Southern New Hampshire University

Module 7-2: Project Two

Monticia Dunn

Tanisha Jacks

CS 320 – Software Test, Automation & QA

12/10/22

As I was developing features for this mobile application, I was careful to ensure that my testing approach aligned to the software requirements that were established. For instance, there were many requirements that set character limits on certain field attributes in the Task, Contact and Appointment classes. In these instances, I would set up some boundary value analysis to check the upper and lower limits of these requirements. The Task class had a description string field that could not be null but also could not contain greater than 50 characters. So, I would set up separate tests to cover those boundary cases of a null value being set or a value longer than 50 characters being set. This was just one example of one testing approach taken, there were several others as well and all of them helped to increase test coverage of the application. By the end, I was able to have over 85% coverage of the entire application from my tests. I think there are ways I could’ve made my tests more efficient and increase coverage, as the goal is always to get as close to 100% coverage as possible. For instance, in some of the classes I failed to check some of the error boundaries, like if and appointment, task or contact is added to the list the list of their respective service and it has the same ID as something else already in the list. There needed to be a check to ensure the appropriate exception was thrown so for now that exception is not covered in any test case.

On top of ensuring my testing approach aligned with the requirements, I wanted to be sure my tests were technically sound as well. The tests were set up in a similar fashion, with the @Test annotation use to denote the method is a test and allow the use of assertions. Along with that I would use different assertion methods to line up with what was expected of each test result like assertThrows for tests checking for an exception being thrown or assertEquals to test that the result is equal to a specific value I was expecting. I also wanted to be sure my code was efficient so I wouldn’t repeat myself too much and still maintain coverage of the appropriate class and unit case. For instance, I would of course have tests to check the getters and setters of each class to make sure those methods are included in test coverage. To more efficiently test each of those getters and setters I would create an instance of the respective object like new Task or new Contact, at the top of the file and reference that instantiated object in each of the getters and setters so I wouldn’t have to instantiate a new object for every getter and setter test. So, for the TaskTest class at the top I initialized:

Task task = new Task("1234567", "John Doe", "Here is  a brief description of a fake person");

Then for each getter and setter I would reference “task” and apply the appropriate method to the same task object each time:

  @Test

  void testGetName() {

    assertEquals("John Doe", task.getName());

  }

  @Test

  void testSetName() {

    task.setName("Jane Doe");

    assertEquals("Jane Doe", task.getName());

  }

To be more specific some of the software testing techniques employed on this project include dynamic testing with a focus on Whitebox functional unit testing. Dynamic testing, as implied by the name, is used to test the dynamic behavior of the software and its dynamic variables. In order to test dynamic behavior and variables, the code must be executed which is a must for functional testing. Originally, I thought I was performing Blackbox testing but Blackbox testing is usually performed by someone who doesn’t know the underlying structure and design of the code that’s being tested. This is the opposite in Whitebox testing where the tester does know the underlying structure of the code. Since I’m in the position of being the developer and tester for this application, I’d say my tests lean more towards the Whitebox testing methodology since my tests can be more specific knowing the base code that I’m testing. More than anything though, I utilized unit testing techniques as the application isn’t very comprehensive just yet. Unit testing focuses on small pieces of code, testing them individually for specific functionality. I couldn’t really utilize Integration, System or Acceptance testing since the classes don’t interact with each other much and the application isn’t in production so end users can’t test. I also didn’t utilize many non-functional testing techniques like performance, recovery, compatibility, security, and usability testing. I feel like the non-functional testing would be of more value for a project that has a lot of moving parts working together.

I tried to keep an analytical mindset while working on this project but also, I needed to employ caution to my testing approach to ensure I’m covering requirements and setting the testing system up for further testing if/when the classes get expanded upon. As mentioned earlier, I would instantiate my objects early in the test files to make the individual tests more efficient. Also, in the base code itself, each service class like the TaskService, ContactService, AppointmentService utilizes a base class like the Task, Contact and Appointment classes respectively. I needed to make sure these service classes would utilize those base classes properly and not have access to things they shouldn’t have access to like having a “setID” method. There was a requirement where the IDs should be unique, non-null and non-updateable. So, I made sure the service class couldn’t even have access to a method that will set the ID. The ID will be set on initialization, and it will check if the ID exists then, if it does then the object won’t be instantiated otherwise the ID is set in stone and can’t be affected otherwise. There’s only so much I can try to eliminate the bias in my code since I’m the one writing it and reviewing it. Mostly I just write small chunks at a time, test to see if it works to the requirement’s expectations and if it doesn't then I will refactor. I try to keep the requirements as a source of truth, so I know what standards I need to meet. Either way, it’s important to remain disciplined in maintaining quality as a software engineer. When you cut corners especially in testing, this could lead to many issues further down the line in the development process or even more devastating if it makes it to production. Letting bugs through can be costly and they only become more expensive to fix the further along in the process they make it without being discovered. So it’s important to be disciplined with the testing process regardless of how early on the project is or how minimal the test seems to add to coverage, as long as it’s helping to add to and/or maintain the quality of the software.

# **References**

Hamilton, T. (2022, November 5). *What is dynamic testing? types, Techniques & Example*. Guru99. Retrieved November 23, 2022, from https://www.guru99.com/dynamic-testing.html