**7-2 Project Two Submission**

Lauren-Ashley Black

August 13, 2023

My approach to unit testing was strongly aligned with the software requirements. My test suite covered the core functionalities and edge cases described in the software requirements for each feature (Contact, Task, and Appointment services). For example, in the Contact Service, I had tests to ensure the addition, updating, and deletion of contacts, validating the adherence to the specified requirements. The evidence of this alignment can be found in the comprehensive test methods, such as testAddContact, testUpdateFirstName, and others, which directly mirror the operations expected from the Contact Service.

To ensure the effectiveness of my JUnit tests, I focused on achieving a high coverage percentage while also addressing different test scenarios and edge cases. I aimed to cover all branches of conditional statements, loops, and logical operators in my code. This approach helps verify that the code behaves as expected under various conditions. I also made sure to include positive and negative test cases, boundary tests, and tests for exceptional scenarios to maximise coverage.

I guaranteed the technical soundness of the code in the ContactService class by creating the JUnit tests that covered various methods and scenarios. I focused on testing the different functionalities of the ContactService class, including adding, deleting, updating, and looking up contacts. In this test, I create a ContactService instance and add a Contact object. I use the assertTrue assertion to ensure that the contact was added successfully. Then, I use assertNotNull to confirm that the added contact can be looked up using the lookupContact method.

@Test

void testDeleteContact() {

ContactService contactService = new ContactService();

Contact contact = new Contact("1", "laurie", "testing", "1234568790", "1265 2nd st sw");

contactService.addContact(contact);

assertTrue(contactService.deleteContact("1"));

// Assert that the contact was deleted and cannot be looked up

assertNull(contactService.lookupContact("1"));

}

To ensure that the code in the ContactService class was efficient, I designed my JUnit tests to cover different functionalities while also considering edge cases and scenarios that could impact performance. While the specific efficiency of the code might not be directly measured in the unit tests themselves, the choice of test cases and the design of the tests can provide insights into the efficiency of the code. In this test, I efficiently deleted a contact using the deleteContact method and then verified that the contact could not be looked up using the lookupContact method. This ensures that contact deletion is an efficient process.

@Test

void testUpdateFirstName() {

ContactService contactService = new ContactService();

Contact contact = new Contact("1", "laurie", "testing", "1234568790", "1265 2nd st sw");

contactService.addContact(contact);

assertTrue(contactService.updateFirstName("1", "jordii"));

// Look up the contact and assert that the first name was updated efficiently

Contact retrieved = contactService.lookupContact("1");

assertEquals("jordii", retrieved.getFirstName());

}

Unit testing is essential for validating the correctness of individual units or components of a software application. It is particularly useful during the development phase to catch and fix bugs early, ensuring that each unit functions as expected before integration. In agile development, where frequent code changes are made, unit testing provides rapid feedback on the impact of changes. It also enhances code maintainability, as any modifications can be validated without breaking existing functionality. Unit testing is crucial for achieving high code coverage, identifying edge cases, and promoting modular development practices.

Boundary testing is used to validate the behaviour of software at the edges of valid and invalid input ranges. It helps identify potential vulnerabilities due to unexpected or out-of-bounds inputs. Boundary testing is particularly effective in catching issues related to data validation, input sanitation, and avoiding buffer overflows. It aids in improving the robustness and security of the software.

Regression testing ensures that new code changes do not introduce new bugs or regressions into previously working functionalities. It safeguards against unintended side effects. Regression testing is crucial in large-scale projects or those with frequent updates. It maintains software quality over time by verifying that existing functionalities remain intact even as new features or bug fixes are introduced.

Being disciplined in one's commitment to quality is of paramount importance for software engineering professionals. Cutting corners in writing or testing code can lead to a host of problems that negatively impact the software's reliability, maintainability, and overall success. The following points highlight the significance of maintaining a high level of discipline in software development: Cutting corners can result in undetected bugs or vulnerabilities that may surface later, leading to system failures or security breaches. Commitment to quality ensures that these issues are identified and resolved before they cause significant harm. Code that is not well-designed or thoroughly tested can become difficult to understand, modify, or extend over time. High-quality code is easier for other team members to comprehend and work on, reducing maintenance challenges.

I employed several software testing techniques to ensure the quality and reliability of the developed code. These techniques included unit testing and boundary testing. I extensively used unit testing, especially with JUnit, to test individual components in isolation. For example, in the ContactServiceTest class, I wrote test methods for each method in the ContactService class, such as testAddContact, testDeleteContact, etc. This technique allowed me to verify the correctness of each method's behaviour and handle various scenarios. In both the ContactTest and TaskTest classes, I conducted boundary testing to validate how the system handles extreme values or edge cases. For instance, I tested the upper limit of character lengths for attributes like contactId, firstName, and lastName, as well as the lower limit by passing null values.

There are several other software testing techniques that I did not use for this specific project. If a bug was fixed or a new feature was added to the Contact Service, regression testing would involve re-running the existing test suite for the Contact Service to make sure that these changes did not break any previously working functionality.During the course of working on this project, I adopted a diligent and cautious mindset as a software tester. I recognized the importance of thorough testing to identify potential issues early and ensure the overall quality of the software. I paid meticulous attention to details while designing and executing tests. I employed exploratory testing to delve deeper into the code's behaviour beyond the scripted test cases.

In reviewing the code, I made conscious efforts to limit bias and maintain an objective perspective. Bias in code review can lead to overlooking issues or favouring certain aspects, potentially compromising the overall quality of the software. To minimise bias, I employed the following strategies: I followed established coding standards and guidelines while reviewing the code. This allowed me to evaluate the code based on established best practices and objective criteria rather than personal preferences. I prioritised evaluating the code against the functional requirements and specifications provided. By aligning my review with the intended functionality, I could objectively assess whether the code met its intended purpose.