Nicholas Wood

CS 320 Project Two Summary and Reflection

The project that was completed involved testing as well as requirements for the overall software. To complete this project many of the requirements must be aligned and the testing must be effective and represent the system well. For my files of contact, task, and appointment each have their own set of requirements. The Contact file has a contact ID, first name, last name, phone number, and an address. The contact ID must not be null and it cannot have a length greater than 10 characters, the first name cannot be null and it must be less than 10 characters, and the last name has the same requirements as the first name. The phone number must have 10 characters otherwise it is not accepted, and the address must not be null, and it must not have more than 30 characters. The task file contains a task ID, name, and description, and each has a requirement that it must meet before being implemented into the system. All the items must not be null, and the task ID cannot exceed 10 characters, the name cannot exceed 20 characters, and the description cannot exceed 50 characters. The appointment file contains an appointment ID, date, and description and each item cannot be null. The appointment ID must not exceed 10 characters, the date must not be in the past, and the description must not exceed 50 characters.

The JUnit testing that is implemented makes sure to test for each of these requirements and this is shown once the JUnit test run and it displays a coverage percentage of 100 percent. This can sometimes be misleading because if I just test one case to make sure it accepts the correct parameters then I would receive a 100 percent coverage but I would not test the system for all the different types of inputs. To make sure we have an effective 100 percent coverage we must test for each type of input and have the correct response from the system.

When we need to make sure that the output is correct for multiple inputs, we must be technically sound in our program. This is shown by the JUnit testing for the contact, task, and appointment files. The contact file has specific requirements for the contact ID, first name, last name, phone number, and address and the JUnit testing must test for every type of scenario that would break the system. This is simply shown by all the different types of testing for the contact items, each item has a test that breaks the system by either being null or by exceeding or not matching the correct amount of characters of that field type. The task and appointment follow this type of testing and to make sure it is technically sound there is a test for each type of scenario that would break the system. To make sure that the code is efficient we are only using one test for each scenario that would break the system, I could try multiple different types of names that pass and I could also try different systems that do not pass but I only need one to pass and I only want to test one portion at a time. Therefore, there is separate testing for each scenario to make sure that it fails for a specified reason and not have multiple fails in the same test.

There are many different types of testing techniques that could be implemented into the system, but I have implemented just the JUnit testing. This type of testing is called regression testing as it covers specific areas and there are many benefits associated with this testing. It allows for a more focused testing and isolation of methods that are used. It also checks for an invalid input as stated earlier and makes sure that any addition or modification does not break the system. It is a very common and useful testing technique to use and it produces results that allow large coverage of the system. I could have implemented a dependency check that would check the entire system for outdated techniques or certain features that have know issues. I have used the dependency checks in other scenarios and they are great to test the overall system, this would not test for each type of input but it does test the system and see if there are any issues with the overall protection of it. I would also say that some of these checks do reveal false positives that do not affect the overall system and they do supply some know resolve if they are applicable. The JUnit testing on the other hand seems to test for scenarios and they must be set up more by the developer compared to the dependency test which doesn’t require as much coding but it does require a more thorough look at the issues.

For working on this project I was focus on meeting the requirements of the system and not too focused on the JUnit testing, I figured if I could implement correct code then I could come back at a later time to focus more on the JUnit testing. When it comes to developing code caution must always be taken, in my first iteration of a few of the files I only tested for a conflict with the amount of characters and I did not include a null parameter as part of the testing. It is very important to make sure that the relationships between the code works correctly as it can be a large issue later if the code is not correctly implemented. I had to make sure that the contact service which interacts with the contact class works correctly, if it was not integrated properly than the testing would also show issues and it wouldn’t have anything to do with the actual testing problems. When it comes to bias in the code I would have to make sure that I am not too attached to the code and even though I created it there are always more efficient and maybe more secured code that would benefit the system even more. Bias can lead to developers not accepting other solutions and when it comes to the testing if the developer is stubborn about their code then they could easily miss some issues with their system. I would always consider a second opinion when it comes to the code as it can make testing much easier and much more efficient. I would also say that just because the code is not completely correct doesn’t mean the developer is not a good developer, it just means the code is not completed flushed out.

It is very important as a software engineer to have quality in your work, your work represents you and how you perform. This doesn’t mean having the correct code all the time because as software engineers we always need more input from others but it does mean that all the code is commented so that it is easy to follow and it is organized neatly and not crammed together too much. It is very important not to cut corners when developing code, in my previous example of not testing for null, this could be a cost saver and a time saver but if someone accidently submits their first name and forgets their last name then it would accept it even though they forgot to add it. Cutting corners may seem beneficial for time and money but in the long run it would require a bigger issue and may end up costing a lot more. To avoid the technical debt, I would definitely focus more on my code and not worry as much about the time it would take, if the product is correct and free of bugs then it leads to a better result down the road. I have also seen products that start out using a simpler concept in the beginning and then down the road the requirements change and the simpler concept is not acceptable at all, it is always better to spend time focusing on the beginning of the project as it will end up costing less through out the project.