Project Two

Zachary Derepentigny

6/25/2023

When writing my unit testing for these features, I focused on establishing tests that aligned with the characteristics outlined in the requirements document. Specific examples would include several character limitations that were present throughout the assignments with limits such as 50 characters. In these cases, an appropriate test included testing if the input exceeded the 50-character count limit. For my test, I utilized a check for if the character count exceeded a provided string of 50 characters, so that anything longer than 50 would fail the test. I found throughout the project that I had difficulty writing functional JUnit tests. While I provided test cases that I felt were functional and covered what was intended, I failed to understand how to actually employ JUnit testing. As a result these naturally functionally failed because the JUnit test itself didn’t function.

When constructing my code I followed basic Class structure for each required component of the given feature. For example, in the appointment service Class, I have the two required Methods set up with similar, understandable structure for the required features of adding and deleting appointments. By organizing the code this way, I can keep similarly structured functions grouped together within Classes that make the program easily accessible. Similarly, minimizing waste in code structure allows a greater degree of efficiency, both for the program’s processes, and for any human interactions with the codes such as updates or changes. The code I have utilized is set up in such a way that the Appointment.java is set up to hold the required appointment variables, with checks for the required limitations such as character limits. This set up allows the code to perform its intended functions, fulfilling the requirements, while allowing any additions to be added simply. This makes the code both efficient and technically sound.

To date, I have completed three milestones for the designated classes for the Project One assignment. Each of these milestones involved adding test classes to ensure that the specific requirements for each class were met. Thus far, I have primarily used unit testing to validate each of these class requirements. Unit testing is described as testing small, individual components of code to test single functionalities to ensure they are producing the correct output. This works well for these assignments, as they allow us to test our code in pieces as it is developed, eventually leading up to the larger final submission of the project. There is a myriad of other testing techniques that I have not used for this assignment so far, some of these include integration testing, functional testing, and acceptance testing. Integration, the first of these, involves testing multiple integrated components of the program that interact with each other. Where unit testing would likely test one individual method, integration could involve testing multiple interacting methods under different conditions. Functional testing is used to test how the program handles different inputs, and what the resulting outputs are and whether they are working as intended. Acceptance testing is where the program receives a full functional run-through by one or more entities, a common one being user acceptance testing where end users would perform the test. This phase is essentially a final test of the program to ensure that it not only functions but does so in a desirable way to the testing group selected. Each testing technique is helpful to the overall success of the program at different stages of development, and often throughout development. Unit testing can be utilized throughout development to test individual sections of code as you go to ensure it functions before moving on, integration testing is used to test sections of code integrated with each other, functional testing tests the programs ability to handle different inputs and confirm the correct outputs are given, and finally acceptance testing is the final test of the testing process before the program moves to production, where it is tested by a select entity.

When developing the code for this project, I definitely leaned very cautiously when trying to specifically work on the unit testing portion, as this was the part of the project that I was completely unfamiliar with. I think an amount of caution is definitely a good quality in software testing, however this can also extend the testing process unnecessarily, so a good balance of caution is likely the best to maintain. There is often a lot of complexity and interrelationships between sections of code. Methods and Classes will usually have some kind of interaction and reliability with each other. For example, for this project, the Appointment and Appointment java classes interact with each other to make up the total Appointment functionality. In this case, one Class is enabling the functionalities to add and delete Appointments, and the other is storing the Appointment information needed for the program. From this example, we can see that these two classes and sets of code are working with each other to establish the overall Appointment functionality for the program.

I think bias is certainly something that can negatively impact developers who are in one extreme or the other of either being very confident or very unconfident. I think in my place, I am affected by the negative side of bias, where I tend to assume my code is poor far sooner than I accept if it’s good. When it comes to testing, either side of bias can negatively impact the testing process. What is most important is to identify the proper extent of testing, as well as what functionalities need to be tested, and ensuring those are all covered. Someone who is biased against themselves may perform too much testing, wasting time and resources, whereas someone biased towards themselves could perform too little testing, having a similar consequence. Being disciplined is a great way to avoid one’s own biases from negatively impacting the quality of testing. Being disciplined ensures that all appropriate testing procedures are being followed and corners are not cut. Cutting corners can often lead to errors in the code, which leads to lost time, money, and faith in the quality of the project. Identifying a testing process that is appropriate for the project or organization overseeing the project and following it every time is a good way to avoid issues such as this. By following a set process, you can ensure that you are performing sufficient testing every time and that you have a paper trail to show that you followed due process when testing any code you are in charge of.