Tim Brady

December 9, 2022

CS-320 – Software Test Automation & QA

Dr. Kraya

**Project Two**

For project one, our task was to create a mobile application. Any application is developed based on a set of requirements that outline the expected behaviors of the application. To ensure that all requirements are met, and that the application is performing as expected, it is always a good idea to test the product. By taking precautions to test early in the development, we can save time and money in correcting any issues that may be found. Because of this, our goal is always to adequately test the application for correct, as well as incorrect, functionality as early in our process as possible. Our approach to testing our product was straightforward. A set of requirements existed for the application, and we created certain scenarios within the code, using JUnit tests, to test that those requirements were behaving as expected.

There were many requirements for the project. All of them, however, could be summarized into two categories. The first category would include the expected behaviors or functions of the application. Examples of these functions would include things like the need to add, remove or update a contact. These functions are what the user is expecting to accomplish while using the contacts portion of the application. We have specific JUnit tests in each of our ContactServiceTest, AppointmentServiceTest and TaskServiceTest files. These tests would use the functions to add, remove or update a record and would then call back to the data to ensure that it was done correctly.

The second category of requirements included validation of input. While performing tasks like adding, removing or updating a contact, we had certain input that we needed to ensure would not be accepted into the system. These scenarios often pertained to the length of the input. For example, a contact’s phone number may only allow up to ten characters. We also had to ensure that the fields were not left blank. To test this, we used a similar approach as testing the functionality. JUnit tests were created in which we tested invalid input to all fields one at a time.

Finally, to round out our JUnit testing, after testing for invalid information, we included one test in each of the ContactTest, AppointmentTest and TaskTest files that would ensure valid information is stored and retrieved. A test scenario was created for each to store a valid value in each of the fields. That information was then called upon to ensure that it returned expected values.

The results of the tests went well but show there is some room for improvement. Our tests within the ContactTest, AppointmentTest and TaskTest files that were validating proper setting and returning of information were at 100%, 100% and 86.5% coverage. This shows that information is being stored and retrieved as expected. ContactServiceTest, AppointmentServiceTest and TaskServiceTest all had room for improvement. With seven out of seven tests passed and a 90% coverage rate, the TaskServiceTest file was the most promising of the three. AppointmentServiceTest, with four of four tests passed and 82.1% coverage, was the second promising. Lastly, the results of the ContactServiceTest file are where the most improvement can be made. Although it passed seven of seven tests with each test being 95% or higher, the coverage rate was only 73.8%. Part of this is due to a main function that has nothing in it which drags the percentage down. More importantly, however, is a missing test for updating a contact. This is a great example of one of the reasons why we run these coverage tests. A test to verify the correct functionality of updating a contact was overlooked and could lead to problems later in development or in the life of the product.

Overall, as you can see, our approach to testing was to use a lot of white-box testing in the form of JUnit tests. Doing this allows an opportunity to test functionality and input validation fairly early in the process using test data and scenarios to valid expected behavior vs actual behavior. This is one of the more popular methods for testing java code. JBehave tests could have also been done as they are like JUnit tests in that we would be writing a story with given data points and expected behaviors of the app. We would then test that scenario within eclipse to ensure that actual behaviors align with expected behaviors. Had the application had an interface we may have tried something like a workflow test to ensure that certain workflows can be met, however, that fell out of scope of this project.

There was a lot that went into this project. It spanned over a few weeks and contained many different pieces. Obviously, a piece was overlooked in the updateContact test which illustrates just how hard it is to keep track of requirements in different situations. It is important that formal documentation of the requirements is kept and referred to as often as needed. While developing each of the files, we referred to the source material often. The requirements were written out somewhat like a checklist which made it easier to track each of them. Steps can also be taken within the code to help ensure requirements are met. Comments as well as naming conventions of things like functions and tests are great ways to be able to quickly identify what exists within the code and what does not. These things also help with the efficiency of coding both for you as well as anybody who may need to work on it later. You can see how our code changed to reflect better practices in these areas throughout the weeks that we worked on this project. Our first file, TaskTest has somewhat generic test names like addTask() while our more recent files include the word “test” at the beginning like in testAddContact(). This made it much easier to see the tests while running coverage. We also started employing a grouping strategy where all tests were grouped together in the same section of the code. This kept all similar pieces together which contributed to a cleaner, more organized feel.

As a developer, there are certain things to consider in testing. It is easy to get caught in the mindset that you write the code a certain way and expect it to work so testing may not seem as important to you. Small errors, however, can have very large impacts. It is crucial to make sure everything is tested as thoroughly as possible to avoid this. Bias is another consideration while testing. Bias in testing and reporting test results needs to be avoided as much as possible. Skewing results, whether intentionally or not, ultimately helps nobody and destroys the integrity of the testing.

There are so many lessons to be learned from the last few weeks. Of course, the different types of testing techniques are important. However, maybe more so is the importance of performing thorough testing. Not every application will have a bug that causes extreme consequences. That is part of the lesson. It is often impossible to tell which little bugs can cause the largest headaches down the road. This highlights the importance of testing as much as possible as early as possible. As a developer, we are the first line, and often the most cost and time efficient method, of ensuring proper testing. We have a duty to ensure we are putting out a product that everyone can be proud of.