Andrew Rapolevich

CS-320

27 April 2024

Dr. Phillips

Project Two

1. Summary

A close-up of a computer screen

Description automatically generatedWhile developing the unit testing for the three features being contact service, appointment service, and task service, I focused on achieving strict and outlined software requirements. For these projects the software requirements needed input validation for user input and for us as developers to ensure the input code, if incorrect, would throw exceptions or if correct would be accepted. An example of this is the following code snippet:

This code demonstrates a test case where we are checking if the task added is being added and not resulting in error. We have the following test that will test if a user enters a duplicate of the current tasks, it will pass an error to the user. These JUnit test cases ensure we are achieving software requirements like adding tasks with appropriate steps to screen inputs.

The overall quality of my JUnit tests achieved what was outlined in the software requirements but could always gain more strength and traction with more testing and validation. For the requirements given however, I would believe that the JUnit tests I created fulfilled their purpose and achieved efficient coverage. When reflecting on my code I ensured at least 80% coverage, giving me a strong justification for proceeding with the next task. JUnit tests and coverage essentially establish a good understanding of what code you have tests for. As you develop methods and create tests for the specific actions, coverage will increase. For example, if I have ten lines of code and only six are being tested, I will have below satisfactory coverage if not all methods are being tested. At the conclusion of my software testing, I had over 80 percent coverage which provided the validation I desired.

A close-up of a computer code

Description automatically generatedWhile I developed my code, I created a list of parameters that would check the inputs of all strings a user would supposedly attempt to input. Determining if an exception was thrown, I would input an incorrect parameter into the test. If the test completed with an executed error when I threw it error parameters, I would write a comment it passed an error test case. If the test concluded and failed, I would have to re-attempt my strategy to get a passing case. Validation with these results had been as simple as inputting a proper string that would throw an error. If the test failed to execute, I would continuously tweak parameters until I determined I had written a proper test case to match my code. An example of this is the above snippet displaying a “Pass” tag to ensure I tracked test executions. While coding the JUnit tests I focused on having efficient and the desired software requirements. I only tested what was required and requested from Grand Strand Systems to ensure I had been not only efficient in code but for my time. An example of writing code that felt efficient and effective is as follows:

This test case had extremely specific objectives such as adding the task, updating the task, and validating the new parameters that were updated. This met perfectly with the software requirements desired and outlined how in a short span of code we could test our methods efficiently.

1. Reflection

Software testing techniques I employed during this project that made the most impact would be static testing and unit testing. Static testing is where the software is tested for errors before the utilization of code execution. We essentially can test and attempt to find errors without executing the code we are running. The opposite of this technique is dynamic testing where code is run and debugged. Unit testing or specifically our JUnit tests achieve individual unit testing by validating the input of a user to ensure data put into the system follows specific guidelines. These tests are an attempt to test the functionality of code written and help the developer understand if methods created are executed correctly. Coverage goes along with this to ensure that we have test cases accounting for most if not all our methods. These techniques ensure as a developer we can provide some layer of testing before dynamic testing.

A couple of techniques that we didn’t employ would be system testing and security testing. System testing is the integration of code into a system to establish its role in a project. When developers are working on a project like this, it’s traditionally split into multiple segments that are worked on by different teams. We brought the program together between the three milestones, but it had yet to be integrated into any type of system. The practical use of this technique is to identify its performance in a system setting where it’s being actively utilized as part of a larger project. As for security testing, modern code must understand the ramifications of security vulnerabilities and the implications when they are ignored or untested. Because we are providing this program to a company it’s imperative that we do everything in our power to mitigate security faults at the development level. A practical use for this form of testing is ensuring we are not leaving input variables that can leave us vulnerable to cross site scripting or other malicious activity.

The mindset that I developed over the course of these milestones and major projects is that it’s our objective as developers to keep up with current software testing techniques and procedures. Without software testing we would have code published and distributed that could contain omissions in security or could result in catastrophic failure. When I initially started developing my code, I did not anticipate such an in-depth testing system to be available within Eclipse and I started to gain confidence in my code as I progressed through the course. However, there were still instances where I wanted to be as certain as possible that my code would provide coverage and vetting to the system I had been testing. As for biases within my code, I attempted to limit the connection I developed with my code and went through multiple iterations attempting to make it more generic. My code could only get better by being reviewed or evaluated by others and often that’s what I would want when developing a full-scale project. It’s important to not cut corners in multiple areas of code development because bug hunting or errors can be catastrophic for a system that’s about to be deployed. Errors that can be solved and addressed in the earlier stages of development can be fixed in a timely manner rather than when sections start to come together, and dynamic testing is taking place. Being disciplined within software development ensures that you do not cut corners and jeopardize the integrity of the code you’re developing. As I progress as a developer, I seek to continuously get better and have a greater understanding of the code testing environments.