### Southern New Hampshire University

7-2: Project Two

Elliot Collins

### Professor Pettit

### CS-320 – Software Test Automation & QA

17 April 2022

The programming projects throughout the testing and QA course made up the features for an application that contained three main features or functionalities. The use cases for these functionalities were the ability to create objects for appointments, tasks, and contacts. The software requirements for each of these features included very specific object requirements such as what fields each object would need and what data types were required for each. For example, the Appointment feature needed to contain appointment objects which required four specific fields. An appointment needed two string fields, appointment ID and description. As well as two date fields, appointment date and current date. These fields would then help properly describe an appointment field, as well as make the appointment service work correctly.

The features also had specific formatting requirements for each field. Some of these requirements were straightforward, such as a string length parameter, while others required some creativity to be satisfied properly. One example where I had to be creative in meeting a parameter requirement was with the appointment date field. One of the requirements for this field was that it could not be in the past. I’m sure there are several ways to work around this, but my solution was to create an object for the current date which I mentioned previously. I then created an argument so that if the date value of the appointment date was before the current date, it would throw an invalid exception.

The “Service” section of the projects really just involved the use case that is; your data objects interacting with a data structure. The software requirements would have you implement the functionality to add objects to a data structure, look up and delete an object from a data structure, and sometimes look up and update an object in a data structure. An example of this functionality can be found in the Appointment project. The program required functions for adding and deleting appointments. I first constructed an array list to store appoints in, and then created classes for adding and deleting appointments. The add appointment feature would have you fill in the fields from the appointment object in the appointment class. It would then check if the appointment details you entered already exist, to help avoid adding duplicate appointments to the list, adding a level of quality assurance to the feature. If it wasn’t a duplicate, it would then add the appointment to the array list. For the delete and update methods of the different service classes, the program would use the ID field of each object to find the desired object in the data structure before executing the method on that object. For example, in the Contact Service class of the Contact Service feature, you would point to the desired object with the contact ID, and then use setters to update each field within the object.

What made this course and these projects unique to my coding experience so far is that they are true back-end coding practices. There are no interfaces coded into these projects so there is no way to run the project and interact with it like a user in order to test it. That is what made the introduction of JUnit tests so valuable. By implementing JUnit tests to these programs, it becomes possible to test each and every individual piece of functionality without any manual user input. The JUnit tests that I constructed for these programs essentially filled the role of that user-input-style manual testing technique that I was used to.

The JUnit tests would provide input and execute the methods of the programs to help you validate their functionality. As previously mentioned, the object building classes for each program (Appointment.java, Contact.java, Task.java) constructed objects and enforced parameters into their fields. The JUnit tests I built for these functions would first attempt to populate the fields of an object and assert that the object was accurately populated. The next set of tests would then attempt to create objects that intentionally broke each individual parameter. For example, when testing a parameter that limited a string to 10 characters, the test would populate the string with 11 characters. If the program threw an illegal argument exception when a parameter was broken, the test would return that it passed.

Creating JUnit tests for the service classes was a simple matter of verifying that the add, delete, and update methods were functioning correctly. To do this with add methods we would simply construct a new object and then use an “assert equals” before executing the “add” method on the new object. The “assert equals” test would pass if the object was added. For a delete method we would run the same algorithm for adding an object, and then use an “assert equals” before executing the delete function. This also tested that the object ID was able to correctly target the object to execute the method on it. For the update service methods, a new object was added, then a series of “assert equals” tests were run on the update object method for each individual field. This process confirmed that the update method could update every field of the object.

The software testing technique I used would be described as dynamic. Using JUnit testing, I was able to test each piece of functionality individually instead of running them all together with a main function. This type of testing is more efficient and easier to troubleshoot because it points more clearly to where my errors are. I would say that I did not use static testing very much. Since JUnit testing is so practical and convenient, I didn’t feel like I needed to do much manual static testing. I did however take my time with the syntax of my code as I had to write it from scratch.

The main concern on my mind when approaching these projects was the fact that I’d be dealing with a program that had different classes interacting with each other, and I’d have to test the interrelationships of these classes. I was able to test the object classes and make sure that they worked correctly, but the functionality of the service methods is heavily focused on interacting with other classes. Because of this I wasn’t really able to run tests on the service classes individually before testing how them in interaction with the object classes. However, I was able to bugfix and ensure that the code of the service classes was accurate just by manually evaluating the code and looking for issues.

Due to the fact that I was building code from scratch and testing it on my own, I can definitely see that bias could impact my development. There have been many times during my CS degree when I reached out to an instructor saying something like “I don’t know why my code isn’t working, I’ve followed the instructions to a T” and then getting a reply that I missed something so obvious. I think that having testing tools such as JUnit tests to prove to me that the code is passing all of the required tests helps to reduce bias in coding. I think that a great way to limit bias much further would be to have a third party review my code and build or provide tests for it for me.

I think that at the end of the day in software development, quality and thoroughness are more important than efficiency. I think that efficiency is really important, and we should use all the tools we can to increase it, but we should never sacrifice quality to save time. Because of that I think we should implement as many testing steps as we can, we can always go back and remove some once we determine they are overkill. In my programming career I plan to embody this idea and always encourage thorough testing in order to ensure quality.