Project Two

Justin Faircloth

My testing approach for the separate projects were very much aligned with the software requirements. Each project was centered around a class with attributes of certain parameters and then the project required the ability to add/delete tasks and update certain attributes. As a result, my tests focused on checking that the projects accomplished those specific tasks in the way they are supposed to. Here is an example of the type of test I created for the Task Service milestone: 

This specific test validated that inserting an ID longer than 10 characters would result in an exception. This fits in with the parameters of the project that ID’s can be no more than 10 characters. Overall, my testing followed this logic: verifying that methods only accept attributes within the specifications and that the code works in accordance with the project guidelines.

When looking at the code coverage for all my milestones, I calculated the average line and branch coverage. My average line coverage was 100% and my average branch coverage was 82%. Since 80% is the industry standard, I believe my code coverage showed that my overall testing was effective.

To ensure that my tests were technically sound, I focused on doing a few things. First, I made my test names very obvious to easily convey what each test did. Some examples are @TestNullID and @TestTooLongDescriiption, among others. I also tried to make very focused tests to lessen the chance of invalid results. I would take care to only test one thing at a time (besides when testing things such as a constructor method). For example, I would use separate tests for every setter and getter method and I would test them multiple times.

To make my tests efficient, I tried to use the same values in each test, besides the value being tested for exceptions, to lessen the chance of false results. Here is an example:



I only used different values on the attributes being directly tested but otherwise used all the same values to have a successful baseline to start from.

The main testing technique I used was unit testing. I took individual methods and “units” of the code and tested them directly to make sure they reacted to different inputs in the right way. The vast majority of these tests also fell under the umbrella of black box testing, as they simply tested the input-output dynamic of the methods.

I did not use white-box testing in my projects, which would’ve required checking the internal workings of each unit. I also didn’t do performance testing since the projects were fairly simple and worked extremely quickly. Due to the simplicity of the project, security testing was also unneeded.

When designing these tests, I did have caution in mind. Even with the rather simple projects we did for the milestones, there are still plenty of ways for your code to mess up and produce unexpected results. I did my best to make sure that each small part of the code was written effectively. For example, even simple getter methods (which I didn’t feel the need to test as I’ll explain later) need to work correctly since I used them in my unit tests to validate results. I used caution in this way to make my overall code more refined and effective.

I did sense a bit of bias in myself as I wrote the tests. Since I wrote the code itself, I knew how it worked and as a result, I sometimes felt the need to not test something because I assumed that it would already work. An example of this is the simple getter methods I mentioned in the last paragraph: I mitigated by doing tests that I thought were unnecessary for the sake of coverage. Just from these projects, I can tell bias would affect me if I were responsible for testing my own code in a professional setting.

This shows that it is important to be disciplined when programming. In a much more advanced and complicated project, I can see how taking the time to do things right and not cut corners will save time in the long run. I plan to avoid technical debt by being as disciplined and thorough as possible. It would also help to seek assistance where necessary to avoid bias or cutting corners.

Sources

Pittet, S. (n.d.). *The different types of testing in software*. Atlassian. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

Ashtari, H. (2022, September 29). *Black box testing vs. white box testing*. Spiceworks Inc. https://www.spiceworks.com/tech/devops/articles/black-box-vs-white-box-testing/