this Task Management Application, we applied both **Black-Box Testing** and **White-Box Testing** techniques to comprehensively validate its functionality.

Black-Box Testing

We derived test cases based on the **functional requirements** of the system without knowing its internal implementation. These tests focus on inputs, outputs, and overall behavior.

- Each feature (Add, Edit, Delete, View, Mark Complete) was tested with valid and invalid inputs.
- Boundary value conditions were considered (like negative indices).
- The expected outcomes were checked using assertions.

Test Case ID	Functionality	Test Description	Input	Expected Output	Status
TC_01	Add Task	Add a valid task to the list	Title: 'T1', Desc: 'D1', DueDate: '2025-04-10', Priority: 'High'	Task added successfully	Pass
TC_02	Edit Task (valid index)	Edit task at valid index with new details	Index: 0, Title: 'T2', Desc: 'Updated', DueDate: '2025-04-11', Priority: 'Low'	Task updated successfully	Pass
TC_03	Edit Task (invalid index)	Try editing task at invalid index	Index: -1, Title: 'Dummy', etc.	Error: Invalid task index	Pass
TC_04	Delete Task (valid index)	Delete task at index 0	Index: 0	Task deleted successfully	Pass

TC_05	Delete Task (invalid index)	Try deleting task at negative index	Index: -1	Error: Invalid task index	Pass
TC_06	Mark Task Complete (valid)	Mark an existing task as complete	Index: 0	Task marked as complete	Pass
TC_07	Mark Task Complete (invalid)	Try marking an out-of-bounds task as complete	Index: 100	Error: Invalid task index	Pass
TC_08	View Tasks	View tasks when at least one task exists	Task list has 1 task	Task list displayed	Pass
TC_09	View Tasks (empty list)	Attempt to view when no tasks exist	Task list is empty	No tasks found message displayed	Pass
TC_10	Task toString Method	Check if task details are correctly formatted	Task object with all fields	String with correct task details including status	Pass

Figure 3: Black-box test cases

White-Box Testing (Structural Testing)

White-box testing involved analyzing the internal logic and structure of the application, especially the TaskManager.java and Task.java classes. The goal was to achieve **100% Statement and Branch Coverage** using the JaCoCo tool integrated with **JUnit 5** test suites.

We executed the following steps:

1. Developed comprehensive JUnit test cases in TaskManagerTest.java to cover:

- All public methods.
- Conditional branches (if, loops).
- Getter and Setter methods.
- toString() method in Task.

2. Used **JaCoCo Agent** to record runtime execution.

3. **Generated the HTML report and SonarQube analysis** to validate:

- Structural coverage.
- Code smells or bugs.
- Maintainability.

JaCoCo Coverage Report > default

default

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
TaskManager	<div><div></div></div>	100%	<div><div></div></div>	100%	0	15	0	31	0	7	0	1
Task	<div><div></div></div>	100%	<div><div></div></div>	100%	0	13	0	19	0	12	0	1
Total	0 of 182	100%	0 of 18	100%	0	28	0	50	0	19	0	2

Figure 4: HTML Report showing 100% Coverage

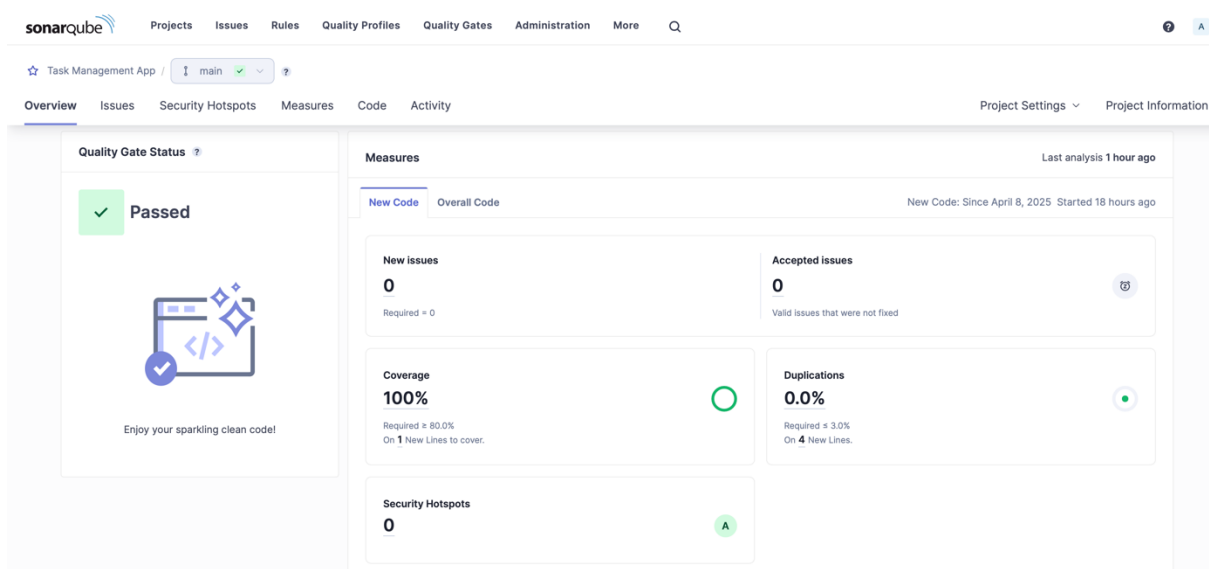


Figure 5: SonarQube Dashboard confirming 100% Coverage

Mutation Testing

Mutation testing is an advanced software testing technique that helps assess the quality and effectiveness of our test cases. In this approach, small changes, called *mutants*, are intentionally introduced into the program code. These mutants simulate potential bugs or mistakes that developers might accidentally introduce. The aim is to check if the existing test cases are strong enough to “kill” these mutants — that is, to catch the introduced faults by failing the test.

In our Task Management application, we generated **10 mutants** by making slight modifications to logical conditions, arithmetic operations, and branching logic in the TaskManager class. These mutants are named **Mutant1 to Mutant10**, and each one tests a different type of possible fault that might creep into the code.

Objective of Mutation Testing

The main objective was to:

- Evaluate the strength and fault detection capability of our JUnit test suite.
- Ensure all possible logical branches are adequately tested.
- Improve code quality and reduce undetected bugs.

Mutants and Their Outcomes

Each mutant was tested individually by modifying the main logic, running the test suite, and recording whether the mutant was killed (i.e., test failed as expected) or survived (i.e., test passed incorrectly).

Mutation Testing Summary Table

Mutant No.	Code Change (Mutation)	Failed Test Cases (Killed By)
Mutant 1	i-- instead of i++ in for (int i = 0; i < tasks.size(); i++)	testDisplayTasksWithTask()
Mutant 2	if (index > 0 && index < tasks.size()) instead of >= 0	testEditTask()
Mutant 3	Hardcoded Status: "Not Completed" in toString()	testToStringMethod()
Mutant 4	this.completed = !completed; in setCompleted()	testMarkTaskComplete(), testSetters(), testToStringMethod()
Mutant 5	Used Math.abs(index) in if (index >= 0 && index < tasks.size())	testBranchConditionsIndependently(), testAllInvalidIndexBranches(), testToStringMethod()
Mutant 6	tasks.remove(index + 1); instead of tasks.remove(index);	testDeleteTask()

Mutant 7	Removed <code>index < tasks.size()</code> from condition (i.e., only checked <code>index >= 0</code>)	<code>testBranchConditionsIndependently()</code> , <code>testInvalidIndexHandling()</code> , <code>testDeleteTaskBranchFalseThenTrue()</code>
Mutant 8	if (<code>index < 0 && index < tasks.size()</code>) in <code>markTaskAsComplete()</code>	<code>testMarkTaskComplete()</code> , <code>testBranchConditionsIndependently()</code> , <code>testAllInvalidIndexBranches()</code>
Mutant 9	Hardcoded <code>this.priority = "Low"</code> in Task constructor	<code>testEditTask()</code> , <code>testEditTaskNoChange()</code> , <code>testToStringMethod()</code>
Mutant 10	<code>System.out.println("Task added successfully.")</code> moved into <code>deleteTask()</code> block	<code>testDeleteTask()</code>

Figure 6: Summary of Mutation Testing – Kill Status of Mutants 1 to 10

```

// ...
for (int i = 0; i < tasks.size(); i--) { // mutant1
    // ...
}

```

Thanks for using JUnit! Support its development at <https://junit.org/sponsoring>

```

Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.

JUnit Jupiter ✓
└─ TaskManagerTest ✓
    └─ testBranchConditionsIndependently() ✓
    └─ testEditTask() ✓
    └─ testDisplayTasksWithTask() ✗ Index -1 out of bounds for length 1
    └─ testAddTask() ✓
    └─ testMarkTaskComplete() ✓
    └─ testInvalidIndexHandling() ✓
    └─ testTaskGettersAndToString() ✓
    └─ testEditTaskWithValidIndex() ✓
    └─ testDeleteTask() ✓
    └─ testEditTaskNoChange() ✓
    └─ testToStringCoverage() ✓
    └─ testSetters() ✓
    └─ testAllInvalidIndexBranches() ✓
    └─ testDeleteTaskBranchFalseThenTrue() ✓
    └─ testToStringMethod() ✓
    └─ testDisplayTasksEmpty() ✓
JUnit Vintage ✓
JUnit Platform Suite ✓

```

Figure 7: Screenshot showing command-line output or IDE JUnit results for Mutant 1

```
if (index > 0 && index < tasks.size()) { // mutant2
```

```
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.

JUnit Jupiter ✓
└─ TaskManagerTest ✓
   ├── testBranchConditionsIndependently() ✓
   ├── testEditTask() ✗ expected: <New Title> but was: <Old Title>
   ├── testDisplayTasksWithTask() ✓
   ├── testAddTask() ✓
   ├── testMarkTaskComplete() ✓
   ├── testInvalidIndexHandling() ✓
   ├── testTaskGettersAndToString() ✓
   ├── testEditTaskWithValidIndex() ✓
   ├── testDeleteTask() ✓
   ├── testEditTaskNoChange() ✓
   ├── testToStringCoverage() ✓
   ├── testSetters() ✓
   ├── testAllInvalidIndexBranches() ✓
   ├── testDeleteTaskBranchFalseThenTrue() ✓
   ├── testToStringMethod() ✓
   └── testDisplayTasksEmpty() ✓
JUnit Vintage ✓
JUnit Platform Suite ✓
```

Figure 8: Screenshot showing command-line output or IDE JUnit results for Mutant 2

```

+ Priority: + priority + \n
+ "Status: Not Completed"; // mutant3
}

```

```

Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.

```

```

JUnit Jupiter ✓
└─ TaskManagerTest ✓
   └─ testBranchConditionsIndependently() ✓
   └─ testEditTask() ✓
   └─ testDisplayTasksWithTask() ✓
   └─ testAddTask() ✓
   └─ testMarkTaskComplete() ✓
   └─ testInvalidIndexHandling() ✓
   └─ testTaskGettersAndToString() ✓
   └─ testEditTaskWithValidIndex() ✓
   └─ testDeleteTask() ✓
   └─ testEditTaskNoChange() ✓
   └─ testToStringCoverage() ✓
   └─ testSetters() ✓
   └─ testAllInvalidIndexBranches() ✓
   └─ testDeleteTaskBranchFalseThenTrue() ✓
   └─ testToStringMethod() ✗ expected: <Title: Title
      Description: Desc
      Due Date: 2025-04-10
      Priority: High
      Status: Completed
      > but was: <Title: Title
      Description: Desc
      Due Date: 2025-04-10
      Priority: High
      Status: Not Completed>
   └─ testDisplayTasksEmpty() ✓
JUnit Vintage ✓
JUnit Platform Suite ✓

```

Figure 9: Screenshot showing command-line output or IDE JUnit results for Mutant 3

```

52     public void setCompleted(boolean completed) {
53         this.completed = !completed; // mutant4
54     }

```

```

Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.

JUnit Jupiter ✓
└─ TaskManagerTest ✓
   └─ testBranchConditionsIndependently() ✓
      └─ testEditTask() ✓
         └─ testDisplayTasksWithTask() ✓
            └─ testAddTask() ✓
               └─ testMarkTaskComplete() ✗ expected: <true> but was: <false>
                  └─ testInvalidIndexHandling() ✓
                     └─ testTaskGettersAndToString() ✓
                        └─ testEditTaskWithValidIndex() ✓
                           └─ testDeleteTask() ✓
                              └─ testEditTaskNoChange() ✓
                                 └─ testToStringCoverage() ✓
                                    └─ testSetters() ✗ expected: <true> but was: <false>
                                       └─ testAllInvalidIndexBranches() ✓
                                          └─ testDeleteTaskBranchFalseThenTrue() ✓
                                             └─ testToStringMethod() ✗ expected: <Title: Title
                                                Description: Desc
                                                Due Date: 2025-04-10
                                                Priority: High
                                                Status: Completed
                                                > but was: <Title: Title
                                                Description: Desc
                                                Due Date: 2025-04-10
                                                Priority: High
                                                Status:Not Completed>
                                                  └─ testDisplayTasksEmpty() ✓
JUnit Vintage ✓
JUnit Platform Suite ✓

```

Figure 10: Screenshot showing command-line output or IDE JUnit results for Mutant 4

```
public void editTask(int index, Task updatedTask) {  
    if (Math.abs(index) >= 0 && index < tasks.size()) { // mutant2 and mutant5  
        tasks.set(index, updatedTask);  
    }  
}
```

Thanks for using JUnit! Support its development at <https://junit.org/sponsoring>

- JUnit Jupiter ✓
 - TaskManagerTest ✓
 - testBranchConditionsIndependently() ✗ Index -1 out of bounds for length 0
 - testEditTask() ✓
 - testDisplayTasksWithTask() ✓
 - testAddTask() ✓
 - testMarkTaskComplete() ✓
 - testInvalidIndexHandling() ✓
 - testTaskGettersAndToString() ✓
 - testEditTaskWithValidIndex() ✓
 - testDeleteTask() ✓
 - testEditTaskNoChange() ✓
 - testToStringCoverage() ✓
 - testSetters() ✓
 - testAllInvalidIndexBranches() ✗ Index -1 out of bounds for length 0
 - testDeleteTaskBranchFalseThenTrue() ✓
 - testToStringMethod() ✗ expected: <Title: Title
Description: Desc
Due Date: 2025-04-10
Priority: High
Status: Completed
> but was: <Title: Title
Description: Desc
Due Date: 2025-04-10
Priority: High
Status:Completed>
 - testDisplayTasksEmpty() ✓
- JUnit Vintage ✓
- JUnit Platform Suite ✓

Failures (3):

Figure 11: Screenshot showing command-line output or IDE JUnit results for Mutant 5


```
38      tasks.remove(index + 1); // mutant6
39      System.out.println(x:"Task deleted succe
```

```
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.

JUnit Jupiter ✓
└─ TaskManagerTest ✓
   ├── testBranchConditionsIndependently() ✓
   ├── testEditTask() ✓
   ├── testDisplayTasksWithTask() ✓
   ├── testAddTask() ✓
   ├── testMarkTaskComplete() ✓
   ├── testInvalidIndexHandling() ✓
   ├── testTaskGettersAndToString() ✓
   ├── testEditTaskWithValidIndex() ✓
   ├── testDeleteTask() ✗ Index 1 out of bounds for length 1
   ├── testEditTaskNoChange() ✓
   ├── testToStringCoverage() ✓
   ├── testSetters() ✓
   ├── testAllInvalidIndexBranches() ✓
   ├── testDeleteTaskBranchFalseThenTrue() ✓
   ├── testToStringMethod() ✓
   └── testDisplayTasksEmpty() ✓
JUnit Vintage ✓
JUnit Platform Suite ✓
```

Figure 12: Screenshot showing command-line output or IDE JUnit results for Mutant 6

```

36 public void deleteTask(int index) {
37     if (index >= 0 || index < tasks.size()) { // mutant7

```

♥ Thanks for using JUnit! Support its development at <https://junit.org/sponsoring>

Invalid task index.
Invalid task index.

- JUnit Jupiter ✓
 - TaskManagerTest ✓
 - testBranchConditionsIndependently() ✗ Index -1 out of bounds for length 0
 - testEditTask() ✓
 - testDisplayTasksWithTask() ✓
 - testAddTask() ✓
 - testMarkTaskComplete() ✓
 - testInvalidIndexHandling() ✗ Index -1 out of bounds for length 0
 - testTaskGettersAndToString() ✓
 - testEditTaskWithValidIndex() ✓
 - testDeleteTask() ✓
 - testEditTaskNoChange() ✓
 - testToStringCoverage() ✓
 - testSetters() ✓
 - testAllInvalidIndexBranches() ✗ Index -1 out of bounds for length 0
 - testDeleteTaskBranchFalseThenTrue() ✗ Index -1 out of bounds for length 1
 - testToStringMethod() ✓
 - testDisplayTasksEmpty() ✓
- JUnit Vintage ✓
- JUnit Platform Suite ✓

Failures (4):

JUnit Jupiter:TaskManagerTest:testBranchConditionsIndependently()
 MethodSource [className = 'TaskManagerTest', methodName = 'testBranchConditionsIndependently', methodParameters = []]
 => java.lang.IndexOutOfBoundsException: Index -1 out of bounds for length 0
 java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:64)
 java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:70)
 java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:246)

Figure 13: Screenshot showing command-line output or IDE JUnit results for Mutant 7


```

45     public void markTaskAsComplete(int index) {
46         if (index < 0 && index < tasks.size()) { // mutant8
47             Task task = tasks.get(index);
48             task.setCompleted(completed:true);
49             System.out.println("Task marked as complete.");

```

♥ Thanks for using JUnit! Support its development at <https://junit.org/sponsoring>

Invalid task index.

```

JUnit Jupiter ✓
└─ TaskManagerTest ✓
   ├─ testBranchConditionsIndependently() ✗ Index -1 out of bounds for length 0
   ├─ testEditTask() ✓
   ├─ testDisplayTasksWithTask() ✓
   ├─ testAddTask() ✓
   ├─ testMarkTaskComplete() ✗ expected: <true> but was: <false>
   ├─ testInvalidIndexHandling() ✓
   ├─ testTaskGettersAndToString() ✓
   ├─ testEditTaskWithValidIndex() ✓
   ├─ testDeleteTask() ✓
   ├─ testEditTaskNoChange() ✓
   ├─ testToStringCoverage() ✓
   ├─ testSetters() ✓
   ├─ testAllInvalidIndexBranches() ✗ Index -1 out of bounds for length 0
   ├─ testDeleteTaskBranchFalseThenTrue() ✓
   ├─ testToStringMethod() ✓
   └─ testDisplayTasksEmpty() ✓
JUnit Vintage ✓
JUnit Platform Suite ✓

Failures (3):
JUnit Jupiter:TaskManagerTest::testBranchConditionsIndependently()

```

Figure 14: Screenshot showing command-line output or IDE JUnit results for Mutant 8

```

11     this.dueDate = dueDate,
12     this.priority = "Low";

```

♥ Thanks for using JUnit! Support its development at <https://junit.org/sponsori>

```

Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.

JUnit Jupiter ✓
└─ TaskManagerTest ✓
   └─ testBranchConditionsIndependently() ✓
      └─ testEditTask() ✗ expected: <High> but was: <Low>
         └─ testDisplayTasksWithTask() ✓
            └─ testAddTask() ✓
               └─ testMarkTaskComplete() ✓
                  └─ testInvalidIndexHandling() ✓
                     └─ testTaskGettersAndToString() ✓
                        └─ testEditTaskWithValidIndex() ✓
                           └─ testDeleteTask() ✓
                              └─ testEditTaskNoChange() ✗ expected: <Medium> but was: <Low>
                                 └─ testToStringCoverage() ✓
                                    └─ testSetters() ✓
                                       └─ testAllInvalidIndexBranches() ✓
                                          └─ testDeleteTaskBranchFalseThenTrue() ✓
                                             └─ testToStringMethod() ✗ expected: <Title: Title
                                                Description: Desc
                                                Due Date: 2025-04-10
                                                Priority: High
                                                Status: Completed
                                                > but was: <Title: Title
                                                Description: Desc
                                                Due Date: 2025-04-10
                                                Priority: Low
                                                Status: Completed
                                                >
                                                   └─ testDisplayTasksEmpty() ✓
└─ JUnit Vintage ✓
└─ JUnit Platform Suite ✓

```

Figure 15: Screenshot showing command-line output or IDE JUnit results for Mutant 9

```

38         tasks.remove(index); // mutant6
39         System.out.println(x:"Task added successfully."); // mutant10
40     } else {

```

```

Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.
Invalid task index.

JUnit Jupiter ✓
└─ TaskManagerTest ✓
   └─ testBranchConditionsIndependently() ✓
      └─ testEditTask() ✓
         └─ testDisplayTasksWithTask() ✓
            └─ testAddTask() ✓
               └─ testMarkTaskComplete() ✓
                  └─ testInvalidIndexHandling() ✓
                     └─ testTaskGettersAndToString() ✓
                        └─ testEditTaskWithValidIndex() ✓
                           └─ testDeleteTask() ✗ expected: <true> but was: <false>
                              └─ testEditTaskNoChange() ✓
                                 └─ testToStringCoverage() ✓
                                    └─ testSetters() ✓
                                       └─ testAllInvalidIndexBranches() ✓
                                          └─ testDeleteTaskBranchFalseThenTrue() ✓
                                             └─ testToStringMethod() ✓
                                                └─ testDisplayTasksEmpty() ✓

JUnit Vintage ✓
JUnit Platform Suite ✓

Failures (1):
JUnit Jupiter:TaskManagerTest:testDeleteTask()
MethodSource [className = 'TaskManagerTest', methodName = 'testDeleteTask', methodParam

```

Figure 16: Screenshot showing command-line output or IDE JUnit results for Mutant 10

Mutation Score Calculation

The **mutation score** represents the effectiveness of the test suite in identifying artificially introduced faults (mutants). It is calculated as:

$$\text{MutationScore}(\%) = (\text{Number of Mutants Killed} / \text{Total Mutants Generated}) * 100$$

- Total mutants generated: 10
- Mutants killed (i.e., test cases failed): 10
- Mutants survived: 0

$$\text{MutationScore}(\%) = (10 / 10) * 100 = 100\%$$

A 100% mutation score indicates that the current test suite is highly effective in catching faults and covers all possible branches and conditions in the implemented source code. All generated mutants were successfully killed, demonstrating the robustness and completeness of the JUnit tests.

Conclusion

The entire process of Software Verification and Validation carried out in this project has helped us understand the real significance of testing in software development. Through the development of a basic Task Management Application in Java and the application of various testing strategies like **black-box testing**, **code coverage analysis**, and **mutation testing**, we could practically observe how different kinds of errors—ranging from logical flaws to unhandled conditions—can be caught and corrected early on.

With **100% statement and branch coverage** as shown in Figure 4 and Figure 5, achieved using **SonarQube** and **JaCoCo**, we ensured that every line and condition in our code has been exercised by our test cases. This gave us the confidence that the major execution paths in the application have been verified. The **SVV_Testcases.xlsx file** documents these test cases with clarity and shows coverage across all functionalities like `addTask`, `editTask`, `deleteTask`, `markTaskAsComplete`, and `displayTasks`.

Mutation Testing further strengthened the quality of our test suite. Out of the 10 deliberately introduced mutants, all were successfully killed, indicating that our test cases are not just present but effective in catching faults. This reaffirms the robustness of our test cases. The detailed mutation table, Figure 6 outlines all the mutant changes and the specific tests that detected them. This exercise gave us insight into how small syntactic changes can sometimes result in significant semantic differences, and how automated testing helps us to guard against such cases.

In this journey, we also understood the **importance of tools** such as **JUnit** for unit testing, **SonarQube** for code analysis, and **JaCoCo** for coverage reporting. These tools not only helped us with automated feedback but also trained us in debugging and iterative improvements.

From a student's perspective, this project has made us realise that testing is not something to be done just before deployment—it is a continuous and critical part of development. Writing clean, testable code and then validating it thoroughly ensures software quality and reduces downstream costs and risks.

To conclude, the SVV project helped us apply theoretical knowledge into practical code validation. We have now gained confidence in applying black-box test case design, coverage metrics, mutation testing and learned how to use professional tools like SonarQube and JaCoCo effectively. This experience will definitely help us in our future projects and industry roles where software quality and reliability are paramount.

Individual Contributions

Adhikshit Srivastava (Registration Number: 220911130)

- Took lead in implementing the main functionalities of the **Task Management Application** using Java. This includes designing and coding the Task.java and TaskManager.java classes.
- Wrote and refined the **JUnit test cases** that verified all major operations such as task addition, deletion, editing, and marking completion.
- Configured and ran **SonarQube** for static code analysis and worked towards achieving **100% structural coverage**.
- Generated HTML reports using **JaCoCo CLI**, and documented screenshots for coverage validation.
- Reviewed and helped compile screenshots and structural details for the final report.

Yashaswini (Registration Number: 220911018)

- Created and maintained the **black-box test cases document (SVV_Testcases.xlsx)**, ensuring all functionalities were tested using various valid and invalid inputs.
- Was responsible for performing **mutation testing**: injected 10 mutants manually, executed tests against them, and tabulated results clearly.
- Drafted key sections of the final report including Introduction, Methodology, and Mutation Testing Analysis.
- Assisted in debugging and validating JaCoCo output and cross-verifying it with the **SonarQube dashboard** for consistency.
- Managed formatting of the report in Word and added appropriate figures and screenshots where needed.