7-2 Project 2: Summary and Reflections Report

Eduardo Gonzalez

Southern New Hampshire

CS 320 Software Test, Automation, and Quality Assurance

Professor Wilson

June 24, 2024

For Project One we used a unit testing approach to the Contact Service, Task Service, and Appointment Service feature development. JUnit tests are created to align with the application requirements. For the Contact Service feature, we created unit tests to ensure each contact had a unique ID, a first name, a last name, a phone number, and an address. Each of these inputs had specific constraints on length and nullability. Our tests were designed to validate their creation and handle invalid input scenarios. For the Task Service, we created unit tests to ensure each task had a unique ID, name, and description, all with constraints specified by lengths and nullability requirements. The tests verified task creation, updating, and deletion within the service. We created unit tests for the Appointment Service feature to validate that each appointment had a unique ID, date, time, and description with the necessary validations for each input. Our tests ensured our program could handle appointment scheduling, updating, and deletion. The unit testing approach aligned with the software requirements. For example in the task service, the test to ensure task IDs were neither null nor longer than 10 characters was implemented as:

@Test

public void testTaskConstructorInvalidId() {

assertThrows(IllegalArgumentException.class, () -> new Task(null, "Task Name", "Task Description"));

assertThrows(IllegalArgumentException.class, () -> new Task("12345678901", "Task Name", "Task Description"));

}

Our test addresses the requirement that the task ID be unique and no longer than 10 characters long.

The quality of the JUnit tests can be defended based on their comprehensive coverage. Each functional requirement and constraint was validated by multiple test cases, ensuring high coverage. The effectiveness of the tests is evident from successful test runs without errors or failures. Technical soundness was ensured by adhering to best practices such as exception handling and boundary value analysis.

@Test

public void testSetName() {

Task task = new Task("1", "Task Name", "Task Description");

task.setName("New Name");

assertEquals("New Name", task.getName());

}

This test confirms that the setName method correctly updates the task name while adhering to our constraints.

Efficiency was achieved by making the code concise and avoiding unnecessary computations. For example, in the updateTaskName method:

public void updateTaskName(String taskId, String name) {

Task task = taskMap.get(taskId);

if (task != null) {

task.setName(name);

}

}

This method checks for the existence of the task before attempting to update it, minimizing unnecessary operations.

During this project, several testing techniques were employed to ensure the robustness and reliability of the software. Unit Testing was a primary focus, validating individual components for correctness and ensuring they met the specified requirements. This technique is important for early detection of defects and for confirming that each component functions as intended. Boundary Value Analysis was another key technique used to test the limits of input constraints. This allows us to ensure the software behaves correctly at the edges of allowable values. Exception handling was incorporated to ensure the system can handle invalid inputs and maintain stability. Several testing techniques were not used in this project. System Testing, an end-to-end testing of the complete system, was also not part of this phase. User Acceptance Testing (UAT), involving validating the system with real-world scenarios and user requirements was not used. These techniques have practical uses in different stages of software development. Integration Testing is crucial for identifying issues that arise when components interact. System Testing ensures the entire system functions correctly as a whole, and UAT ensures the system meets their needs and expectations.

A cautious mindset was crucial for this project. Understanding the complexity and interrelationships of the code being tested was essential. For example, ensuring that task IDs were unique and not null required careful validation across different parts of the application. To limit bias, I reviewed the code objectively. Putting a focus on potential issues rather than assuming it was error-free. Writing rigorous tests minimized the risk of bias. For example, validating input constraints in the task constructor helped ensure thorough error checking. Being disciplined in committing to quality is essential. Cutting corners can lead to technical debt that can lead to costly to address later. To avoid this I maintained comprehensive documentation, conducted regular code reviews, and updated tests to cover new edge cases. This structured and disciplined approach to testing and development ensures the reliability, efficiency, and maintainability of the software. This foundation is critical for future projects to ensure the creation of robust and quality software.