So far, throughout the 5 modules of this semester so far, I’ve been focusing primarily on using static testing & unit testing. These methods fall underneath the umbrella of white box testing. Static testing is the process of reviewing a codebase, comparing it against the specifics that are used to identify potential bugs or deviations. I used this quite a few times when my Junit test failed, as I was able to backtrack and attempt to rectify some of the issues the software has highlighted for me.

An example would be my week four module when writing the code for Task Service & it’s testing. I followed the same steps when I made my class for contact service yet was still met with errors. Walking back my steps, I encountered some new errors that I had difficulty diagnosing. On several occasions, a JUnit test failure prompted me to inspect the static code, leading to the discovery of issues such as incorrect attribute usage within a logic branch. Addressing these issues ensured the code met the required standards. This led to my successful tests during this week’s assignment. In addition to this, I’m starting to pick up that I need to verify & read deeper into each exception thrown instead of attempting a quick fix option found in eclipse. This seemed to cause more issues than solutions, as I would end up with a few different identifiers. The exceptions showed bad constructors, but each having a different output based on how the system interpreted it. Like a false positive, it prompted me to reexamine my methods in week five. With greater understanding of the exceptions, I have been able to improve the reliability of my testing.

Module 5 also had presented an opportunity for integration testing. Very little system testing was used & we seemed to only touch the concepts of integration testing in weeks prior. Integration evaluates the entire application by combining all systems. In eclipse, I could have loaded my other assignments (TaskService, AppointmentService), and tried to run my testing to see if they would function with one another. I assume this is the upcoming weeks assignment (foreshadowing). There hasn’t been any pressing of trying to use automated testing. This is more accepted in smaller projects, but a large project would implement this to speed up efficiency. Automation of these test into a continuous cycle such as testing between new build rollouts, would prove most beneficial.

Security testing was unnecessary. Certain libraries & their components like Spring & Java, require additional scanning for vulnerabilities. My projects did not have need for extensive use of external libraries, databases, or additional shared components. Therefore, there was no need for security testing at this stage.

Some applications play a crucial role in software development as they have direct influence over the reliability, security, & the overall quality of applications. Depending on the project’s complexity & requirements, different testing methods will offer selective advantages. I had to carefully consider which tests to implement during my development process, primarily based on ease of use & the scope of work for the project.

Automated testing is useful for large-scale applications, especially those that utilize continuous. Integrations & delivery practices. By automating tests at different stages—during the build process, pre-deployment, or post-deployment—developers can ensure consistent quality and detect issues early. In my projects, I attempted to emulate these methods, normally running code after each end block to see what exceptions would be sent back. This allowed me to save the integration testing for the full build to confirm that different components work together.

Unit testing is an essential practice in any software project, as it helps identify minor errors before they escalate into significant problems. By testing individual functions and logic paths, unit tests improve code reliability, catch overlooked conditions such as null values, and enforce constraints like maximum string lengths. This will not only improve the functionality of the software but also ensures compliance with the project’s specifications.

Ultimately, the choice of testing techniques depends on the nature of project. So far, I feel that unit & static testing have been sufficient to ensure functionality vs stability. Although, as the project complexity increases, I may need to develop more advanced techniques to stay efficient in my testing. Thoughtful testing will not only improve my software’s quality but also improve the end user-experience.