**7-2 Project:**

**Project Two**

Abdul-Aziz M. Jubilee

Southern New Hampshire University

CS-320: Software Test Automation & QA

Prof. Angelo Luo

Aug 17, 2023

**Describe your unit testing approach for each of the three features.**

**To what extent was your approach aligned to the software requirements?**

As the requirements weren’t *too* intensive, I was able to attain 100% coverage with my tests for the given software requirements.

**Defend the overall quality of your JUnit tests. In other words, how do you know your  
 JUnit tests were effective based on the coverage percentage?**

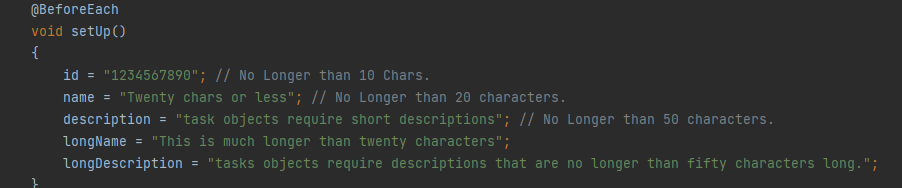
Given the requirements, my JUnit tests met 100% coverage. Non-null, variable length constraints, updatability and lack thereof, etc. Repeated tests were used for the generation of IDs to ensure they were unique.

**Describe your experience writing the JUnit tests.**

**How did you ensure your code was technically sound? Cite specific lines of code**

**from your test to illustrate.**

Essentially, what I did was set up variables with all the given constraints removing the need to type the restrictions into each test and instact just utilize the variables. This modularity also allows for quick alterations should the requirements change in the future. Showcased below as well as lines 15-19 of TaskServiceTest:



**How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

Along with the above excerpt of code, I also utilized repeated tests. Particularly in the case of creating a unique user ID. An example of this would be lines 126-139 of TaskTestService showcased below:



**Reflection**

**What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.**

I mostly used unit testing for my project. This was the simplest way, in my eyes, to ensure that all given requirements were met. The basis of unit testing is breaking the project down into basic units and testing them individually before moving on to integration testing.

**What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.**

There are three other testing techniques not utilized during my project:  
*Integration testing*, where tests are done to see how each component works together as well as within the pre-existing system.

*System testing*, where tests are done on the complete system to ensure the requirements are met thoroughly.

And lastly*, regression testing*, where tests are done after alterations are made to ensure the changes didn’t cause any new unexpected bugs to arise.

**For each of the techniques you discussed, explain the practical uses and implications for   
 different software development projects and situations.**

Integration testing’s practical use is using it to test how module units of a given project function together (or with the pre-existing system it’s being integrated with. It’s done to find and fix bugs early to save both time and money.

System testing is mostly done to make sure software requirements are met, but as It’s most often done from the perspective of an end user, it’s also useful for improving the user experience.

Regression testing is used to ensure changes made to pre-existing code doesn’t introduce new bugs in the existing software/functions.

I’m unsure if you’d like to hear about Unit testing as well, but basically it’s just used to modularly ensure each individual piece of the puzzle is functional and meets requirements.

**Mindset**

**Caution:** Truthfully, I didn’t bother using much caution when doing this project. The requirements were fairly simple and I knew that the only result of me messing up is a slightly worse grade. I’d say it’s important to respect the complexity of the code because complexity can definitely lead to over-testing or even more bugs that could slip through the cracks. These are true as well when more interrelationships are introduced.

**Bias:**  The code in general was fairly simple which made making sure the accuracy and lack of biases in the review fairly easy. More complex code bases could introduce things such as ego and bias in the review which could lead to flaws being hidden and not thoroughly checked. I personally made this flaw a few times due to not really thinking I’d run into any problems with my code.

**Discipline:** Discipline in your commitment to quality as a SWE is basically paramount to being successful in the profession. It provides the end users with a better experience and helps better the reputation with clients not to mention saving time and money of having to re-deliver when sub-par work is sent out. Cutting corners often leads to defects that opens up both severe security risks and UX issues.