EVALUATION AND REFLECTIONS

Lately I will evaluate and reflect on the topics I have used while solving the Test projects.

Team collaboration

I could not attend teaching and work in a group because of illness. Therefore, I worked alone and not utilizing scrum or agile principles, and any branch. I read the assignment description thoroughly and I try to meet the requirements for the assignments.

Test automation strategy

Today, Test Automation is considered the most effective way to improve the coverage, efficiency and effectiveness of any software application. It redefines how the engineers perform test operations.

To find the right automation tool for a project, I need to understand the project requirements thoroughly and Identify the most important criteria suitable for a project for make projects effective and efficient.
I use Java Server Pages (JSP) and servlets because I have to work with web page and we have worked a lot in school, and I used Integration Test and Mock test and Cucumber

I used Test-driven development (TDD) which I writing the tests before implementing the code and used Selenium for assignment 6, selenium is a free, open source tool available to test web applications and websites. Selenium is made precisely for the purpose of improving the test automation strategy.

Non-functional requirements

Non-functional requirements are requirements that specify criteria that can be used to judge the operation of a system rather than specific behaviors. This type of need is considered as a constraint on the service or performance that the system provides. They include time constraints, process constraints, or development standards.

Non functional requirements (NFRs) describe system behaviors, attributes and constraints, and they can fall under multiple categories.
System performance, security, failover, capacity, scalability, usability, and reliability are just a few categories. I could use BLAZEMETER, but did not get to implement it.

Testability

Testability is about two things. Firstly, can you actually test the software, and secondly, how likely is it that your tests will reveal any bugs. While writing the methods, I had observability in mind to make it possible to see where it went wrong if the system crashed.

In our case we did this by writing some “system.print”-statements in the code that tell me how far we have come in the code. This allows me to easily observe how far the runner has come in the code.

Testability is one of the most important factor that generally gets overlooked in the design and development phase of software production. Development practises like TDD actually helps in early testing and test-driven developments but whether all possible aspects of testing are considered practically is a question. By ensuring that every class had a constructor I reduced the amount of dependencies.

The tests should meet the criteria of being in the “smallness”. That means at reduce the amount of lines of code as much as possible, so there is less to test. When it comes to the actual tests, I also want them to have a "singularity", which means that the instantiations used should be as little as possible, which also means that the test should focus on one function at a time.

Test design techniques

Test Design is creating a set of inputs for given software that will provide a set of expected outputs. The idea is to ensure that the system is working good enough and it can be released with as few problems as possible for the average user.

In this case I use Black-Box techniques in the form of Equivalence Partitioning (EP) and Boundary Value Analysis. This has proven to be extremely advantageous. It achieve a very high-test coverage and structuring the test cases.

Furthermore, we are used to test the internal structure and code of the software solution

Structure-based / white box techniques in the form of unit testing and test coverage measurement. I got many benefits from this, For example. The test case could be easily automated.

Unit testing proved to be extremely advantageous. Unit testing built-in tests that allowed us to make changes easier. In other words, unit testing facilitated secure refactoring.

In addition, unit testing improved the quality of the code. Unit testing also helped us identify everyone

defect that may have occurred before the code was passed for integration testing.

Furthermore, unit testing helped me find errors early in the process.