Tristan Maloy

CS 320

2/23/2025

CS 320 Project Two

**Summary**

Software requirements need to be followed to produce a properly functioning software. Definition of the requirements within the software was the first thing I made sure to add, such as creating if statements that throw an IllegalArgumentException if the variable being used was outside of the expected parameters:

A screen shot of a computer

AI-generated content may be incorrect.

If any of the variables did not comply, the software will throw an exception saying the input was invalid. If all variables comply with the requirements, then the current object would be updated with the variable.

All my JUnit tests achieved over 90% coverage which means that over 90% of my code’s statements, branches, conditions, and functions have been tested and meet the software requirements that were defined.







Testing of the mutators and accessors as well as the constructor was the most important to ensure that the variables followed the correct data flow and threw the correct exceptions when applicable. Certain things such as testing whether a Boolean was false in a test when it was set locally as false seemed redundant, so I made sure the test whether or not the function properly made the Boolean true when applicable. It was areas like this that I chose not to put as much focus on so I could ensure the rest worked properly and couldn’t cause a defect.

Continuously testing and altering the JUnit tests to test for all branches is one way that I made sure that my code was technically sound. For example, when testing for the functions that deleted contacts/assignments/tasks, I made sure to recall the add function more than once, then used assertEquals() to test for the size of the list that was getting added to ensure that multiple were being added as was expected. Then, once I deleted what was to be deleted, I utilized a Boolean that would turn true if the ID was equal to the item in the list that was deleted. The test would then only succeed if the Boolean was returned false. In addition to using best practices like this, I also utilized assertThrows() to ensure that the correct exceptions were being thrown with each individual variable. This allowed for a more maintainable code as I was able to then use assertEquals() to compare the exception to the expected exception.

A screenshot of a computer program

AI-generated content may be incorrect.

In addition to being technically sound, efficiency was also important in my code. The example above would be considered efficient as it was able to take a bad constructor and compare the expected exception with the actual exception. If the JUnit test were to fail, then I would have been made aware that something within my test or the code was not working correctly. Another example of my code’s efficiency was when I tested adding, deleting, and the mutators, I cleared the respective list after every single test so that way each test could be done in a clean environment.

A black background with text

AI-generated content may be incorrect.

**Reflection**

In project one, and throughout the previous three modules, I utilized static testing techniques. After I was done coding each class, I manually inspected the code for errors such as typos, or technical errors such as incorrect passing of variables (i.e. passing a false Boolean when a true Boolean should be passed). JUnit testing was then coded and performed, which needed its own manual testing done to ensure the tests were doing what they were intended to do. The JUnit testing then allowed me to test for expected outputs vs actual outputs as well as input validation. Functional testing was also done to ensure that all requirements were being met. Each class had it's own specific requirements that needed to be met so it was essential to make sure the program was meeting all of the requirements. These forms of testing are essential for software development projects as it helps to ensure a high quality project that is less likely to have a defect.

None of the tests were automatic, as the JUnit testing was all done manually. Automated testing would help to speed up the process of testing which would allow for more time to focus on other forms of testing or another area in the SDLC. Integration testing was also not performed, however there was not an overall program that tested or combined all 6 classes so it was not necessary. Security testing was also not performed, which should always be a requirement in software development to ensure data can’t be leaked or accessed by those that are not permitted. Security testing can also prevent vulnerabilities and threats from external sources.

As a software tester, my mindset primarily focused on trying to break the program to produce results that were not expected so that way I could alter the base code to help prevent any unexpected output. For example, in the Appointment class, this was the first time I had to use the java.util.Date package so I was not familiar with how to ensure that the date was not before the current date. Once I figured it out, testing was simple as all I had to do was set a local variable equal to a date in the past and then use assertEquals() to compare the error message that would be thrown.

A screen shot of a computer code

AI-generated content may be incorrect.

Trying to reduce bias in my code was rather easy, as I know I am not a professional at coding at all. I looked at every function and expected to find something that I can improve on. What was interesting is the further into the previous modules we got, I was able to adapt better practices which I ultimately ended up using in the final project in all six classes. There is plenty of room for improvement, so I was open to finding ways to make the code more readable and efficient. In the example above with the appointmentDate variable test, I originally just tested to see if an exception was thrown, but in the final submission, I found out how to test for a specific exception to ensure that the correct exception was being thrown and not an incorrect exception such as “Invalid Description” when the exception should say “Invalid Date”.

As a software developer, my commitment to quality will be one of the most important aspects of the job. Allowing a program to be released when it has defects, or failures is a sign that the coding quality is not where it needs to be in a professional environment. Quality assurance is key in helping to produce better time and cost savings, improving user satisfaction, while also providing an increase in company reputation, and organizational ethos (Pham, 2025). Quality assurance is essential to integrate into the SDLC and should be utilized in all aspects of software development from gathering requirements, to testing and all steps leading up to and including final release.

**Resources:**

Pham, A. (2025, January 8). *Why quality assurance in software engineering is important?*. SavvycomSoftware. https://savvycomsoftware.com/blog/quality-assurance-in-software-engineering/