Gabriel Feng

February 20, 2022

Software Test Automation & QA

Module 7 Reflection

I believe that the tests I developed was built off of the requirements for the software, as each test evaluated the functionality of the class that was built in regards to the software requirements. For example, the telephone number for the contact was required to be 10 characters long, and so the tests were set up to ensure that in case the entry was too long, too short, or null the program would report an error. I’m certain that the Task and Contact classes were tested very thoroughly, being covered almost by 100%. However, the Task and Contact Services were less tested as I could not think of what other areas to cover yet feel as though the tests inadequately covered all potential errors. I ensured that my code was technically sound by listing all of the properties at the top of the code, followed by the constructor, then by the get methods and set methods. I also ensured that my code was efficient by breaking the for loop at line 44 in ContactService when the contact is updated.

I used both static and dynamic software tests to ensure these past milestones were fit for submission. For static testing, I reviewed my code by looking through it for any syntactic errors and other flags the debugger had shown. I then ran through the code in my mind to make sure that the code structures would theoretically run correctly. For dynamic testing, I tested for coverage of the requirements by creating automated tests that would ensure the data structures were created properly. Some tests I did not include were peer evaluations, technical reviews, exploratory testing, or state transition testing. Peer evaluations would be helpful as a second eye would help catch errors the original developer might not catch. A technical review of the code that examines the structures and methods was not used as I feel like I lack the experience in coding to make the right call on the best use of each structure beyond just meeting the requirements. Exploratory testing and state transition testing were not used as without a proper user interface these tests cannot be done. Trying to include as many static tests as possible is always good because solving errors before running them is always better efficiency and cost-wise. For dynamic testing, the tests I conducted fit the program that I had developed, and dynamic tests should do just that. As dynamic testing always has a resource constraint, utilizing computing power and time where the code needs to be more rigorously tested would be for the best. An example of this would be testing a video game’s ability to collision detection system, LOD system, and scene transition for an open world game whereas a software application like Photoshop would focus on the usage of tools and explore when the tools would fail the artist.

One way I exercised caution while testing my software was to doubt my own work. Because both the tests and code were developed by myself, I needed to first evaluate if the test was written to test what I wanted and second to consider the test result and where the error could lie in the software. I also had to consider whether there was an error between the usage of the base class and its dedicated service class. This caution also helped to eliminate some bias in the tests. However, since I am still the developer of the program, I cannot eliminate all bias from the tests. As the developer of the code, I would have an inherent bias towards my work. Not only would I be biased due to pride, but because I would be blinded due to my intimacy with the program. Having worked with the code from scratch, it would be more difficult for me to identify the error as I had to have intentionally put everything in the code which would lead to tunnel-visioning. From a professional standpoint, it is important to be disciplined in testing program quality as the program will be for consumer use. If the consumer is dissatisfied with the program or leads to privacy issues, the profit margin of developing the software will fail to meet the expected gains. Catching errors in the program early on will save time and money for the business, which stresses the need to test the quality and functionality of the code throughout the development of the program.