7-2 Project Two Submission

Brennan Haggett

02/17/2025

**Summary**

* **Describe your unit testing approach for each of the three features.**

For each of the three features contact service, task service, and appointment service, I followed a structured approach using JUnit to verify their functionality. I created test cases for adding, updating, and deleting records, this was done for each feature behaving as expected to mee the client’s requirements. For the contact service, I tested the validity of contact IDs and other fields, to make sure they met the required criteria. In the task service, I focused on ensuring tasks could be added, updated, and deleted while sticking to constraints like task name and description length. For the appointment service, I made sure that appointments could be added with valid dates and descriptions, and that they could not have been in the past. My tests covered edge cases, such as invalid data inputs.

1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

The unit testing approach I created aligned well with the software requirements. Each feature requires validation for specific fields contact ID, first name, last name, phone, task ID, name, description, and appointment date. I tested these requirements, ensuring the system rejected invalid values and only allowed data conforming to the specifications. An example of this applied is the contact service tests checked for unique IDs and non-null values, ensuring everything meets the requirements. I also included test cases for error scenarios, such as invalid dates for appointments and duplicate IDs for contacts and tasks. This comprehensive approach ensured that all system requirements were met, and that the application could handle expected use cases and edge cases.

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 success of the JUnit tests can be defended through both the coverage percentage and the accuracy of test cases. I used a combination of positive and negative tests to ensure that all system behaviors were validated. The tests accounted for edge cases, such as invalid data inputs, and the system's handling of exceptions like duplicate IDs. With code coverage exceeding 80%, I was confident that the core functionalities of adding, updating, and deleting contacts, tasks, and appointments were thoroughly verified. The tests not only ensured the features worked under normal conditions but also identified potential bugs, making the tests an effective safeguard against defects in the system.

* **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.

To ensure the code was technically sound, I focused on validating all requirements and making sure their exception handling for edged cases. The example used in the ContactServiceTest was that I validated that the contact ID was not null and did not exceed 10 characters.

This test was done to make sure that invalid inputs would trigger the expected exceptions. AppointmentServiceTest also had this approach, I made sure to appointment, and it was not in the past.

assertThrows(IllegalArgumentException.class, () -> new Appointment("123", new Date(System.currentTimeMillis() - 1000), "Description"));

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

To make the code efficient, I worked on minimizing repetitive logic and ensuring that each test is focused on a single functionality. An example of this is instead of writing separate tests for similar scenarios, I created reusable methods for common setups. In the TaskServiceTest, the creation of task objects was handled efficiently using a helper method to reduce redundancy.

private Task createTask(String taskId, String name, String description) {

return new Task(taskId, name, description);

}

The method is used across multiple tests, such as testAddTask() and testDeleteTask(), to make sure there is efficient task creation. By reusing the code, I was able to reduce test duplication and maintain cleaner, more efficient tests. This meets the requirements and

**Reflection**

* **Testing Techniques**

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

I employed unit testing as the primary software testing technique. Unit testing involves testing individual components or units of code in isolation to ensure they function correctly. This method helps to catch bugs early and makes sure each part of the application performs as expected. I used JUnit as the testing framework, which allowed for easy setup, execution, and validation of test cases. An example of this used in the project is in the ContactServiceTest, I wrote tests for adding and deleting contacts information, verifying that the expected actions occurred, such as ensuring the size of the contact list changed correctly.

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

Other software testing techniques not used in this project were integration testing and system testing. Integration testing focuses on verifying the interactions between different modules or components to ensure they work together. Testing how ContactService interacts with other services such as TaskService would fall under this categorization. System testing will include validating the entire application as a whole, ensuring all components function correctly together and meet the overall software requirements.

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

The Unit testing that I used in this project was there to make sure that the individual components or the methods that were implemented worked correctly in isolation so that each unit is able to withstand its own logic to meet the applicant requirements and the code flows correctly with no errors. It is practical for validating specific logic and functionality early in the development process, making it easier for the programming going forward falling the correct requirements and making the workload easier, so that code doesn’t need to be redone halfway or at the end of the coding process. This can be a huge burden when projects become very in depth and go back to looking for errors or logic hiccups after all is done.

Integration testing verifies that different modules or services interact correctly, which is essential when systems are composed of multiple interconnected parts. System testing checks the entire application’s functionality, ensuring it meets the overall requirements and works in a real-world environment.

* **Mindset**

1. 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.

I adopted a cautious mindset while working as the software tester, understanding the importance of testing each feature thoroughly to ensure reliability and functionality. I appreciated the complexity of the code by recognizing how various components interacted, especially when updating or deleting data in the services. When testing the contact service, I made sure to account for the unique contact ID and field constraints. The test assertThrows(IllegalArgumentException.class, () -> contactService.addContact(contact)) ensured that the system appropriately handled invalid input. Recognizing these interrelationships helped avoid potential issues like data corruption or unexpected behavior, ensuring that the application functions as intended when integrated.

1. 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 review, I focused on test cases that challenged edge cases and invalid inputs rather than just verifying expected behavior. I reviewed my tests carefully, questioning whether I overlooked potential failures. As a developer testing my own code, bias could emerge by subconsciously overlooking flaws I didn’t want to acknowledge. For example, while testing the appointment service, I made sure to test both valid and invalid dates to avoid bias by only testing for the expected valid inputs.

1. 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.

Being disciplined in maintaining high standards for code quality is important in software engineering. Cutting corners during development or testing can lead to issues like bugs, performance problems, and security vulnerabilities, which could harm both the user experience and the long-term sustainability of the software. Neglecting comprehensive testing of edge cases might result in unexpected errors during real-world use. To avoid technical debt, I plan to adopt the best practices, including thorough code reviews, continuous integration, and writing unit tests. I’ll prioritize long-term stability over quick fixes by ensuring that each feature is properly tested and refactored.