

Agile Software Engineering

Unit -4

Case Study :- Implementing Test-Driven Development (TDD) for a Healthcare Application

Student Name :	Pranav Vedula
Student RollNo:	24B11CS355
Department :	Computer Science Engineering
Section :	CSE- 9
Faculty Name :	K. RAJENDRA MTech,(Ph.D.) Assistant Professor Computer Science and Engineering

1. Introduction

Healthcare software systems require precision, reliability, and compliance with strict regulations. In this case study, a healthcare startup aimed to build a **Patient Health Monitoring Application** that tracks vital signs, sends emergency alerts, and integrates with hospital databases. To ensure high code quality and minimize bugs in production, the development team decided to adopt **Test-Driven Development (TDD)** as their primary software engineering practice.

2. Problem Statement

Traditional development approaches often resulted in:

- Late detection of defects during the integration stage.
- Inconsistent performance across different patient modules.
- Poor maintainability and difficulty in scaling.
- Regulatory compliance risks due to unverified features.

The team needed a **structured and test-first** approach that would enable **early bug detection, better collaboration**, and a **stable, secure application** for handling sensitive patient data.

3. Objective

The main objectives were:

- To build reliable healthcare modules using TDD principles.
 - To ensure each component is tested before development.
 - To reduce integration issues and rework.
 - To increase overall code quality and maintainability.
 - To meet compliance standards such as HIPAA through robust testing.
-

4. TDD Approach

The team followed the **Red-Green-Refactor** cycle:

1. Write a Test (Red):

Before writing the feature, a unit test was created.

Example: For patient registration, a test was written to verify valid patient data inputs.

2. **Run the Test (Fail):**

Initially, the test failed since no functionality existed yet.

3. **Write Code (Green):**

Minimal code was written to make the test pass.

4. **Refactor:**

The code was cleaned and optimized while keeping tests green.

5. **Repeat:**

This cycle continued for each feature such as vital sign tracking, emergency alert notifications, and real-time hospital data sync.

5. Implementation

- **Technology Stack:**

Java, Spring Boot, JUnit, Mockito, REST APIs, and MySQL.

- **Modules Covered:**

- Patient Registration and Authentication
- Vital Signs Monitoring
- Emergency Alerts Module
- Doctor-Patient Communication Interface

- **Testing Tools Used:**

- JUnit for unit testing
- Mockito for mocking dependencies
- SonarQube for code quality checks
- Jenkins for continuous integration

6. Challenges Faced

- **Initial Resistance:** Developers initially found it slower to write tests first.
- **Complex Test Cases:** Medical data validation required carefully designed test scenarios.
- **Integration with Third-Party APIs:** Ensuring test coverage for external API failures was challenging.

To address this, the team conducted **training sessions**, created **TDD templates**, and **gradually increased coverage** over sprints.

7. Outcomes & Benefits

- **Defects Reduced by 45%** in the first 3 sprints.
 - **Improved code confidence**—developers were less afraid of introducing changes.
 - **Faster releases**—due to automated testing and CI/CD integration.
 - **High compliance** with data integrity and security standards.
 - **Stronger team collaboration**, as tests clearly defined expected behavior.
-

8. Conclusion

The successful implementation of TDD in the healthcare application not only improved software quality but also enhanced patient safety and trust. By **shifting testing to the start of development**, the team delivered a **robust, secure, and compliant system** with minimal production issues.

TDD proved to be more than a testing technique — it became a **development mindset** that aligned well with the sensitive and critical nature of healthcare software.

THANK YOU