Darren Sheftic

Software Testing, Automation, QA

Prof. Angelo Luo

2/21/2024

Module Seven Project

1. 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.

The code was designed to meet the client requirements of the software, due to the basic nature of the code there was minimal exception handling needing to be done, with testing encompassing possible common errors, such as capitalization, upper and lower bounds or adding in characters that don’t belong and possibly cause errors. This was achieved through || / OR Booleans and if-else statements in the code to cover all bases that input was valid.

From Contact.java

// First Name

if (firstName == null || firstName.isEmpty()) {

this.firstName = "NULL";

// If first name is longer than 10 characters only use the first

10 characters

} else if (firstName.length() > 10) {

this.firstName = firstName.substring(0, 10);

} else {

this.firstName = firstName;

}

* + 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 tests are designed to fail the input and modify data, if a test fails the test could be poorly designed or the code being tested was poor. I ran into issues with this when deciding on how to create unique ID’s and the length of these generated ID’s were longer with extra symbols in them thanks to the Java UUID Library I used. The coverage of the tests were well enough that I discovered what I needed to modify in my code in order for the tests to operate properly until 36 tests were executed and all of them succeeded. The true test would be running that same code on other operating systems and in other IDEs that may experience errors that would require revisiting the code for optimal efficiency, but that is outside of the scope for what is required of the course.

* 1. 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.

The tests verify functionalities such as adding data with valid and invalid input, deleting and modifying stored data, ensuring ID uniqueness, and checking for the existence of existing IDs. All of this in accordance to the requirements for each section of requested code. For example, when adding an appointment the date cannot be in the past, if it is then the test asserts an invalid date.

From AppointmentServiceTest.java

@Test

@DisplayName("Test invalid appointment date (past date)")

**void** testInvalidAppointmentDatePast() {

// Arrange

String appointmentID = "A123456789";

Date appointmentDate = **new** Date(System.*currentTimeMillis*() –

86400000); // Past date

String description = "Valid Description";

// Act and Assert

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

Appointment(appointmentID, appointmentDate, description))

}

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

The code was efficient by using multiple testing principles of Junit testing. By using JUnit Set Up and Tear Down annotations (See Below) we were able to effectively isolate the tests and run them asynchronously. This also minimizes dependencies between test cases to make sure the code does not depend on the initialization of other tests to run. It also uses Arrange-Act-Assert principles of Junit testing to focus on the independent behavior of the code and keeps it simple and organized.

@BeforeEach

**void** setUp() {

service = **new** AppointmentService();

}

@AfterEach

**void** tearDown() {

service = **null**; // Ensures clean state after each test

}

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

The testing techniques are referred to as “White Box Testing” consisting of Unit Test Cases of each individual component accompanied by testing the components working together in the service they are apart of in the ServiceTests. Boundaries testing was also used to determine if the given input was too long, too short, or invalid through exception handling. Exception handling is crucial to prevent errors from crashing a system and that it may run reliably in the presence of errors. Integration and System Testing are essential aspects of testing larger, more complex projects to ensure the software runs as intended. Unit Testing is the foundation of all testing practices and is the simplest form of ensuring your code works as intended on the smallest scale. Boundary Testing ensures you are validating input, as invalid input could potentially cause a critical system failure

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

Integration Testing and System Testing were not used as part of these testing techniques; these would require testing the components interacting with each other in the software and testing its functionality in the system environment it is hosted. The scope and simplicity of the current software project may not need a System Test phase due to the low amount of resource use and lack of interdependent usage of code in the overall software.

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

Unit tests are the foundation of testing practices, ensuring code works as intended on a small scale. Useful for all software projects to catch bugs early and ensure individual components behave as expected. Boundary Testing validates input and ensures proper error handling. Essential for all projects to identify potential vulnerabilities and ensure software reliability. Integration testing would be beneficial for larger and more complex software projects where multiple components interact with each other. It helps verify that individual units function correctly when integrated and helps identify integration issues early in the development cycle. System testing is essential for ensuring the software behaves as expected in the production environment and meets the needs of end-users. It helps validate system performance, reliability, and scalability, and ensures that all system components work together seamlessly.

* 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 writing code the objective is just to make it run, but testing requires you to look at the smaller details with fine-toothed comb and ensure the code is not only operational, but is compatible and reliable for long term and unintended usage. You have to really assess your critical thinking skills to ensure you have everything covered properly in accordance to standard coding principles and the requirements of the client. I paid close attention to the interactions and dependencies between different components of the code, though it was minimal. Creating these tests helped me understand the potential for error in using UUIDs for method calling, which caused my tests to fail in the Service tests.

* + 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.

Bias is created from experience and opinion, and opinion is based on subjectivity. By measuring solely on objectivity through clear and established goals you can remove your bias from the review. Consistency in coding style was also important so that someone can look at all the code and notice similar approaches that remove greater chances of error on integration. I tested multiple scenarios that would force the code to fail and implemented given feedback to ensure my testing had good coverage. One criticism I initially received was failing to include happy path testing as well as failing path individual test cases, which I promptly fixed and included in future code. There is a lot of possibility for different types of biases when testing your own code. Confirmation bias, familiarity, overconfidence… These are all issues and hurdles we must overcome to be professional, proactive software developers.

* + 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 careful about making quality software is important for software engineers. If we rush through writing or testing code, it can lead to a multitude of problems later on. It's like building a house on a poor foundation, it seems fine at first, but it can cause bigger problems down the road that are more costly to fix. That’s why it's better to take our time and do things right from the beginning. To avoid these problems we must adhere to standard coding practices. We should write our code neatly and make sure it works well by testing it thoroughly. It's also a good idea to get other people to verify our work to catch any mistakes we may have missed. And if we find any problems they should be fixed immediately. By doing so we can make sure our software is efficient and reliable.