Mekelle University

# Eitm

**School of computing**

Dept – Software Engineering

Year - 4th year

Course Title – Software Tool And Practices

Assignement – Writing a Brief Explanation of what TDD Is.

Group members

Name ID

1. Dagem Getu Molla Eitm/ur130323/10

2. Danay Hayelom Eitm/ur158026/11

3. Fikre Tesfay Eitm/ur157711/11

4. Measho Abrha Eitm/ur157966/11

5. Mubarek Mohammed Eitm/ur162363/11

Test Driven Development (TDD)

Test Driven Development (TDD) is a software development methodology where tests are written before the actual code. The TDD cycle consists of three main steps: Red, Green, Refactor.

**1. Red:** Write a test for the next bit of functionality you want to add. Initially, this test will fail because the functionality is not yet implemented.

**2. Green:** Write the minimal amount of code necessary to make the test pass.

**3. Refactor:** Clean up the code, ensuring that it still passes the tests. This step involves removing any duplication and improving the structure of the code without changing its behavior.

TDD encourages simple designs and inspires confidence. By writing tests first, developers can clarify requirements, make the code more robust, and reduce the number of bugs.

**The Good and Bad Sides of Test-Driven Development (TDD)**

The good side of TDD is that it encourages a disciplined and iterative approach, leading to more robust and maintainable code. It promotes a modular and testable codebase, making it easier to refactor and extend the application. The immediate feedback from the test suite helps catch bugs early. However, the bad side is that it can be time-consuming, especially for new developers, and may be challenging to adopt in organizations not used to this approach, leading to resistance and potential conflicts.

**TDD Demonstration**

Note that although we had started by defining the methods before writing the test cases initially, as we started to understand how to go about the Test Driven Development we started to implement the process in the correct way (by writing test case for the next bit of functionality which will fail at first but then we added the functionality to make the already written test cases pass. And then this is done in an iterative way. This can be seen from the last parts of our git history.)

**First Step: Writing the Tests (Red Phase)**

We started by creating a simple Java program called `Calculator.java`. Before implementing the methods, we defined our method declarations and wrote JUnit test cases for these methods. Here is an example of a few method declarations and their corresponding tests:

public class Calculator {

public int add(int a, int b) {

return 0; // placeholder

}

public int subtract(int a, int b) {

return 0; // placeholder

}

}

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertThrows;

import org.junit.Assert;

import org.junit.Test;

public class CalculatorTest {

Calculator calculator = new Calculator();

@Test

public void testAdd() {

assertEquals(5, calculator.add(2, 3));

assertEquals(0, calculator.add(2, -2));

}

@Test

public void testSubtract() {

assertEquals(1, calculator.subtract(3, 2));

assertEquals(-1, calculator.subtract(2, 3));

}

}

**Second Step: Implementing the Methods (Green Phase)**

Next, we implemented the methods in `Calculator.java` to ensure that the tests pass.

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

**Step 3: Refactoring (Refactor Phase)**

After implementing the methods, we refactored the code (mainly, we focused on correcting the naming of the test cases) to improve its readability while ensuring that all tests continued to pass.

**Git and GitHub Collaboration**

When working on a project with a team on GitHub, TDD becomes a crucial practice to ensure the codebase's integrity and maintainability. By writing tests before the actual code, developers can create a shared understanding of the project's requirements and ensure that new features or changes do not break existing functionality. This approach helps to catch issues early in the development cycle, making it easier to collaborate and integrate changes from different team members. GitHub's version control system, with its ability to track changes, merge branches, and manage conflicts, works seamlessly with the TDD workflow. Developers can create feature branches, write tests, and submit pull requests for review, allowing the team to collectively ensure the quality of the codebase. The combination of TDD and GitHub collaboration promotes a culture of shared responsibility, where team members work together to write high-quality, well-tested code that can be easily integrated and deployed.

To demonstrate Git and GitHub in a group work project, we initiated a collaborative project using GitHub.

**1. Repository Initialization:** We initialized a GitHub repository and set it up for our project.

**2. Forking and Cloning:** Each group member forked the repository to their own GitHub accounts and cloned it locally.

**3. Branching:** We created branches for new methods or test cases. But we didn’t use many.

**4. Development and Collaboration:**

**- Developing Features:** Each member developed features and wrote corresponding test cases in their branches.

**- Pull Requests:** Once a feature was complete, the member created a pull request (PR) from their branch to the main repository. The PR included the code changes and the test cases.

**- Code Reviews:** The main repository owner (Fikre Tesfay) reviewed the code changes.

**5. Merging:** After successful reviews, we merged the branches into the main branch of the repository.

**By using this open-source collaboration model, we simulated a real-world software development environment. Each group member's contributions were tracked, and the complete Git history provided a clear record of our collaborative efforts.**

**Summary**

In this assignment, we demonstrated the principles of Test Driven Development (TDD) by initially writing tests and then developing a `Calculator` program to pass these tests. We also effectively used Git and GitHub for collaborative development, employing branching, pull requests and code reviews to manage and integrate our work. This approach not only ensured the quality and reliability of our code but also fostered teamwork and collaboration among group members.