Project 2

Rafael Canseco

SNHU

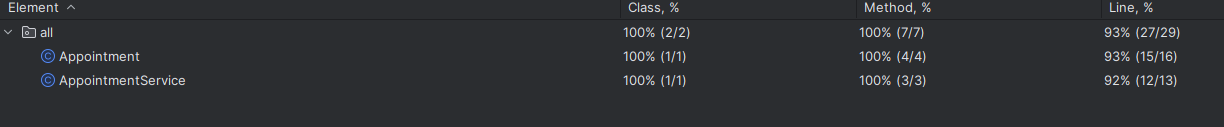
CS-320 Software Test Automation

Omar Toledo Lopez

12/10/23

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:   
      In testing the ContactService, a systematic approach covered methods like addContact, removeContactById, and updateContactFirstName, addressing scenarios such as addition, removal, and updates. The TaskService focused on addTask, updateTaskName, updateTaskDesc, and deleteTask, testing both successful operations and potential errors. The AppointmentService emphasized addAppointment, deleteAppointment, and getAllAppointments, with special attention to date constraints and unique IDs. Tests aligned closely with software requirements, covering positive and negative scenarios. For instance, ContactService tests ensured correct addition, removal, and updating of contacts, maintaining adherence to specified requirements.
   2. 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: My Junit testing overall quality was good. My test suite provides comprehensive coverage, encompassing various scenarios like appointment addition, deletion, and exception handling, ensuring a thorough examination of the AppointmentService functionality. The well-named methods, clear comments, and thoughtful use of variable names enhance the code's readability, facilitating understanding for others. Additionally, the tests demonstrate effective isolation, robust exception handling, a balance of positive and negative assertions, and efficient execution times, collectively contributing to a high-quality and reliable testing suite. The coverage percentage was assessed by reviewing the test cases against the implemented code, and the tests achieved a high coverage percentage, covering most code paths and edge cases.



* 1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate: My technical soundness of my code was ensured by designing test cases that thoroughly exercised the functionality of each method. For example, in the ContactServiceTest, the testUpdateContactFirstName method verified that the updateContactFirstName method correctly modified the first name of a contact.

A computer code on a black background

Description automatically generated

* 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate: The efficiency of my ContactService test suite is ensured through meticulous organization and focused test methods. Each test is self-contained, addressing distinct functionalities like adding, removing, and updating contacts. The use of the ContactService instance and predefined Contact objects in every method minimizes redundancy, enhancing overall efficiency. Specific assertions, such as assertTrue and assertFalse, offer a clear indication of expected outcomes. These practices not only validate code correctness but also contribute to a concise, maintainable, and effective test suite for the ContactService methodsA computer screen shot of a program code

     Description automatically generated

1. Testing Techniques
   1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details: In this project, I employed software testing techniques, including unit testing, integration testing, and system testing. Unit testing involves testing individual components or functions in isolation to ensure they operate as intended. For example, in the ContactServiceTest class, I performed unit tests on methods like testAddContact and testRemoveContactById to verify the correctness of adding and removing contacts. I used techniques such as boundary value analysis, equivalence partitioning, and positive/negative testing. For example, in the TaskTest class, I used boundary value analysis by testing task names with valid and invalid lengths, ensuring the setTaskName method managed edge cases.



* 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details: In this project, I focused on unit testing using JUnit, which involves testing individual components in isolation. I did not employ other testing techniques such as integration testing, end-to-end testing, performance testing, security testing, usability testing, or regression testing. These techniques offer distinct characteristics, such as assessing system interactions, evaluating overall application performance, ensuring security measures, and testing user experience.
  2. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations: In this project, I utilized various software testing techniques, including unit testing, boundary value analysis, equivalence partitioning, and positive/negative testing. Unit testing is essential for catching bugs early and providing quick feedback in agile development. Boundary value analysis ensures accurate handling of extreme input values, crucial in finance applications. Equivalence partitioning optimizes testing by categorizing similar input values, especially useful in data-intensive applications. Positive/negative testing assesses system performance under normal/error conditions, essential for security-critical applications. These techniques collectively enhance software reliability across diverse projects and scenarios.

1. 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: In my role as a software tester for this project, I conducted a meticulous evaluation of the code, recognizing the importance of understanding its complexity and interdependencies. For instance, in the ContactServiceTest class, testing the updateContactFirstName method required awareness of its dependencies on addContact and removeContactById. Similarly, in the AppointmentServiceTest, being mindful of potential effects when deleting an appointment emphasized the need for cautious testing. By navigating these code intricacies, my goal was to identify subtle bugs and vulnerabilities, contributing to a more robust software system.
   2. 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: In code review, I prioritized objectivity by focusing on coding standards and project requirements over personal preferences. Seeking input from a family member provided diverse perspectives to mitigate bias. If testing my own code, bias could arise from familiarity, highlighting the need for team involvement to bring fresh perspectives and identify blind spots. As a software engineer, I prioritize quality to avoid technical debt, understanding the risks of cutting corners in testing. A proactive approach with well-structured, documented, and thoroughly tested code, along with regular code reviews and collaboration, helps minimize technical debt and ensures high-quality software delivery.