CS 320 Summary and Reflections Report

Kennedy Uzoho

Southern New Hampshire University

Table of Contents

Table of Contents………………………………………………………………………………2

1. Summary………………………………………………………………………………….3

1a. Describe your unit testing approach for each of the three features…………………....3

1b. Describe your experience writing Junit tests.…………………………………………5

1. Reflection………………………………………………………………………………….6

2a. Testing Techniques…………………………………………………………………….6

2b. Mindset………………………………………………………………………………...7

**Summary**

**1a. Describe your unit testing approach for each of the three features.**

**Contact Entity** and **Contact Service.**

**To what extent was your approach aligned with the software requirements?**

The `Contact` test and `ContactService` test were designed to cover the key functionalities of the Contact and ContactService classes ensuring that the classes behave as expected under various conditions. For example, the Contact Entity specified requirements for each field type, length, and nullability. The Contact Service Entity also provided specific requirements for the expected function of each service, such as the ability to add, delete, or update an object. These specified requirements made it a lot more possible to create tests for each requirement to ensure that everything works based on the requirements. For example, `firstName` field should not be null, this specification can be validated through testing by passing a null value for name field and observing if the error is caught or not.

**Defend the quality of your JUnit tests.**

The JUnit tests include a variety of test methods that check different constructors and methods of the `Contact` class. This approach ensures that all aspects of the class are thoroughly tested. Each constructor was tested separately to verify that they correctly initialize their object with valid data and handles edge cases, such as null or excessively long input.

Comprehensive Coverage:

**Use of Valid and Invalid Test Data:**

The tests used both valid and invalid data to validate that the `Contact` class handles input correctly. For example, it tests the behavior of methods when given too-long strings or null values, ensuring that the appropriate exceptions are thrown. This approach helps in verifying that the class adheres to constraints and provides useful feedback when the constraints are violated.

**Assertions for Validations:**

The use of assertions (`assertEquals`, `assertNotNull`, `assertThrows`) in the tests helps to validate that the class methods behave as expected. This not only confirms that the correct values are returned or set but also that the system correctly handles errors, such as when an invalid input is provided.

**Incremental Testing:**

The test cases were incrementally built, starting from testing individual constructors to more complex interactions within the class. This ensures that any issues are detected early in the simpler components before moving on to more complex scenarios.

**Automated Testing with JUnit:**

The testing approach uses JUnit, a popular framework for automated testing in Java. This ensures that tests can be run automatically and consistently, leading to more reliable software.

During SDLC, the developers, end-users, and as well any other intended audience of the application will test the software for its intended functionality capabilities. The engineers and product testers will search and find all the errors and bugs in the software by performing testing and preparing the software for its deliverable/deployable stage. Although at this stage, it may not be completely feasible to resolve all the bugs discovered during the testing process, however, the error findings can be used to minimize the number of errors and update software, as necessary. The testing approach that I used to test the three (3) features of the final project (Appointment\_Task\_Contact.jar) was mainly the Unit Testing approach and White box testing. In the development stage, I implemented a detailed instructed test for every individual unit or component of the overall project to test for its corrections, errors, and unit functions instructed tasks. The unit test is called unit because it tests each unit portion of the code

Although there are QA engineers, I strongly believe that unit testing is often implemented at the development stage of SDLC, meaning that developers also play a crucial role in unit testing. As I was testing, I made sure that I included the run/test with the coverage option, which yield detailed information regarding the percentage measurement of the amount of code that was run by unit tests. The unit test and the coverage options run through the entire project, starting from lines, branches, and methods used to write the code. I went further to analyze test results by reading the coverage percentage results and tracing the numbers to each unit (class, branch, methods, or the single lines), doing so helped me explore the protocols, stack tracing, and testing objects as it corresponds to the internal code units (class, branches, methods, and single lines). With the JUnit testing, there are multiple methods to ensure that the software code is technically sound. The JUnit test, tests for functionalities as well as the overall format and structure of the code. I learned a lot about overall programming just by reading and analyzing the tests because the test was run based on units, every part of the code units, runtime analysis, and module functions were revealed without deploying the code. For example, the ContactService.java was meant to have specific requirements like a contact object requiring contact ID, it cannot be longer than 10 characters and must not be null. I implemented those requirements first initially, however, the implementation of the code is not the case with Junit testing, the case is making sure that the code implementation does what it was intended to do and evaluates the environments, functions, and other things depending on the test specification. The code below was implemented to test the structures and functions of their corresponding objects, based on the requirement of the project, the contact ID shall not be longer than 10, if you look closely, you will see that one of the tests in the method (module) is testing for the specified size of contactID, i.e. contactID should not take more than 10 characters. tooLongContactid is testing for contactID function to see if the code will allow contactID to contain more than 10 characters.

**public class ContactServiceTest {**

**protected String contactId, firstNameTest, lastNameTest, phoneNumberTest,**

**addressTest;**

**protected String tooLongContactId, tooLongFirstName, tooLongLastName,**

**tooLongPhoneNumber, tooShortPhoneNumber, tooLongAddress;**

**@BeforeEach**

**void setUp() {**

**contactId = "10293A475F";**

**firstNameTest = "John";**

**lastNameTest = "Smith";**

**phoneNumberTest = "5553331234";**

**addressTest = "1 Audrey Jersey City NJ 07305";**

**tooLongContactId = "112233445566778899";**

**tooLongFirstName = "John Jacob Jingle";**

**tooLongLastName = "-heimer Schmidt";**

**tooLongPhoneNumber = "55512341234";**

**tooShortPhoneNumber = "1234567";**

**tooLongAddress = "1 Audrey Zapp Drive, Jersey City, NJ 07305";**

**1b. Describe your experience writing Junit tests.**

**Reflection**

**2a. Testing Techniques**

It is important to first understand the different types of software testing and know the ones that mostly relate to the developing software to be tested. For this module’s project, I implemented Unit testing! Other software testing technics I ended up not implementing were Usability Testing, Security Mapping, and there are others.

In software testing, all the testing methods are important, and this depends on the integrity of the software in development. I curiously think that the compatibility-software testing methods will have multiple protocols, as they will test and ensure that the software’s functionality is stable and supported across multiple platforms/environments.

**2b. Mindset**

Based on the White Box testing technics that I used in the projects, I personally appreciate the environmental configuration and sweeping and monitoring of the developmental environment for better and deeper development and possibilities of analyzing outside software (someone else’s code). The interrelations of the code I was testing helped me explore more options and other ways of development other than constantly writing code from scratch. The APIs modules are insanely amazing open-source technologies that were designed to aid in developing anything, literally anything.

I do think that testing my own code will inevitably be a biased operation because if I wrote the code, I know exactly what the code will do, so if I have to test my own code then I will just test for functionalities and not be bothered about structures, styles, and behaviors of the code since I already know the code I developed. I am thinking that my only concern will just be to make sure that the code runs and is ready for delivery, unlike having someone else implement a test method and study my code with the test.

Finally, QA itself is a discipline that existed because there is a need for it. Software testing is more than how it sounds! A single defect in a software application can bring down the whole software, so that is why some APIs test methods tried their best to make sure that every aspect of developing software to be tested gets tested and the test can be used as a control version of the software as the APIs testing methods can test and update software without changing the outcome that software.

Reference

Monnie, Craig, R., Mahnoor, Awais, &amp; Jane. (2022, April 3). Types of software testing: Different testing types with details. Software Testing Help. Retrieved April 23, 2022, from https://www.softwaretestinghelp.com/types-of-software-testing/