**CS 320 Project 2**

Alexis Indick

Computer Science Department, Southern New Hampshire University

CS 320: Software Test Automation & QA

Dr. Errol Waithe

December 11th, 2022

After completing Project One, I can confidently say that my approach to the requirements of the project were aligned with them. For example, the requirements for Project One were to make sure that my projects from the previous modules were working and updated if the teacher had asked for changes. I also had to make sure that each requirement was being hit for each separate project like for the ContactService, TaskService, and AppointmentService assignments, I had to check that the contact ID was not longer than 10 characters, null, or updateable. I used Junit tests to check that each criterion was being met.

I would say my Junit tests were pretty accurate. For the ContactService assignment, I had a test coverage of 88.2%. The TaskService assignment I had a test coverage of 94.9% and for the AppointmentService assignment I had a test coverage of 92.7%. Each assignment was at least above 80% which was required for accuracy of Junit tests. My tests were not perfect by any means since they did not achieve 100% coverage but for the most part, they were good enough when it came to the assignments. I think I could have made them more accurate by making more Junit tests and fixing up failures that were left over.

When it came to ensuring my code was technically sound, I made sure I followed the requirements that were asked. I made sure to take each requirement separately and write a test after fulfilling each requirement. For example, in the ContactService class, on lines 28-39, I made sure that the deleteContact function removed a contact on command:

Graphical user interface, text, application

Description automatically generated

For the testing of the deleteContact function that was on lines 43-73:

Text

Description automatically generated with medium confidence

The test was made to test if the contact in the list of contacts was properly deleted, and it was since the test had no errors or failures show up.

For the efficiency part, I made sure that my tests were testing if variables were too long, null, or empty. I also made sure there were no errors or failures especially in the TaskService where I had the highest coverage of 94.9% and there were no errors or failures. To illustrate efficiency, on lines 90-102 of the TaskTest file I tested if the program was picking up properly if there was too long of a name:

A picture containing text

Description automatically generated

Regarding my testing techniques, I used multiple different types of tests within both the black box and white box categories. Black box techniques test for the functional components of a project such as the functionality of the program and if the function work properly like if the program correctly blocks a name too long. The techniques I used within the black box category were equivalence partitioning, boundary value analysis, and decision table. In equivalence partitioning, this technique involves taking a small portion of inputs and categorizing by what can be accepted as an input. For example, with “taskID”, only 10 characters could be accepted, could not be null, or empty, or it is considered invalid input. Boundary value analysis is similar but there is a limit for a value like “taskID” could not be longer than 10 characters, so the boundary is 10. For decision table testing, there is a “rule,” “condition,” and “action.” This is where I had a chance to write out what would happen for each if statement. For example, with the appointment description, I wrote each criterion that needed to be evaluated in the if statement such as if the description was over 50 characters, null, or empty. These were considered the “rules” of the decision table. Then, the “conditions” were whether or not the ifs were true or false and if they were true or false, an “action” would happen. If the “rule” of “appointDesc was over 50 characters” ended up being true as the “condition,” an “action” such as the program proceeding would happen.

I also used some white box techniques for testing. White box testing is where we test the internal components of the program and make sure the input and output flow correctly. I used statement coverage and decision coverage testing. Statement coverage is where every possible condition is tested at least one time. For example, in each assignment given, I made sure I tested each condition that was in the requirements like with the task description and where it could not be over 50 characters, null, or empty. I tested each condition separately and at least once with the Junit tests. For decision coverage, it is like statement coverage, but this is where each branch of the functions is tested at least once. So, for example, I tested the if statement branches each separately with Junit tests and made sure the correct output was given. Something I did not try before were experience-based tests like using checklists. For the checklist testing, I made a list of tasks that needed to be checked off. I found that it really helped with organization and kept me from feeling all over the place.

There were some techniques I did not use for the black box testing such as state transitioning and use case. I did not see a reason to use those black box tests for Project One, but they are very useful for projects that require a lot of different things where diagrams would help immensely. I also did not use a couple of the experience-based tests such as the exploratory test and error guessing. Exploratory tests are more for time pressured projects, like limited time, and I did not feel pressured for time. I tried to use error guessing but I found it quite hard trying to guess what else could go wrong since I completed each part of Project One in the previous modules.

For the practical use of the tests, I have used with black box and white box testing, they imply a lot that they can be used for any project no matter the size or requirements. Black box testing works well with projects that have a clear vision of what the inputs and outputs need to be without knowledge of the code underneath and white box really works best with projects that need input and output flows, usability, and design to be verified. White box also involves having knowledge of the code underneath. Experience-based tests really work mostly for projects that are on a time crunch and require those who have had experience with similar projects. When it comes to choosing what tests to perform, it really depends on the project and the experience of the tester. Some programs may not need or do well with certain tests, but others may really do well with certain tests.

Before this class I did not have any knowledge of what testing involved fully. I had a mindset of “how hard can testing really be?” But, when working on the projects, I realized how hard it can be to test and how important it is to test thoroughly. It is important to employ caution when testing because it can help with testing everything. This is because if we do not test everything to its full extent, there could be fatal consequences and that would mean unhappy customers. For example, for the ContactService program, if I did not completely test if fields were null, empty, or too long, there could be problems with a contact like missing information or inappropriately long names and such. This could cause a lot of issues for the client when using the application and inputting names if there is not a system in place to catch the error.

Bias is hard to delete completely from the mind when reviewing code. We often like to add our own ideals and morals when it comes to writing code, but it is important to stay as unbiased as possible. Especially if a client wants something specific and you do not want to add it due to your own beliefs, this could cause a lot of issues with the customer if you do not fulfill their request. I believe I did a good job keeping bias out since I reminded myself that this was for a client and not for myself. I kept my own opinions to myself and kept the code free of my thoughts which I believe is what helped with eliminating bias altogether.

It is very important to stay disciplined when writing good quality code. Cutting corners is not something a serious software developer should ever do. This is because fatal errors could occur if the code is not written and tested properly. Customers in return will be upset with the work you performed as a developer and may never ask for your work again. As a developer, I will never write or test code that I am not 100% sure about. It is so important to be confident and sure of the code your write and the tests you perform. I will not hand a customer a program I am not confident in, have not tested properly, and if I think it is a hazard to them.