Jon Frodin  
Southern New Hampshire University  
CS-320-H2213 Module 7  
December 7, 2020

**Project Two - Summary and Reflections Report**

*Summary*

My unit testing approach in these modules stayed largely the same throughout the different modules. This was a primarily checklist-based testing model that considered the software requirements as criteria for success. This would fall under the category of “methodical approaches” as outlined in *Software Testing* (Hambling, 2015, p. 151). I believe that my approach aligned exactly with the software requirements. This could be based on any of the files included, but I will list one specific test from each of the three features.

The first dealt with the requirement relating to the ContactService.java file. The requirement says “[t]he contact service shall be able to delete contacts per contact ID” (Module Three). This was dealt with in the unit tests for the Contact Service in lines 23-33 (Frodin, 2020)[[1]](#footnote-1). First, the code adds a contact, then it runs the deleteContact method[[2]](#footnote-2), which uses the ID number per the requirements, and finally the method searches the list to confirm that the contact is, indeed, removed[[3]](#footnote-3). The second example of conforming to the requirements comes from the requirement of the Task class: “[t]he task object shall have a required name String field that cannot be longer than 20 characters. The name filed shall not be null” (Module Four). To test this requirement, I used two tests due to the two major conditions. The name cannot be longer than 20 characters was addressed in the test called testNameTooLong[[4]](#footnote-4) and the null requirement was tested in the test called testNameNull[[5]](#footnote-5). These tested to make sure that errors were thrown if a name over 20 characters or a null name were passed into the system.

The final example for my tests and approach aligning with the software requirements comes from the appointment module. This module required the following: “[t]he appointment Date field cannot be in the past. The appointment Date field shall not be null” (Module Five). I addressed this by designing a condition in the date field to throw an exception if it is null or if the date is before the date on which the method is run. The software deals with this in the Appointment class[[6]](#footnote-6) and the test is in the Appointment JUnit test[[7]](#footnote-7) (Frodin, 2020). The tests assert that an exception will be thrown if those conditions are met, and the exceptions are thrown, verifying the software requirement has been met.

The tests I developed were, by any measure, highly effective. There may have been better ways to do some of them, but they did exactly what they were designed to do. The coverage percentages of my files for all 6 of the classes were over 90%. One of them, Task.java, was covered at 100%, showing just how effective these tests were at verifying the functionality required by the client. To ensure that my code was technically sound, I extensively tested every method. I also re-wrote entire methods to make them clearer and more concise. One example of a technically sound bit of code from my tests comes from TaskServiceTest (Frodin, 2020)[[8]](#footnote-8). The test was for a method I wrote to generate unique IDs. This one required me to supply the ArrayList with two objects that had the same ID number to ensure the second was not added. The test returned the expected result of an Invalid Argument Exception when it processed line 107, which called the function to add a Task.

Efficient code is also critical to any software project. To ensure the efficiency of my code, I re-wrote several portions of it after getting it just working. Once I had it working, it helped to tear down some structures and rebuild them better and cleaner. One example of this is the ID generator method in the Appointment Service class (Frodin, 2020)[[9]](#footnote-9). A version of this method was used in all three of the modules, but this was the cleanest and most efficient. Working through the modules helped me to see where I was doing unnecessary work. For example, the same method in the Task Service module, on lines 19-21, assigns a unique ID, but instead of returning it early and ending the function, I went on to update it and then return it. It is simpler in the Appointment Service module to do a return in the if statement and leave the other handling to the else branch.

*Testing Techniques*

The list of software testing techniques I am using comes from two places, the article online by Onur Baskirt and the textbook by Brain Hambling and others. The methods I used were predominantly the checklist method and the use case testing methods. These would fall under the classification of methodical approaches (Hambling, 2015, p. 151). Use case testing involves the “operations that a user/actor performs in the system to achieve a specific purpose” (Baskirt, 2017). Testing based on those operations is how this method accomplishes the goal of extensive testing. Checklist based testing is similar and it involves creating a checklist based on either the user stories (in Agile development) or the software requirements generally and developing tests based on those requirements (Baskirt, 2017).

There are many other software testing techniques that I did not use for this project. Standard-compliant approaches are a good example because those involve industry-specific standards to build the software and ensure compliance via the tests (Hambling, 2015, p. 151). I did not use this because there are no industry standards available for this particular program. I also did not use risk-based testing, which involves assessing the level of risk of a function and testing the highest risk items first (*Ibid*). I did not use this method because there is no particular risk involved with this project. The risk was very general in that the risk of not having any particular piece work would involve the program not meeting the specifications, so a check list approach was more appropriate. A final type that I did not use, though there are many more, was a consultative approach. This type of testing involves using guidance from tech or business experts outside the test team to help develop tests (*Ibid*). No particular expertise was required for this project from outside so this approach would have been unnecessary.

*Mindset*

Caution is important to developing tests for software. The complexity of some programs and the methods within them can become overwhelming if a model is not used that covers those relationships extensively. In my programming and test development, I would say that the extent to which I employed caution was substantial. I carefully went over tests to get the most coverage possible in all of the files. Relationships can be very important to developing tests because methods may work properly when called independently, but methods may be called by other methods, which could cause unexpected behavior. One example in my project of this is the second addApt method from the Appointment Service class (Frodin, 2020)[[10]](#footnote-10). This method, on line 48, calls a different method which generates a unique ID number[[11]](#footnote-11). This method, at first, behaved in a manner that was difficult to track when I was testing, because the tests did not necessarily run in order when they were run. This led me to be more careful and add a method to add an appointment while supplying a manually-created ID number[[12]](#footnote-12).

Bias can be poisonous to just about any undertaking. Bias makes us look for what we already want to see. This can, of course, happen with testing code as well. If I had written my code without knowing that I would then be testing it, I could have used my code exclusively to generate my tests rather than develop my tests to fit the requirements. I tried to limit this in my own testing by sticking exactly to the requirements for the client. One area where I did use my own bias, unfortunately, was the delete test in the Contact Service class[[13]](#footnote-13). In this test, on line 32, I used the method for searching, which was developed specifically for this test, to verify the result. I should have made a way, within the delete function, to run a search for confirmation which would have made the tester’s job easier and been a better piece of code to submit.

Discipline in commitment to quality is something that must be kept throughout the career of any developer. Cutting corners in code can lead to lost time, lost profits, and in certain industries could even lead to lost lives. If you commit to quality at the outset of every project, you will approach building your methods and objects in a different way and be more thorough. You will also test everything you can to ensure that the software works with potential unknowns that could be encountered in the real world. I plan to avoid technical debt when writing code in my career by methodically planning my code before writing it and by testing it until it is working once it is written. One way in which I have already done this on this project was when I was writing my Appointment Service class[[14]](#footnote-14). In earlier modules from this project, I did not always include a loop to verify that a manually-entered object of the class had a unique ID number. This was something I added to ensure that the requirement was met even on manually entered appointments. I plan on taking these little extra steps whenever I see them to keep my technical debt as low as possible.

**References**

Baskirt, Onur (2017). 8 Important Software Testing Techniques. Retrieved December 7, 2020, from https://www.swtestacademy.com/software-testing-techniques/

Frodin, Jon (2020). Project One [Computer software]. Grand Strand Systems.

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition) - 5.4 Test Approaches (Test Strategies). BCS The Chartered Institute for IT. Retrieved from

<https://app.knovel.com/hotlink/pdf/id:kt00UC2K54/software-testing-an-istqb/test-approaches-test>

Module Five Milestone Guidelines and Rubric (n.d.). Retrieved December 7, 2020, from <https://learn.snhu.edu/d2l/le/content/574616/viewContent/10539464/View>

Module Four Milestone Guidelines and Rubric (n.d.). Retrieved December 7, 2020, from <https://learn.snhu.edu/d2l/le/content/574616/viewContent/10539459/View>

Module Three Milestone Guidelines and Rubric (n.d.). Retrieved December 7, 2020, from <https://learn.snhu.edu/d2l/le/content/574616/viewContent/10539456/View>

1. ContactServiceTest.java lines 23-33 [↑](#footnote-ref-1)
2. ContactService.java lines 84-93 [↑](#footnote-ref-2)
3. ContactService.java lines 95-106 [↑](#footnote-ref-3)
4. TaskTest.java lines 34-39 [↑](#footnote-ref-4)
5. TaskTest.java lines 41-46 [↑](#footnote-ref-5)
6. Appointment.java lines 14-16 [↑](#footnote-ref-6)
7. AppointmentTest.java lines 73-93 [↑](#footnote-ref-7)
8. TaskServiceTest.java lines 101-109 [↑](#footnote-ref-8)
9. AppointmentService.java lines 16-33 [↑](#footnote-ref-9)
10. AppointmentService.java lines 47-51 [↑](#footnote-ref-10)
11. AppointmentService.java lines 16-33 [↑](#footnote-ref-11)
12. AppointmentService.java lines 36-44 [↑](#footnote-ref-12)
13. ContactServiceTest.java lines 23-33 [↑](#footnote-ref-13)
14. AppointmentSerive.java lines 36-44 [↑](#footnote-ref-14)