CS320 Summary and Reflections Report

Nicholas A. Franklin

Southern New Hampshire University

Table of Contents

[Table of Contents 2](#_Toc121853840)

[1. Summary 3](#_Toc121853841)

[1a. Describe your unit testing approach for each of the three features. 3](#_Toc121853842)

[1b. Describe your experience writing the JUnit tests. 4](#_Toc121853843)

[2. Reflection 4](#_Toc121853844)

[2a. Testing Techniques 4](#_Toc121853845)

[2b. Mindset 5](#_Toc121853846)

[References 6](#_Toc121853847)

CS320 Summary and Reflections Report

# 1. 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 way I approached testing of the Contact Service is the same as I approached the testing of the other classes. I aligned my testing with the software requirements in a checklist sort of way so that I could mark off that each requirement was added, tested, and working successfully. In order to test the Contact entity, I simply went down, line by line and created tests for each. I first tested the creation of the class itself and then using that, I tested the functionality of the getters and setters. I then went down the line of requirements as far as input restrictions and tested my code that covered those. Tests that attempted to create a Contact object with improper inputs such as name strings that were too long or completely empty were created and made sure that my code threw the proper errors. The same was done with the Contact Service Tests in order to establish that no illegal input could be sent through the program without throwing an error.

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

My Junit tests were effective as they had great code coverage and I made sure that they covered all of the necessary requirements for the Contact and Contact Service classes. I was able to attain 100% code coverage for the Contact class and 96.9% code coverage for my Contact Service class which is proof enough that my JUnit tests were efficient and effective.

### Task Entity and Task Service.

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

As I stated in the section above, I went about testing all of the classes in the same fashion which was in a checklist format and going through each class line by line. This allowed me to ensure that all the requirements and input restrictions were tested, and the desired results were attained. For the Task class test, I first tested the creation of a Task object and the tested all of the setters and getters in order to make sure of their functionality. I then tested attempting to set the parameters with illegal inputs such as too long or null which were required restrictions. These all were successful, giving the desired error. I also tested created a Task object with illegal parameters of all kinds to make sure that it also threw the desired error and those were also successful.

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

My JUnit tests were effective and efficient in the testing of my Task and Task Service classes because they had excellent code coverage percents and made sure to cover all the required functionality and input restrictions. I attained 100% code coverage of my Task class and 95.7% code coverage of my Task Service class which proves that nearly all my code is functioning as expected.

### Appointment Entity and Appointment Service.

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

My approach to the Appointment Service testing was to go down the requirements highlighted in the assignment guide and make sure that they were all working properly but also to try and get as much code coverage as possible. Like the other class cases, I first tested the creation of an Appointment object and then all of the setters and getters with allowed inputs. After those tests, I went down the lines of the class and created tests for as many as possible but specifically making sure to get coverage of the input restriction code. Tests that attempted Appointment creation with illegal inputs such as expired dates or null entries were all attempted and made sure that the correct errors were thrown.

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

My JUnit tests were effective and efficient and I know this because they all worked and I had excellent code coverage. I made sure to test all of the requirements and the test coverage of the Appointment class was 91.9% and the test coverage of the Appointment Service class was 93.8%.

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

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

I made sure that my code was technically sound by ensuring that it was producing the desired output based on the input. In order to test this I created many different tests such as in my TaskTest.java lines 80-85 where I create a Task object and then change one of the variables of the object and then make sure that I receive the correct data when I retrieve that variable from the object. Tests such as this one ensured that my application’s source code was sound and working as intended.

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

To ensure that my test code was efficient, I tried to make my tests simple and easily reusable and with as much code coverage as possible. My test code may have been able to be bundled together in certain cases to reduce the amount of code necessary to test but I wanted to make sure of the coverage and thoroughness of my tests. The creation of tests and testing in general also helped me by pointing me to lines of code that I could get rid of completely such as my default constructors with empty parameters. I had put them in my code originally out of habit but realized that they were unnecessary since the requirements for all three of the classes, Task, Contact, and Appointment, were to not allow any null parameter inputs.

# 2. Reflection

## 2a. Testing Techniques

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

The software testing techniques that I employed were first, that I created a checklist of the system requirements and went down the line creating tests for each using statement testing and coverage which is a white-box testing technique. Statement testing and coverage is basically the creation of tests that target a specific line of code or statement to make sure it is functioning properly. I used this many times to check the setters, getters, class creation and basically all of my codes functionality. Use-case is perhaps the other method I used which is based off the system requirements and is basically if you do this then this should happen. That is the type of testing I used for my input restriction requirements. So, I created tests that would send data to my class that should be unallowed to test whether my code successfully followed my input restrictions.

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

I did not use any black-box testing such as partitions or decision tables which are more broad and tests the specifications of a system but since my system never had a main method implemented, it was only really able to be tested from within using JUnit tests on my internal code. Risk assessment or dependency tests were not used because I was not testing for risks to the system but rather the functionality of the code within the program.

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

The tests I used which are white-box tests such as statement testing and decision testing are used within the system to test specific lines of coding and ensure proper coding. Black-box testing is used for testing the broader functionality of a system and that it meets all of the system requirements and functionality, this is more from a user’s view. The risk assessment or dependency tests are used to check for vulnerabilities in the system and code and are very helpful specifically when creating a system with online capabilities and usage where there is much higher risk to the program from many different sources.

## 2b. Mindset

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

In acting as a software tester, I made sure to employ much caution and to ensure that my code was as thoroughly tested as I could so that I could be confident in its proper functionality. It is very important to appreciate the complexity and interrelationships of the code when testing because although code may function correctly when called alone, it may have a different result when called on by another function or when interacting with other functions and classes. I had to take this in to account when first creating my program because I implemented the input restrictions into the service class instead of the base class. This meant that my restrictions were properly observed when called upon the methods through the service class but when calling upon the base class directly, there were no restrictions put into place for the inputs. This meant I needed to recode the restrictions into the base class and illustrates how code complexity and interrelationships of the code need to be taken into account when testing.

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

On the software developer side, I could imagine that bias might be a concern if I was responsible for testing my code but not intentional bias. Since I wrote the code, some unintentional bias may come up since I know how I intended my code to function as well as any of its shortcomings. A specific example may be of how I tested the Appointment and Appointment Service classes. I was having trouble with the date format because I wasn’t too familiar with the functionality in Java and I kept getting warnings that the old date formatting was not supported. This caused me a bit of a headache when creating the classes and when creating the tests for the code. In the end, because of my frustration with the code, my Appointment and Appointment Service classes ended up having my lowest code coverage percentages meaning they were not quite as thoroughly tested as my other code. Their coverage was still in the 90’s percentage though so I considered it good enough.

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

Regardless of the code in question, cutting corners can be detrimental to a system and should be avoided. Ensuring that code is properly written and tested and all of it’s functionality is working properly is extremely important and is a reflection of a coder’s integrity. But it is also important because any mistakes or bugs could have unforeseen, catastrophic consequences, as has been seen in the news many times and in many different areas. I plan to avoid technical debt by always explicitly planning out my design and code beforehand, making certain to always code deliberately, efficiently, and to the best of my abilities, and by thoroughly testing all my code to make sure of its functionality and to eliminate any errors and mistakes I may have made. I made sure to follow all of this in my project as I made sure there was no way to pass any illegal parameters such as null or too long and that all of my functionality was there and working properly. I also made sure that I tested my code thoroughly which is evident by my nearly 97% total code coverage.

References

Bechtold, S., Stein, C., Rancourt, J. de, Philipp, M., Merdes, M., Link, J., &amp; Brannen, S.

(n.d.). JUnit 5 user guide. Retrieved February 16, 2023, from

<https://junit.org/junit5/docs/current/user-guide/>

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.