Aspen Blue Koreny-Crawford

Southern New Hampshire University

CS 320 Software Test, Automation QA

12 December 2024

Project Two

In the Grand Stand Systems CS 320 project, I used unit testing techniques to develop and validate a mobile application's contact, task, and appointment services. Implementing these services with in-memory data structures, I created JUnit tests to ensure they met the specified requirements and identified potential errors.

For the contact service, I focused on validating the uniqueness of the contact ID and enforcing constraints on fields like first name, last name, phone number, and address. My approach using JUnit’s `assertThrows`, I ensured that invalid inputs, such as null values or excessive character lengths, triggered appropriate exceptions. I also tested adding, deleting, and updating contacts. The `testAddContact` method confirmed new contacts were added successfully, while `testDeleteContact` verified that contacts were removed correctly. The `testUpdateContact` method checked that updatable fields, like first name and address, were modified, while the contact ID remained unchanged.

In the task service, I focused on validating the task ID, name, and description fields to meet specified requirements. The `testUpdateTask` method ensured that the task name and description could be updated without changing the unique task ID. I also created tests for adding and deleting tasks, such as the `testDeleteTask` method, which verified that tasks were inaccessible after deletion.

I tested the appointment service by checking the appointment ID, date, and description fields. The tests confirmed that the appointment date was a future date and not null. The `testInvalidAppointmentDate` method ensured that creating an appointment with a past date caused an exception. I also verified that appointments with valid data were stored correctly and could be deleted as requested.

My unit testing approach was closely aligned with the software requirements. Each test case targeted specific requirements, those such as validating field constraints and service method functionality. For instance, the testContactInvalidID method in the contact service ensured that null or overly long IDs were rejected. By addressing edge cases and boundary conditions, I confirmed that the implementation met its intended functionality. The green bar in the Eclipse JUnit runner indicated that all tests passed without errors, demonstrating the effectiveness of the tests.

A systematic approach was needed to develop the JUnit tests. I included exception handling and validation checks in both of the service classes and test cases. For example, in the task service, I used code to validate field updates:

```

service.updateTask("12345", "Updated Name", "Updated Description");

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

assertEquals("Updated Description", task.getDescription());

```

This ensured the updates were applied efficiently. Additionally, I streamlined testing by combining checks into a single method, avoiding redundancy while maintaining coverage.

Alongside unit testing, I performed black-box testing to validate functional outputs against the requirements. I added valid appointments and confirmed their presence, showing that the services worked as expected. However, I didn’t use integration or system testing since the project focused solely on individual services without a user interface or database.

Throughout the creation of this project, I maintained a cautious mindset and thoroughly reviewed the requirements, considering all edge cases. I tested both the valid and invalid inputs to help ensure comprehensive coverage. To minimize bias, I also approached testing objectively, validating each service based on the requirements. This methodical structure helped reduce the risk of overlooking potential issues.

Discipline and attention to detail are crucial for ensuring code and test quality. Skipping testing steps can lead to vulnerabilities. For instance, testing character limits for task names and descriptions prevents potential issues. As a student and future software engineer, I will avoid technical debt by following best practices, writing thorough tests, and documenting my work.

This project provided valuable experience in developing and testing back-end services. The unit tests ensured the reliability of the contact, task, and appointment services, highlighting the importance of disciplined testing in software development.