Brett Nottmeier

Desmond Springer

CS-320

08/18/2024

Module 7 Project Two

For each of the three services, Contact, Task, and appointment, the unit testing focused on validating the required requirements to adhere to standards and prevent bugs from entering the software.

* Contact Service: The test ensured that the Contact ID, first name, last name, and phone number all met specific requirements. For example, it checked whether the ID was unique and non-null.
* Task Service: The test verified that the task ID was unique and all information met the character length.
* Appointment Service: Verified the appointment ID was unique, which was unique, the date was not in the past, and the description length was.

My JUnit test was adequate based on my 77.9 coverage percentage. Most of the tests were covered, with only a few exceptions. All contact service classes covered all requirements, including adding and deleting contact IDs. The Task Service class also met the requirements by adding and deleting tasks based on ID and making contact info updatable. Overall, my test had effective coverage by ensuring valid inputs and invalid inputs and checking input length and effectiveness.

I ensured my code was technically sound by writing tests that validated if it was successful or failed. For example, in TaskServiceTest the testAddTaskSuccess verifies that the task is added and retrievable.

String taskID = taskService.addTask("Task Name", "Task Description"); Task task = taskService.getTask(taskID); assertNotNull(task); assertEquals("Task Name", task.getName());

I ensured efficiency by focusing on functionality and avoiding unnecessary code for

example the generateUniqueID method in TaskService class creates a unique ID in a

ingle step.

private String generateUniqueID() { return UUID.randomUUID().toString().replace("-", "").substring(0, 10); }

For each milestone, we used JUnit 5 to verify program units. This week, we specifically targeted the Appointment and AppointmentServices class to ensure that all requirements were met and working as intended.

We do this by developing test cases that ensure each function written in the classes works, such as validating inputs, handling edge cases, and ensuring proper execution. This approach is called test-driven development, where test cases are written before the code that validates them, providing the code meets the requirements from the beginning and throughout the program's development.

**Integration testing**: A type of testing that confirms the data exchange and interactions between different program components. This testing can identify a problem integrating various components, typically performed after unit testing.

**System testing**: This type of testing, known as black-box testing, evaluates the functionality and performance of an integrated program. It is typically conducted after integration testing has been completed. The goal is to identify any flaws in integrated components and follow all system requirements.

Unit testing is for all projects and ensures that all software requirements will be met from the beginning by checking each unit with test cases written before development begins. This testing before the build can help catch early bugs, simplify debugging, and prevent technical debt from the building by checking that the code works as intended before moving on to other parts of the program.

Integration testing is significant for projects with multiple components or services that connect or interact with one another. This integration check ensures they work seamlessly together and identify any integration issues in a complex system.

System testing is important for validating a complete software program or application, checking the system compliance with the requirements, and meeting the stakeholder's expectations when released or out of production. This testing is excellent for large projects that are getting ready to launch.

While working on this project, I approached it with the mindset of identifying each component's role while working on them independently and determining how the software would fit together for project one. I focused on making sure the requirements were met for each test case. For example, I considered the attempts to update a contact with an invalid phone number or delete nonexistent contacts in Contact Service.

Recognizing the complexity of test cases and what can go wrong can be important. For example, in Appointment services, we want the date to be set in the future and not in the past. We need to think about what could go wrong, input proper test cases for such things, and recognize the complexity of the code we are implementing.

It can be challenging to limit bias when reviewing your own code. When writing them, I separated my roles between developer and tester. I focused only on the requirements to ensure the test would be valid and functional rather than on how I would implement the code. When coding TaskServiceTest, I tested the functionality by considering the requirement specifics, such as task name and description.

When coding and creating my test cases, I had to commit to quality and avoid cutting corners. Saving time in the short term can lead to technical debt and more time fixing issues in the future or completely redoing the work. I did this by sticking to the requirements and building my test cases first, so I knew what to write and stuck to completing each task with the correct requirements and features. By maintaining quality above all else and commitment to requirements and not cutting corners will help immensely in my future career, help prevent technical debt, and ensure my programs are reliable and easy to implement new features or changes.

Work Cited:

GeeksforGeeks. (2022, October 31). *Test driven development using Junit5 and Mockito*. <https://www.geeksforgeeks.org/test-driven-development-using-junit5-and-mockito/>

GeeksforGeeks. (2024c, July 30). *Integration testing - software engineering*. https://www.geeksforgeeks.org/software-engineering-integration-testing/

testsigma. (n.d.). *System testing: What it is, why, types & how to perform?* https://testsigma.com/guides/system-testing/

GeeksforGeeks. (2024b, June 4). *Introduction to junit 5*. https://www.geeksforgeeks.org/introduction-to-junit-5/

Jatin. (2023, November 30). *8 types of test cases in software testing you should know*. QACraft. https://qacraft.com/types-of-test-cases-in-software-testing/

GeeksforGeeks. (2024b, June 19). Static testing - software testing. https://www.geeksforgeeks.org/software-testing-static-testing/

GeeksforGeeks. (2023b, December 27). Dynamic testing - software testing. https://www.geeksforgeeks.org/software-testing-dynamic-testing/