Project Two

Alexander Egelston

Southern New Hampshire University

CS-320: Software Test Automations& QA

Ph. D Ramsey Kraya

My approach to developing the unit tests was aligned to the software requirements by taking into account what each class needed to achieve and ensuring that the intended functionality was covered with the unit tests. There were several classes that required the verification of variable character length. One example of this was the requirement that in the TaskService test I needed to check for the length of the ID, name, and description all were appropriate lengths so I ran tests that checked the lengths of variables of each type in the TaskService class. This type of approach with considering the testing requirements of each class and then writing tests to accommodate the requirements with maximum code coverage was applied in every step.

The tests had very high coverage throughout all three sections (Contact, Task, and Appointment) of the project and given the high amount of coverage I would argue the quality of the JUnit tests was adequate for the intended purposes.

I ensured that the code was technically sound by following excellent examples of JUnit test development on YouTube, one of which was the provided SNHU tutorial video. Through several hours of debugging and rewriting I ensured that the tests were functional such as in the AppointmentServiceTest.Java class where the date entered for the appointment was checked against the calendar class that was imported to the test in lines 32 through 37.

I ensured that the code was efficient by writing the classes and unit tests as modularly as possible. A good example of this is in the Task.Java class the formatted accessors and mutators for the ID, name, and description strings in lines 22 though 44. Then similarly in the TaskTest.Java unit test from lines 11 to 30 I very uniformly wrote the tests to the point where they could be copied and pasted with slight modifications to reapply in other classes or even other programs. Trying to keep code that I write for programs neat, and uniform is how I best display upholding industry best practices.

The software testing technique that I tried to implement as much as possible was conditional coverage. I had a good knowledge base of the program functionality given I had a list of requirements for how the different classes should work so generally the type of testing implemented was white-box. I was able to make tests for each class specific to functions that each class within the project had. The benefits of using white-box testing and specifically conditional coverage is that you get more accurate assurance that the code is performing as the client intended. With every condition being accounted for in the running of the JUnit tests you can see exactly where errors will appear in your program if you have an invalid variable entered at some point. The implications of testing this way are the higher level understanding you need to have as a developer of the program being tested which in turn can significantly raise the projects’ technical debt.

Software techniques that I did not choose to utilize in this project were black-box testing techniques. I did not have a need for wide-spread generalized testing due to my pre-existing knowledge base of the project and its functionality. The benefit of black-box testing is its ease of use and implementation. You don’t need to know much about the program or its intended functions in order to write and apply these JUnit tests. Being able to write these tests quicker allows developers more time to focus on the actual code running the program which will save the company time and money. The implications of these testing techniques lie in their nature with how generalized they can be. If a test is not specific enough then there are going to be things that are missed and subsequently errors that pass through the testing phase.

Throughout this project I encountered many errors that stopped my in my developmental tracks. When it comes to reconfiguring build paths and mistakes made during a projects’ setup taking caution is of the utmost importance. The mindset I had during development was equal parts being cautious of potential ways forward through whatever step in the process I was on as well as being in a rush to complete the project given the time constraints. One specific example of this duality of mindset was when piecing the different assignments together for the final project I realized that two of the assignments had been made with different JUnit versions. Initially I couldn’t understand why there was an error with one of the tests as it had worked the exact same way in its specific assignment the way I had it in the final project. Upon updating the JUnit type in the build path for the project I found there were other errors appearing. Feeling the time go by I was in a rush to get through the roadblock as quickly as possible. Normally I would immediately begin searching through some stackoverflow thread to see if a similar error had been discussed and solved before but instead, I decided to further examine the new errors. I noticed that the new errors were also related to the build path and was then able to make the connection and rewrite the tests using the different version of JUnit tests.

Bias would most definitely be a concern for any developer reviewing their own code because they likely take pride in the programs they write and do not have an outside perspective. That is why I ultimately believe that outside software testers are important in the developmental process. I tried to limit bias in my review of my code by only stating exactly what you can see from testing which is that most of the conditions for the clients’ code were covered by the unit tests and that they function as intended.

Being disciplined in your commitment to quality as a software engineer professional is extremely important. There are many instances you could use as an example but imagine for instance you are a developer at an insurance company and you slack off on testing one particular class that ends up being responsible for a patients’ information not being registered correctly and consequently them being denied coverage for a desperately needed medical procedure that they then go without. It may be a bit of an extreme example, but professionalism and high standards during testing can be vitally important. I intend to uphold industry best practices by only submitting code that I am sure functions as intended and which, if I am responsible for, can be proven with testing. I would like to learn more about and in the future implement black-box testing wherever I can in order to minimize technical debt for clients and employers alike.