Charles French

10/17/2021

CS 320

Project Two

My testing approach for the three features was relatively straightforward – it involved unit testing on every decision branch for each function in each class to ensure correct functionality. As the functions were tailor made to meet requirements, the tests were written to prove this functionality. For instance, for the Appointment object, the requirements stated that it needed to have an ID component that was a string that was not null nor over 10 characters in length. This was accomplished in the code by having the constructor throw an error should it receive an input for that field that did not meet those requirements, and two tests were written to make sure the constructor throws the appropriate error if the field was either null or over 10 characters in length. The tests in this case passed, which proves that the program meets the given requirements for that field.

Overall, I feel that my JUnit tests were well written. For most classes I was able to achieve 100% coverage, meaning that every possible branch was tested. Some classes did not quite reach 100% but were relatively close, usually from things such as the test looking for a change in an object rather than console output that also occurred at that particular branch. That said, though some classes may not have reached 100% coverage, they still passed tests to prove that requirements were met.

I ensured that the code was technically sound in the tests by only changing one attribute between tests. For instance, in testing to see if all fields in the Appointment class constructor were behaving correctly, I would first make a test using “good” data to make sure that it was able to successfully create an Appointment object. Next, a test would see if an error was thrown for having the id field’s string being too long (leaving everything else the same from the first test). Next, it would be checked if an error was thrown for the id field being null. Following this, the other fields would be tested in a similar manner, with the id field being replaced with the valid data from the initial constructor test. This pattern of changing things one part at a time per test would persist through the other tests in the project, systematically testing each possible branch for the classes.

I ensured the code ran efficiently by taking advantage of known data in the testing environment. For instance, in the tests for the task service class, to ensure the correct task object was being updated with a given function, two objects were added to the list. Knowing the position of the object in the list allowed me to access it through it’s index, rather than having to loop through and find it, taking a task that could run on the order of O(n) or O(n2) and instead having it run on the order of O(1), making the test very efficient.

The main testing technique implemented in this project was unit testing. This method tests the given components at the smallest scope, making sure that each function of each class works as intended.

Other techniques that weren’t utilized for the project include integration testing, system testing, and acceptance testing. Integration testing focuses on making sure that components interact together as intended. For example, a later implementation of the appointment service class may interact with an API for a scheduling database. In this case, integration testing would ensure that the class uses the correct API calls and that the database is updated appropriately. System testing involves testing how all of the “new” components would behave with the current “in production” system. An example of this would be if an insurance company already had a billing system in production for its customer service representatives to view customer data, but was implementing a new option to schedule monthly electronic funds transfers from this system. Testing would involve making sure this new component did not affect the regular use of the program while making sure the new features work as intended. Lastly, acceptance testing involves gathering user feedback on a new system to see if new updates have in fact been improvements. Using the previous example of the updated billing system, service reps may be interviewed or asked to fill out a survey on their experience with the update. Though the intention of the feature may have been to provide a more efficient means to accomplish its given task (less mouse clicks, menu surfing, etc.), service reps may have noticed a noticeably longer load time for the system, actually increasing call time versus the older system, showing that the update may not have been worth the effort. This data could then lead the dev team and other leaders towards focusing efforts on optimization on the feature, or to revert the system and work on something completely different altogether.

While integration testing wasn’t used in this project, it is a common tool to use in the industry. In fact, if more “pieces” of the app were made for this project, integration may have then been a good tool to use, however with the given requirements, the three features had no interaction with one another, so integration testing was not needed. Similarly, system testing would be a commonplace method for an established business. However, for a brand new application being developed, what would normally be covered here would be covered in integration testing, since everything is new. Without an “old” system to check how features interact with it, system testing was not needed. Also, acceptance testing at this time was not needed, as there are currently no users for the application. That said, market research could be included under this testing method for new products, as it would be important to see how the target audience would react to the project and see if any improvements may need to be made before release.

In acting as a software tester, employing caution was similar to ensuring the test code was technically sound. It was important to make sure that the code met the requirements. As touched on earlier, one component of a function was changed at a time between each test to show it worked as intended. It was important to make sure that each decision branch was covered in a test to ensure a high percentage of test coverage for the class. By systematically approaching the testing process this way, not only did it help give a roadmap for how to implement the tests for a class, but it also helped to show thoroughly that the code met requirements and worked as intended.

Bias wasn’t as big of an issue as I may have thought it was at the beginning of the course. Given requirements, I needed to see if the code worked as intended. Along the way there were issues with tests unexpectedly failing, but more often than not it found an error in the code that needed correction. Also, there were times where I needed to improve the test to prove functionality. For instance, in the test class for the task service, I needed to show that a given task was removed from a list. Since the task would be removed from memory though, I decided to write the test to capture the console output from the same decision branch as where the object was being removed from memory. The output happens after the object is removed, so if the program reaches that point in the code without error, then the object must have successfully been removed. However, capturing console input proved to need a bit of finesse – that is, knowing the correct whitespace characters used in the output. After a bit of searching online mixed with a bit of head scratching over the issue, I found that the println() function ends a line with the characters “\r” and “\n”. I had expected to need a newline character to match the expected output, but the carriage return character was a bit of a surprise. As such, by avoiding bias and letting the results speak for themselves, the tests helped build better code, and in turn the code helped build better tests. That said, after working through the course and the project in particular, I wouldn’t imagine bias being a real concern should I need to test my own code in the future.

Attention to quality is important in any profession, and software development is no different. Cutting corners in coding or testing will result in poorly received work at best and can cost lives at worst. Many systems vital to keeping society running these days (transportation, emergency services, etc.) often has software behind it. Similar to how a construction company’s work on a building is subject to official inspection, while software may not have a similar governance, we should strive to make sure that our code would be “up to code” in a similar fashion. In order to reduce technical debt, I would make sure to ask questions to clarify gray areas in requirements as soon as they are found, and also correct work along the way when an error is found, rather than “passing it off” to the next person on the team. Should I need to test my own code, I would do like I did on this project to improve both code and test to make sure it not only passes, but that it also meets requirements.