Jesse Morin

Module 7 Assignment

**Summary**

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

* **To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.**
  + **Appointment**
    - Approach: I implemented unit test that were able to validate creation, deletion, and manipulation methods. I also implemented tests to ensure input validation and ensure required lengths, required attributes and input validation.
    - Alignment: I made sure to match each function with the specified requirements by using unit test with different scenarios like null inputs and invalid formats to ensure the app behaved correctly.
  + Contact
    - Approach: Implemented unit test to validate creation, deletion, and manipulation. Tests to ensure input validation and correct format. For example ensuring we cannot create Contacts with null Id’s or last names.
    - Alignment: Matching each test with specified requirements to ensure alignment. For example we tested adding and deleting contacts
  + Task
    - Approach: Unit Tests to validate creation, deletion and updating. Test to ensure input validation and correct format. For example testing to ensure we cannot create tasks with null ID’s or names.
    - Alignment: Matching tests to specific requirements. For example ensuring we can create, delete and update tasks and ensure we return store the correct values.
* **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?**
  + I ensured that my unit test were effective and ensured coverage by testing all scenarios where we would not align with the requirements and when requirements were met. For Example, appointmentIDTooLong and dateInPast tests ensured that these requirements were met without invalid inputs or states.

**Describe your experience writing the JUnit tests.**

* **How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**
  + We ensured technical soundness by testing specific examples that prove alignment and functionality. For example here we test for a valid appointmentID:

**@Test**

**void appointmentIDTooLong(){**

**assertThrows(IllegalArgumentException.class, () -> {**

**new Appointment("12345678910", LocalDate.of(2024, 2, 25), "testDescription");**

**});**

**}**

**In these assertion tests we could ensure that we could properly create tests, or that we could catch the right input errors when needed**

* **How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**
  + We ensured efficiency by reusing setup code to ensure we weren’t overloading our memory. We show this in this example where we use the Junit @beforeeach method which constructs and deconstructs our objects efficiently. This ensures that we can efficiently create and deconstruct an instance of the class at runtime to introduce efficient memory management.

@BeforeEach

void setUp() {

service = new AppointmentService();

}

**Reflection**

**Testing Techniques**

* **What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.**
  + Edge Cases: Ensured to test edge cases to ensure that our methods alignment with the requirements, for example, length of values, null values, and format of values.
  + Equal Values and Assertions: Ensured to test specific values to ensure that we were able to change and manipulate values effectively, for example we were able to test service creation such as adding a service by creating an instance locally, and then getting the service from our data stored to ensure that we successfully or failed to create or add services.
* **What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.**
  + Stress Testing: We did not stress test these classes by implementing high number of implementations with different scenarios.
  + Interaction Testing: We did not test all the different interactions that a classes and methods could interact with each other.
* **For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**
  + Edge Case: This would be used when a system has dynamic inputs and scenarios, where you can test how the system behaves to each scenario
  + Equal Values: This would be used when you ensure behavior of implementation of functions to ensure you are getting the right values or the right values are null
  + Stress Testing: This would be used when multiple users or other systems would the using the application to see how the system behaves with high traffic
  + Systems Testing: this would be for larger systems to see how the system behaves and interacts with each other.

**Mindset**

* **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.**
  + The mindset I implemented was to ensure an understanding of how the system is supposed to behave to the requirements of the system. I adopted a cautious mindset to understand interrelationships, and the edge cases the system may face and how it is supposed to react. For example, ensuring that the state of a date is in the future demonstrates how this constraint interacts with other functionalities.
* **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.**
  + By keeping a neutral perspective and understanding the different edge cases the system may face, I was able to reduce risk for confirmation bias. It is critical to test different scenarios that could make or break the system. Keeping a outsider perspective is useful to understand the different scenarios the system may face to avoid bias.
* **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.**
  + It is very important to stay disciplined and unemotional when building for quality and robustness. When cutting corners, you can lead your code open to vulnerabilities and risk not meeting requirements. This can lead to less effective and efficient development as cutting corners can lead to technical debt which leads to you or other developers having to go back and fix problems or broken code in the future. Testing all scenarios and ensuring the system behaves as it should is a great step in the right direction especially when developing for production or for clients.