Matthew Trembley

4.17.2022

Southern New Hampshire University

CS-320

Project Two – CS-320

With each of the three features included in Project One; Contact, Task, and Appointment, it was a fantastic experience writing unit tests for each of them. Overall, the three features had very similar facets to them like a unique ID, a description, or name. With that being said, the approach to testing was similar throughout each feature. First and foremost, no attribute to each feature may be null or empty, so testing each of them individually proved helpful in making sure that they were in fact populated with some arbitrary string data. My approach was exactly aligned with the software requirements in a very important way: data length. Below is a code snippet of what I tested for every attribute in the requirements, but to their own respective requirement.

Text

Description automatically generated

For every “getter” within the code I had written, I made sure to test for two things – nullity and character length. Every getter started with an assertion that it was not null, which was a requirement. Then, for the first name for example, I tested to make sure a test name was less than 10 characters, and a purposefully too long was more than 10 characters.

In hindsight, the quality of my tests could have been better, even though the coverage was 89.4% which in my opinion isn’t half bad for someone with little experience. I feel that testing for a string that is more than 10 characters will always pass when the string has more than 10 characters, so it seems somewhat redundant to do that, but at the time of writing it felt right. One thing that it seems I have overlooked is testing for unique ID’s. Although I tried implementing a random ID generator, in none of my tests did I test to make sure that it was actually unique for either a contact, appointment, or task.

Text, chat or text message

Description automatically generated The actual writing of the unit tests was a bit enjoyable, and it constantly gave me a bit of a confidence boost as I wrote unit tests and code. I made sure that my code was technically sound by means of some of my test’s parameters, like pictured below.

I tried to make sure in this test that each parameter passed through the method of addNewTask would properly be input into the Task object, which was successful. Although, being hardcoded into the test it may be easy to fail in the future if the strings did not match which is again another hindsight issue. My attempt at making sure my code was efficient was more-so in the testing of the “getter” methods, like the one pictured below. This snippet of code entered a new Appointment object and tested for nullity, length, and to match to a default description of the Text

Description automatically generatedappointment that was hardcoded.

The most common software testing technique used throughout this project would be white-box testing, which involves knowing and using the internal structure of the code as a basis of tests. In an article found online at career service website Indeed, white box testing is described as “a technique used by developers to evaluate code and the internal structure of software” (“What Is White-Box Testing?”, 2021). From this definition alone, it is clear that white-box testing is the most primary technique used as all the tests written are based on the code for each feature and the structure surrounding it. The next best testing tool, known as a type of static testing, is incorporated with the IDE being used, Eclipse, as it analyzes code while it is being written for any kind of error that may prevent the code from compiling.

A software testing technique I did not use is the opposite of white-box testing, called black-box testing. Black-box testing is defined as “A method of software testing that examines the functionality of an application without peering into its internal structures or workings” (National Insitute of Standards and Technology, n.d.). This wasn’t used at all, as the testing having been done was directly related to the code structure. Another technique that was not implemented in this project was Ad hoc testing. Ad hoc testing is described by a blog post written on guru99.com, a development informational site, as “an informal or unstructured software testing type that aims to break the testing process in order to find possible defects or errors at an early possibly stage” (Hamilton, 2022). Essentially, this is a brute force form of testing that developers who are ultimately familiar with the system can find faults in seemingly random points.

Each of these testing methods have their own strengths. In my opinion, white-box and ad hoc may go somewhat hand in hand. If a tester knows the code well, they can essentially brute force some possible failures through that will create errors and expose possible bugs. Black box testing is a great way to use different inputs without knowing the criteria and seeing if the uninformed input will in some way expose a bug or fault within the code. White-box testing seems to be a great way to test the structure of the code extensively, as the information from the code itself can give parameters for unit tests to be created.

As I was working through this project and each milestone, I took a very mindful and step-by-step approach as a software tester. It was important for me to be cautious as I wrote tests because I didn’t want to have a test fail for something little like a misspelled variable or calling an incorrect method. After I had written the code, I had gone through every method and made sure to write a test, one by one. They started small, with each one starting with an assertion of nullity, making sure each “getter” for example was instantiated and not null or “empty”. It was important to appreciate the complexity and interrelationships of the code I was testing because generally each method may have fed into another one that can cause a failure if one was incorrect. It was like searching for syntax errors, but instead of using the compiler, it was my unit tests that I had to understand.

During my review of the code, I certainly tried my best to limit bias as I wrote tests. As I do not have much experience coding, I was fully aware that my code was/is far from perfect and has room for improvement. In this case, I tried to the best of my abilities to test every angle of my code and get as much coverage as I possibly could. As a developer, I can definitely imagine bias being a concern if I were responsible for testing your own code. For example as I was writing the tests for the project, I didn’t feel like that I needed to test the “getter” methods in each feature. I had the notion that they were sound and even deployable, but as I wrote the tests anyways I had realized that in my code, I had written all my Contact getters to return an empty variable that I put as a placeholder rather than an actual set String. To rectify this, I actually had to rewrite my constructors to correctly assign each variable that the corresponding getter would return.

Being disciplined in a commitment to quality as a software engineering professional is so incredibly important to oneself as a professional, and as an employee for a company. Even free-lance type software engineers must strive for top quality as it only further helps establish them as a successful developer. Also, cutting corners is very detrimental to writing and testing code. If one was to cut corner testing code, like creating a unit test that checks to see if a variable is assigned a value, rather than to see if it is the correct value, then it could go potentially go unseen and end up as a headache of a bug or run-time error further down the road. It would be more expensive to fix that than it would be to correctly write a test for this issue. Cutting corners can be lazy, and more often than not, lazy programmers have the worst code in terms of readability and maintainability.

# References

Indeed Editorial Team. (2021, August 25). *What Is wWhite-Box Testing? Techniques and Examples*. Retrieved from Indeed: https://www.indeed.com/career-advice/career-development/what-is-white-box-testing

National Insitute of Standards and Technology. (n.d.). *Computer Security Resource Center: Glossary*. Retrieved from https://csrc.nist.gov/glossary/term/black\_box\_testing