**CS 320 7-2 Project Two**

Eric Galtieri

[Eric.Galtieri@snhu.edu](mailto:Eric.Galtieri@snhu.edu)

Southern New Hampshire University

Unit testing is a vital portion of every development cycle, its purpose is to find issues within an application or a system to bring issues to the attention of the developers early in the development process. For these reasons the code that I wrote was made to test every possible outcome of the application with the intention set forward by the client. In an example, the client laid out specific requirements for the contact id needed within contact java, where they directed the id should not be greater than 10 and should not be null. The code within contact java will evaluate the contact id to make sure it’s not breaking these parameters, then to make sure these work the test java I created has a test that will create a new contact with the contact id that is far beyond this limit to force an exception to be thrown, the same was done for the contact id being null. This is the same method I used in all the unit testing I wrote for this project, as these requirements are set by the client limiting the variables to a limit each of the tests, these limits were already preset so the tests are made to verify if the limit is met and if the ability to perform a setter or getter with the variable is possible. For both the appointment and the task java you can clearly see these examples as the test will set the description then use the assertions’ ability to test if the variable is correct if not it will throw false thus letting the developer know there’s an issue. The code being simple it was easy to validate its technically sound as I was able to write the code within another temporary program where I crated a test account information then wrote it to the console if the system could read it variable then ran the testing there to validate every part worked correctly. The efficiency of the program was to make sure it was clean and clearly commented on to ensure that everything was written well and to make sure I hit all parts the client wanted for the project.

The technique I used when writing the code was to write to reach an outcome, where I’m only creating code to meet a specific outcome then writing the test to find any outlining issue outside that outcome. An example of this is for the contact id, I would write in pseudocode what I need such as the client wanted to have a contact id, that was not null, and that was not more than 10. Each step would simply be taken to accommodate the last part then written to meet the needed requirement, then the step would be checked off to verify the step was done. This tactic is best done when the outlined requirements are clear and easy to represent, this can later be built on to, adding the need for an automated id production to remove the need for manual set and possible conflicts of the ids. With this idea in mind the test is easy, if the id must be between one and ten and not empty then simply write a test to test the two extremes. I did not use black box or gray box testing as I did not need to validate or test the user input, or the risks associated with adding the user input information as there is no security risk associated with these at this time. Black box should be used to validate input and output streams such as a validation or authentication software, where gray box would be to test both white and gray box functionality and can incorporate them both to test the user perspective within the system instead of just the user interface.

Mindset, the mindset I adopted was just one of familiarity as I’ve ran hundreds of unit tests in my time as an engineer, many being automated I knew there was not perfect way to write a test and no matter what the code will never be perfect when testing so I only took to making sure that the limits of the requirements were tested out. The tests can be changed and manipulated later on, but the scope should be limited and verified as early as possible to ensure that there are no limit breaking variables set early on. Bias when writing code can be shown when a developer writes with an idea in mind and forgoes any review or critical evaluation of their own code, this can limit the development process and make a developer less talented over time as they don’t see there is always ways to improve, as everyone is guilty of this bias it’s important for us to take a step back and review and critically think about out code and what we can do to make it better. For me I typically forget to write comments and only strive to meet the requirements, because I know I’m prone to doing this I can easily take a step back and correct these issues allowing me to become a more well-rounded developer in the long run as I can change the style and the way I’m writing my code. Code quality and consistency is important to ensuring everyone’s jobs are easier, much of our time is devoted to a lot of difficult tasks and we tend to push ourselves to work faster and to make a product that meet what is needed without thinking if was the best way to perform the task or if was the correct outcome the client wanted, this can cause us to have to rework a lot of tasks and can cause problems latter on. If we as a collective took more time to develop our code and to write cleanly and clearly, we can contribute more even though we are taking longer to write the code we were assigned.