Christina Jimenez

Southern New Hampshire University

CS-320

23 August 2024

Module 7 Project Two

Summary

* 1. **Describe your unit testing approach for each of the three features.**
     1. **To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.**

My testing approach was carefully designed to align with the specific requirements provided by the client. By focusing on each criterion individually, I ensured that every aspect of the program was thoroughly evaluated. For example, if a requirement specified a character limit for a field, I created tests to verify that the program correctly handled inputs that exceeded this limit. This targeted testing approach ensured that the program met all client specifications and caught any exceptions where it did not.

* + 1. **Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were effective based on the coverage percentage?**

The overall quality of my JUnit tests can be supported based on their comprehensive coverage of potential errors. For each requirement, I developed tests that checked for cases where an object either exceeded character limits or was null. These tests effectively ensured that the program adhered to the defined requirements.

* 1. **Describe your experience writing the JUnit tests.**

1. **How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

I ensured that my code was technically sound by making it readable, maintainable, and testable. For readability, I used proper comments and spacing throughout the program. This also enhanced maintainability, as the code is easier to understand and modify. For example:

// Deletes a task

public void deleteTask(String taskID) {

if (!tasks.containsKey(taskID)) {

throw new IllegalArgumentException("Cannot delete task. Task ID not found.");

}

tasks.remove(taskID);

}

The comment helps with the understanding the code, if the tests were not passed, for example, the test showed that task could not be deleted we could easily find the part of the program that may be causing then error.

**ii. How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

To ensure efficiency, I implemented tests that validated requirements in an optimized manner. Using the assertThrowsmethod allowed me to efficiently test for exceptions in a single line. This streamlined approach ensured that the code met requirements in an optimized and maintainable way. Examples of this include:

@Test

public void testTaskCreationInvalidDescriptionLength() {

IllegalArgumentException exception = assertThrows(IllegalArgumentException.class, () -> {

new Task("1234567890", "Christina", "descriptionnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn");

});

assertEquals("Invalid description.", exception.getMessage());

}

This test specifically checks for an exception related to the description length, ensuring the program can catch such exceptions efficiently.

1. **Reflection**
2. **Testing Techniques**
3. **What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.**

For the milestones, I employed unit testing and integration testing. Unit testing was used to validate individual methods. For example, in the appointment milestone testing was used to validate methods such as appointment creation, ensuring they met the specified requirements like ID length and date validity. This technique allowed for isolated testing of each component, catching issues early. Integration testing was employed to verify that various components, such as appointment creation, deletion, the ability to read and update the appointments, making sure each operation worked together correctly. This ensured that the system's components interacted as expected and maintained data integrity across operations.

**ii. What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.**

The techniques I did not utilize in the milestone include the following, system testing, user acceptance testing (UAT), and performance testing. System testing involves testing the entire application in an environment like production, ensuring the complete system functions as expected, this was not needed at the time. User acceptance testing focuses on end-users validating the system based on real-world scenarios to ensure it meets their needs before deployment, the program is not currently in this phase of development. Performance testing assesses how the system performs under various conditions, such as load and stress, to identify potential bottlenecks and ensure scalability and reliability, which were not tested in the milestones.

* + 1. **For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**

Unit testing is crucial for catching bugs early in development by validating small, individual components of the application, as we did at every milestone. It is useful for ensuring that code changes do not introduce new issues, and since sections are tested at one time, the cause of errors can be narrowed down. Integration testing helps verify that different components work together seamlessly, which is vital for detecting issues that arise from component interactions. On the other hand, system testing ensures that the complete application functions correctly in a production like environment. UAT provides final validation from the user's perspective, ensuring the software meets their expectations. Performance testing is essential for assessing how the application handles varying loads and stress conditions, which is crucial for applications expected to scale or handle high traffic.

1. Mindset
2. **Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ caution? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.**

In this project, I adopted a detail-oriented mindset. By acting as a software tester, I had to focus on the function and details of the code to ensure efficiency. Attention to detail was crucial for designing comprehensive test cases that addressed not only the functionality of the application but also the broader system impacts. For example, while testing appointment date validation, I considered how this logic might affect related services so input validation in this case was crucial for the functionality of the system as a whole.

ii. **Assess the ways you tried to limit bias in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.**

To limit bias in my code review I approached the code objectively, focusing on functionality and creating test cases which would prove the functionality of the application based on the client’s requirements. I recognize that testing my own code and bias could be a concern, as coder and tester this may cause me to overlook issues or only test expected scenarios. To counter this, I would approach my code critically, as if it were written by someone else, ensuring a thorough and unbiased review, ideally it would be more beneficial for someone else to review my code.

**iii. Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.**

I am committed to delivering quality. It is crucial as a software engineering professional to pay close attention to details because cutting corners in writing or testing code can lead to long-term issues, including bugs, security vulnerabilities, and increased maintenance costs, all which can harm the clients and users, and your own reputation as a developer. Avoiding technical debt is key to ensuring sustainable, high-quality software development. For example, by thoroughly testing each feature and refactoring code, when necessary, I can prevent small issues from becoming larger problems later. Testing code throughout helps ensuring the functionality of the program in general. I plan to avoid technical debt by adhering to coding best practices, such as writing clean, well-documented code, conducting regular code reviews, and prioritizing comprehensive testing over quick fixes.