Makayla Anderson-Tucker

CS-320-X6137

Professor Luo

20 August 2023

Project Two

1. **Summary**

For this assignment, I used the test basis, the project description, and the listed requirements to define test conditions and eventually developed the test cases. Once all the test cases were written, I wrote the application’s code and debugged it in the console. The main reason I did it is because it adds another layer of testing and assurance that the code is functioning properly before unit tests are written (Hambling, 2019). In last week’s reading, Hambling (2019) mentioned that it is good practice for developers to debug their code before passing it off to the testing team.

When I started with the contact service application, I used the requirements listed to create pseudocode for the Contact and Contact Services class attributes and methods. Once all the requirements were met, I started coding the application and debugged it in the console. Here is an example from Contact.java A screen shot of a computer program

Description automatically generated and ContactService.java A screenshot of a computer program

Description automatically generated. To help ensure that only certain type of data was accepted, I used RegEx for the attributes. It was the easiest way I could think of to add extra checking for the correct, valid data and still be efficient. I used it with JavaScript in a personal project a few months ago and figured the same concepts applied. For example,

private static final String ***CONTACT\_ID\_REGEX*** = "^.{1,10}$"; (Contact.java)

ensures that the contact id variable cannot be more than 10 character or null. All of my Junit tests for ContactTest and ContactServiceTest passed:

A screenshot of a computer program

Description automatically generated (ContactServiceTest.java)A screenshot of a computer program

Description automatically generated (ContactTest.java)

I took a different approach to the task service application. By this point, I realized I could draw a UML diagram that displayed an object’s attributes and methods. A close-up of a task

Description automatically generatedA close-up of a task

Description automatically generated. I was able to much more quickly create the application’s code and create the unit tests. I used a lot of the same functions for both applications.

1. Reflection
2. Testing Techniques

For the first milestone, I used a combination of static and black-box testing techniques to review the work product and determine functional and non-functional requirements. For example, one functional requirement is that the contact ID is a unique string that is not longer than 10 chars or null. A corresponding non-functional requirement would be that the system quickly determines whether or not the contact ID variable is valid. Once all the static testing and application code were completed, I wrote unit tests in Junit5 to confirm all units were functioning properly. Unit testing is a form of white-box testing because it focuses on the components and functions (Hambling, 2019).

For the second and third milestones, I employed a lot of the same techniques from the first milestone. However, I started these assignments by drawing UML diagrams of the classes instead of writing pseudocode. It allowed me to utilize use-case testing to ensure functional and non-functional requirements were covered. I also used decision testing (if-else) statements to help set the parameter validation for each variable. For example, I used if-else statements to determine whether an attribute’s value could be set or retrieved. For every milestone, I debugged the application code in the console. While debugging is not the same as testing, there are no clear criteria, it is extremely useful in the testing process (Hambling, 2019).

The following are a few types of testing that I did not use:

* Decision Table Testing: Tests ensure that every possible combination that might occur is tested (Hambling, 2019). It reminds me very much of the matrices in Discrete Math.
* State Transition Testing: Outputs are triggered by different input values. The table records all generated outcomes (Hambling, 2019).
* Acceptance Testing: Determines whether the application’s functionality matches the end users expectations (Hambling, 2019).

1. Mindset

In the first milestone I was particularly cautious because the requirements were asking for a specific result, but didn’t mention how to validate the attributes. For example, the work product mentions that the contact id cannot be longer than 10 chars or null, but does not specify what to do when those conditions are met. I decided to add validation checking to each attribute so that the system would fail the test and also refuse to allow the attribute’s value to be changed until it met the right parameters. Without this extra step, I could have written the program where the test for contact id fails, but the attribute is still changed to the incorrect value.

Not cutting corners is important because small mistakes can cost organizations time, money, and quality (Hambling, 2019). To ensure no corners are being cut, testing should be incorporated from the very beginning. It can be done using a static method like reviewing work product or creating use-cases based on functional and non-functional requirements to limit bias. For example, without use-case testing in the first milestone, it took me much longer to develop the application code and unit tests. I was working from pseudocode which doesn’t always help paint the best picture of functionality.

References

Garcia, B. (2017). *Mastering Software Testing with JUnit 5 : A Comprehensive, Hands-on Guide on Unit Testing Framework for Java Programming Language*. Packt Publishing.

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing : An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning & Development Limited.