Kalin Mason

CS 320 Software Testing and QA

June 26, 2021

Final Project

Testing could be arguably one of the most important phases of the SDLC. Testing needs to be methodical and executed well to ensure requirements and regulations are met. Testing is essential in finding errors or bugs but also testing those corrections to ensure the new fix won’t cause a new bug somewhere else. It’s also important to know that the deliverable is up to the clients standards.

There are quite a few variations of testing techniques. Two that stand out are static and dynamic testing. Static testing is testing that is done without running any code; whereas dynamic testing does execute the code. Static testing often leads to lower costs in testing since the code isn’t ran yet. Static testing is used to find early errors in code and syntax or logic errors. While dynamic testing is used to ensure the code is operating as intended.

One testing method that was used for my tests was boundary testing. Boundary testing is when we have minimum values and maximum values and we test values within this range. If a user’s input was either below the minimum or above the maximum, an error would be thrown. This type of testing was used in all testing cases and would ensure nothing was null or above a maximum value.

Coverage percentage from the JUnit tests was new to me and it can be used as a measure of how effective your test cases were. With all the classes and .java files made, I created tests that involved the requirements. For instance, with contact.java there was a first name max length of 10 characters. The unit test that was ran would ensure the inputted name was not more than 10 characters but also not null. If the name was over 10 characters it would fail and throw an error.

Text

Description automatically generated with medium confidence

When I would test this method with my name (Kalin) it would not throw an error as I’m within the 10 character limit, but when tested with “tooLongFirstName = "Sir John The Third";” this would properly throw the error the first name input is too long.

Graphical user interface, text, application, email

Description automatically generated

Another example would be with the appointment class and scheduling appointments for users. A constraint was that we could not schedule an appointment for anything in the past. This was achieved by comparing the suggested date with today’s date and if the suggested date was lower (in the past), it was not permitted.

Text

Description automatically generated

One method that I used during this class was regression testing. Regression testing is intended to save time and is useful for this intention. When a new change is made to the code, it should not negatively impact the code base and other portions of the software. When I was developing my tests and would change a method or class, I would ensure it would do so not disturbing my tests I’ve written.

Another technique I learned from this class was boundary testing. This method of testing was to test the extreme points of a method. If there was a minimum and maximum limit of the method, then a test would be designed to test under the minimum, between the constraints, and above the maximum. This method of testing was found in all of my test files and important when an input had a maximum number of characters allowed.

I’ve come to learn that code simplicity affords developers to navigate the code base with little to no hassle. If a code base is over complex, more issues tend to arise. If the code isn’t well documented or even documented poorly, this too can cause issues downstream. This leads to the test cases need to be thorough and looking at the major components of the code base. Within this project, my code was looking at the user’s input to ensure it was fitting into the application properly and not outside an extreme range. I think one of the most important things I’ve learned from this class was to identify vulnerabilities may lie and how to test them adequately.

I attempted to limit my own bias and that the code met the requirements. I read each requirement thoroughly and implemented it to how the documentation was listed for these components. I thought of a few “nice to haves” but didn’t want to pass them off to the customer as they didn’t align with the initial plan. I think that removing myself from the customer perspective did help to a degree to stay focused on what was being asked. For instance, with the input values of “first name”, “last name”, and “phone number”, I wanted to ensure that they were only accepting the intended values and nothing further than that. With the boundaries and not allowing null inputs, I was able to stay on track with what the customer had asked from me to create.

It’s important to remain disciplined with your commitments to producing quality software to minimize downstream costs or issues. I think it’s very important to create code that is sound in logic and requirement to deliver comprehensive software to the customer and to not lose their trust. If trust is lost then it’s hard to continue creating software to customers who will not buy it. My own plan in the future is to create software that is to the customers requirements and not include unnecessary add-ons unless asked. I think another way to ensure that corners aren’t cut is to create software that will limit the number of bad inputs or even bad interactions between the software and the user. This can be achieved by using constraints like I did in the updateFirstName method and not allowing for null inputs or over a specific number of characters.