CS320 Summary and Reflections Report

Michael Medina

Southern New Hampshire University

Table of Contents

[Table of Contents 2](#_Toc117338013)

[1. Summary 3](#_Toc117338014)

[1a. Describe your unit testing approach for each of the three features. 3](#_Toc117338015)

[1b. Describe your experience writing the JUnit tests. 4](#_Toc117338016)

[2. Reflection 4](#_Toc117338017)

[2a. Testing Techniques 4](#_Toc117338018)

[2b. Mindset 5](#_Toc117338019)

[References 6](#_Toc117338020)

CS320 Summary and Reflections Report

# 1. Summary

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

### Contact Service.

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

The contact service class had three main requirements. It should add contacts with unique Ids, delete contacts based on the contact ID, and it should update the fields of firstName, lastName, number, and Address. I began writing unit test for the first of the requirements which specified that contacts shall have a unique ID. Lines 33-58 of the contact service test file took care of this portion of the testing. In it you can see four different test methods addressing this requirement. The first one tests whether a null argument is entered, the second one prevents reusing an existing contact ID, the third one tests whether the new contact was successfully added to the data base, and finally the fourth one makes sure required fields are not left empty. The next section of unit tests consisted of testing the delete per contact ID functionality. Lines 60-91 of the contact service file contains four methods that I used to test this requirement. These methods tested whether the contact to be deleted had an existing ID and whether the ID was valid and not null. Finally, I attempted to test the update functionality, but after running the test it resulted being uncovered.

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

[Prove how your JUnit tests were effective using specific examples and code coverage statistics for the Contact Service code files.]

The overall coverage of the contact service class was 70.7% which fell short of the required minimum of 80%. This was evident in the coverage analysis because although most of the lines were in green, there were lines 37, 45, 57, 85, 90 were yellow indicating limited coverage, and line 97 which was part of the update requirement was in red indicating no coverage. It seems like the update method was incorrectly implemented and this is probably the main reason why the unit test failed.

### Task Service.

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

The Task Service had the following three requirements: adding tasks with unique IDs, delete tasks per task ID, and being able to update only the Name and Description fields. The testing methods were very similar to those used for the Contact Service class. I began by first testing for the adding of a new task. I used four test methods that check whether the ID didn’t already exist, it check that it was not null or empty, and if it was added successfully. This is seen in lines 33-58 of the contact service test file. Then in lines 60-91 another four methods were used to test the delete requirement. These methods check that the provided task id existed, that it was valid, and it was not null. Also, it checked whether the task id was deleted from the database. Lastly came the testing of the update requirement as seen in 96-98. Here once again I failed to implement the update method properly mainly because I didn’t quite understand how it worked.

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

As indicated by the Junit test the overall coverage of the task class testing was 71.5% which is short of the 80% minimum. The update part of the testing was all in red indicating no coverage while the rest of the testing was mostly green with only some of the lines in yellow. The results indicate that I wrote good Junit tests for the first two requirements but failed to cover the last of the requirements.

### Appointment Service.

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

The testing of the Appointment Service was the simplest of all three because it only had two requirements. The first requirement being that it must be able to add appointment with unique appointment IDs, and the second requirement being that it must be able to delete appointments per appointment ID. I tested the first requirement using three test methods as seen in lines 35-56 of the appointment service test file. The first of the tests methods tested for a null argument, the second tested whether the id didn’t already exist, and the third one check for invalid arguments such as blank entries of the required appointment fields. Lines 58-89 contained four test methods related to delete requirement. These four methods handle the null Id and invalid Id cases, as well as checking whether the appointment ID existed and if it was successfully removed.

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

At 73.5% coverage the Appointment Service test had the most coverage of all three service. But once again I failed to achieve the minimum of 80% coverage. It seems that I failed at testing the verification of an existing Id as well the testing of the delete functionality. This is evident by the yellow in lines 39,47, 55, 83, and 88, and red in line 45 of the appointment service test file.

## 1b. Describe your experience writing the JUnit tests.

### How did you ensure that your code was technically sound?

Since this was my first experience writing Junit tests, I relied a lot on the examples that I was given. I did my best to translate what I learned in the resources provided in each of the modules to the actual assignments. I was challenging because the examples given were very simple compared to the requirements of the milestones. I follow the Junit naming conventions by naming all three of the test classes using the same name of the class with the test word added at the end. I also used Junit annotations such as @Test and @BeforeEach. Additionally, I commented where appropriate and I placed the modules in different folders to keep everything organized.

### How did you ensure that your test code was efficient?

To ensure that my code was efficient, I had to clean up my code often. The use of Java validation annotations in the contact, task, and appointment files greatly reduced the amount of code that I had to write. Understanding how to use annotations such as @Size and @NotBlank help me validate all the requirements with just a few lines of code. By using libraries available in Java, I eliminated the need to use repetitive code and was able to write a functional program with as little as possible lines of code.

# 2. Reflection

## 2a. Testing Techniques

### What were the software testing techniques that you employed in this project?

I used both static and dynamic testing technique to complete project one. Static testing techniques are performed without running the code and they in include requirements and code manual reviews (Hambling, 2019, p.75). I reviewed all the requirements and made sure I understood the scope of each one of them. Then after writing the code to implement the requirements, I also reviewed and eliminated mistakes or unnecessary code. Static testing helped me eliminate bugs early which in turn made it easier to perform dynamic testing (Hambling, 2019, p. 76). On the other hand, dynamic testing is done at code execution and for this part I used Junit as my testing framework (Hambling, 2019, p. 76). I wrote a corresponding test file for each of the objects as well as for the service classes. These were the ContactTest, ContactServiceTest, TaskTest, TaskServiceTest, AppointmentTest, and AppointementServiceTest files. In them I wrote Juniit test that verified that the requirements for each class were being met.

### What are the other software testing techniques that you did not use for this project?

Although I reviewed the requirements, it was assumed that they were defect free so there was no reporting of missing requirements. I did not used experience-based testing techniques such as error guessing, exploratory testing, and check-list based testing (Hambling, 2019, pgs. 145,146) These are techniques performed by more experience testers and were not required to complete this project.

### For each technique you discussed, explain their practical uses and implications for different software projects and situations.

Static testing techniques such as requirements and manual code reviews should always be done before proceeding to dynamic testing. They help set the foundation for the testing process and can help to eliminate and fix many defects early in the process. Dynamic testing frameworks such as Junit provide a great structure for testing software requirements. It helps speed up the testing process because it provides feedback of the coverage so that the developer can easily identify the areas that are lacking coverage.

## 2b. Mindset

### Assess the mindset you adopted working on this project.

Being the first time conducting an actual software test I proceeded with a lot of caution. It was very confusing at first because I did not know how to write test units using the Junit framework. It required me to spend countless hours looking at examples of Junit tests. It was a lot of trial and error. A lot of the Java code was new to me and therefore I had to pay close attention to the complexity and interrelationships in the code. For example, Java validation annotations were new to me, so I had to look their definitions and review some examples to understand how they work and when it is best to use them. Junit annotations such as @BeforeEach and @Test were also new to me so I also had to investigate when to use them and where to place them. Also, the structure of Junit tests, assertions, and parametrization were also new to me so it took me a while to figure them out as well.

### Assess the ways you tried to limit bias in your review of the code.

Again, being my first experience at testing software using Junit, I tried to approach each of the milestones with an open mind. This mindset allowed me to learn from others instead of trying to do things my own old way of doing things. I tried not to be critical of the video and text examples included in the resources as well as in the announcements. Instead, I tried to absorb as much as I could and dedicate enough time to learn new more efficient ways of writing code.

### Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional.

Every software engineer should prioritize quality of software. Placing quality on a second place might save time and money initially but later on as defects begin to arise it can turn out to be more costly or even deadly. Today’s complex software are found in machines used by humans in a daily basis. Software engineers should be aware at all times of the consequences of skipping important software development phases such as testing. The time and effort that it is invested in testing the higher the quality of the product. When all the appropriate testing techniques are applied, the resulting product will be virtually defect-free, customers will be happy, and more importantly there will be very little risk of accidents that may result in injuries or loss of human life.

References

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.

Medina, Michael (2022). Project One. Grand Strand Systems Mobile App

Module Five Milestone Guidelines and Rubric (n.d).

Module Four Milestone Guidelines and Rubric (n.d).

Module Three Milestone Guidelines and Rubric (n.d).