Brandon Stultz

June 16, 2021

CS-320: Software Test Automation

Southern New Hampshire University

**Summary & Reflection**

Summary

Thought out the development process my approach was focused on effective and efficient tests for each feature and their corresponding methods. I will begin by discussing my approach to designing tests for the contact service. My approach was requirement centric, using the requirements for the software as the basis for the tests initial design parameters. The requirements provided specified that variables for a contact cannot be null nor can they exceed a specific character limit. These requirements served as the test parameters that would be used to determine the pass or fail values for each variable and corresponding method. Examples of this can be seen in the ContactTest class. Each variable had two corresponding tests that checked whether the variable exceeded the character limit and/or had a null value. Lines 9-22 of my ContactTest class showcase how the requirements document directly influenced the design conditions that were tested to verify proper source code design. When designing tests for the service portion of the feature I still maintained a focus on testing the programs’ ability to function as the requirements document specified; no more, no less. Lines 28-32 in my ContactService test display this well by presenting a test the seeks to confirm that existing contacts can be accessed and updated using a unique key. Each of my tests takes into account every variable, data structure, and method that is utilized in the functionality of the feature. This informs me that my code was efficient seeing as the tests cover over 80% of the source code.

Writing these tests was both insightful and productive from my perspective. The attempt to ensure my code was technically sound provided engaging and thought provoking. Lines 10-13 from my ContactService class show case this effort nicely. This test was designed to test both my ability to add new contacts but also to test the data structure that would store them. The data structure served as the core element to the functionality of the feature by allowing information to saved and accessed. To ensure efficiency in my code I sought to design a system that was functional and quick, which I accomplished by using methods and variables inside a HashMap structure. Lines 9-12 of my ContactService class show this efficiency. It does this by creating a new object and attaching that object to a unique identifier key which can later through its memory location rather than a need to traverse memory.

I could continue explain each approach and experience with each feature we designed but have chosen to use the above examples to represent my experience. This is due to the fact each feature I utilized the same approach. I examined the design requirements, determined the necessary variables, Determine the methods necessary for the system to function as specified, create a HashMap to store variables using a key, then use the methods to access the HashMap. The test design process also followed the same approach for each feature due to my approach. Starting with testing each variable for the two constraints followed by testing the methods for their ability to create, update, and delete instances of the class. Further explanations on each feature would be redundant and counterproductive. Review of my source code will reveal this to be true.

Reflection

Throughout this project I found I utilized only portions of the many techniques performed when designing tests. All the techniques I used would fall under functional testing. These include unit, integration, system t, and acceptance testing. Unit testing helped ensure each individual component of the system was operating correctly reducing the likelihood of defects arising later in development due to component flaws. This was followed by integration testing which tested the compatibility between cooperation components, verifying that things such as methods operated correctly with the other portions of the system. The system testing after the integration testing verified that the system operated properly, testing that the variables, methods, and data structures behaved together. The final test was the acceptance test, this test verified that the system returns the proper information given input that fit the design requirements.

However, their exist other techniques that we did not touch on in the project. These make up non-functional testing that include performance, security, usability, and compatibility testing. Performance testing determines whether a program is quick and efficient at performing its tasks. This was unnecessary in our project due to its low computational demand. Such a test can however be useful in projects that deal with more demanding computational operations. Security testing was another test not used throughout the project. This type of testing is obviously crucial on projects that deal with sensitive information or with programs that are publicly accessible. The usability testing seeks to test how easy a program is to use by average users. Our project had no user interface making this testing void. Although, such testing is invaluable when developing projects that are intended for a broad public audience whose skill sets very widely. Lastly, compatibility testing, these tests determine if a program operates correctly on different operating distributions. Our project obviously just focused on distributing this program to windows thus making the need for such testing unnecessary.

The development process for this project was one of caution early later turning into productivity. In the beginning I attempted to make a conscious effort to design something simple but effective. I personally believe simple solutions to problems are often the most effective answer that cause the least issues later. This caution served as the core development tenant as I progressed in designing each feature. The HashMap’s provided the simple solution. The data structure enabled me to link unique keys to entire objects. Allowing for quick and easy access to the information. Originally, I had considered other structures that could only work with extra complexity added on to make them work such as linked lists. Reducing the complexity made designing tests much easier and transparent. Tests are only as good as your ability to understand the functionality of what is to be tested. A more complex system may have components that are hard to judge their functionality at first glance which may result in designing tests that don’t effectively gauge the effectiveness of the program.

Testing my own code does have its cons however in hindsight. One such con is bias. It is easy to judge code less harshly when you are the one who wrote it. When designing tests its crucial you have a critical eye to finding flaws, not just validation the current code. I can imagine myself spending countless hours designing a system than designing tests for that system and instead of properly designing tests to find flaws and weaknesses in the code base, I just generate tests to validate what I already am aware works. Therefore, discipline is crucial in software development. The more dedication you put into designing eloquent and effective code, the less defects that will be inherited into the system. Not only making your own job easier but also those who must test your system. Following best practices for development is a good start in developing better code. I personally like to plan how I will develop a project before I start so that I get a better picture of what exactly I am attempting to achieve before I start designing. This helps reduce redundant and useless code, as well as minimizes error. Do more with less is my motto.