CS 320 Module 4 Journal

Joseph Eaton

Southern New Hampshire University

CS-320

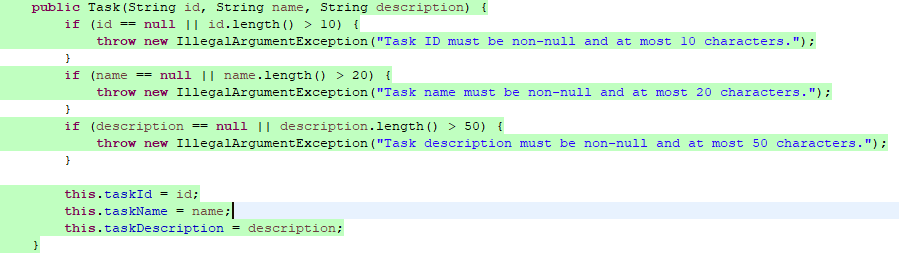
February 16, 2025

**Summary**

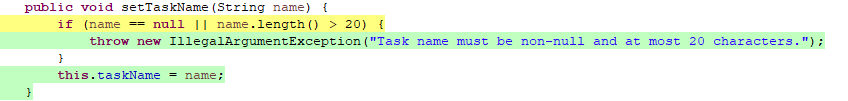
When looking at the testing approach I used and how it was aligned to the software requirements I started by looking over the rubric to allow myself to separate the requirements into smaller tasks that allowed me to manage them more effectively I need to ensure that I do not miss any requirements. I expanded onto this back looking at specific feedback that was given through different milestones of the project to go back and ensure the mobile application that was being made was meeting the requirements by incorporating the given feedback and adjusting my work accordingly.

When looking at the overall quality of my JUnit Tests for the project I would say my JUnit tests were effective could still see some improvement since typically a high coverage on the Junit tests would be anywhere from 80% to 100% and I came in a little short at about 75% overall coverage meaning that while the coverd a good amount of the programs they still could be areas of improvement to ensure that a majority of the code is covered through our Junit tests. To fix this for future references, I would go back over areas not covered through the Junit tests and try to form some tests for those specific areas.

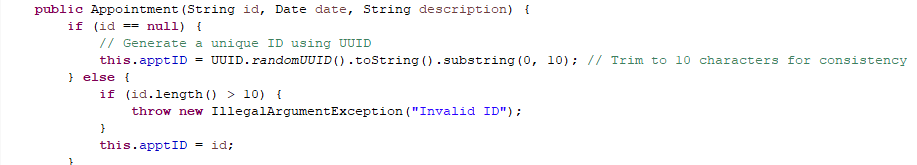
In terms of how I ensured my code was technically sound I would have to cite some of the lines of code where validation was the main concern for example

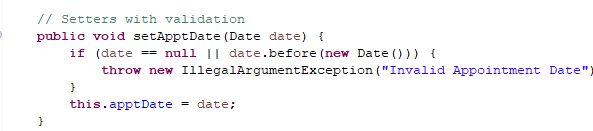
  
Where I am validating input via the setter to ensure that no invalid inputs are going to be assigned to the given fields.

Along with that, when information was modified, I also ensured that any modifications were made to follow the restrictions set in the setters.

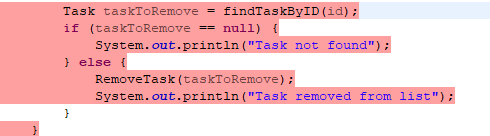


Validation was a big focus through all the program files for example in Application.java there was more input validation included in the constructor with:

  
Having input validation involved throughout all three areas of the project allowed me to ensure requirements were met while making the code technically sound.

All three of these can be relevant examples of efficiency in the code as well as they are both efficient ways of input validation but some more examples of I how I made my code more efficient would include the code taking advantage of short-circuiting and minimal condition checks for example: 

This takes advantage of the OR operator and deems if date == null then the second condition will not be evaluated which would allow for computation time to be saved.

In this code block above shows another thing that was used throughout the project that would be throwing early exceptions that would reduce unwanted object creation by validation an input early. Another block of code that shows effiency would be:   
This allows the search of \_tasklist to be performed just once versus multiple times since we are using findTaskByID(id). Which was important throughout the project any time something needed to be found.

Some of the testing that I used through each of the milestones was unit testing which was used to test specific parts of the code or certain units. This was done by following the software requirements and testing the specified requirements to ensure the program was meeting its requirements.

Some other testing that can happen but has not been used yet throughout the milestones include integration testing, system testing and regression testing. Starting with integration testing it is used to make sure a whole section of code is working versus small fragments like with unit testing it will work through each test moving on and grouping the next part in until the whole section is considered working and tested. Regression testing is used when you add new code to an existing project to ensure the functionality of the new code and to ensure it does not negativity impact and existing code And system testing is used to test the integrated system to ensure it meets what the client is looking for its main focus is on the functionality versus the inner working of the code.

Taking a deeper look at the practical uses and implications of the code mentioned above, we can look at how Unit Testing in practical use is good for early-stage development to verify each individual piece is functional. The implications of using Unit testing consist of catching bugs early and ensuring code reliability. With integration testing the practical use is that every piece of the project is functional after they are combined. It also helps with interface issues around different components, for example how a data service may interact with a user interface. Some implications of this technique include identifying flaws with API integration issues which may not be caught by a unit test. It is also useful in microservices architecture where many different services will need to interact with each other. Now with regression testing its Practical use is to make sure when modifications are made that it does not break the whole program. The implications of this include maintaining stability within the software through various updates and changes. Lastly for System testing the practical use of this type of testing is to ensure that our client's needs are met. It focuses on the usage and functionality of the program versus the individual pieces of the code. The implications of this test are done typically by a QA team and helps with the real-world simulation of the program to see where the program needs to be changed to improve usability.

The mindset I took on while working on this project would have to be striving to deliver a finished product that met all the requirements. As I worked through different pieces of this project, I would try to get as close to the requirements as possible but would fall short due to missing a vital piece like making the ID unique which I missed in all three milestones. For this reason it is important to look and respect the overall complexity of the code and develop tests that ensure you are hitting those requirements which I aimed to approve on in terms of the finished product which I felt that I strived further to ensure all requirements were met in my best ability.

When taking a review of my code the main way I tried to limit bias was to look at the project's requirements and take into deep consideration if my work was truly meeting the requirements. I could see when testing my own code where bias could prove to be an issue especially considering you want to believe that you are turning in your best code and that your tests are sufficient. I way to take a step away from this and break the bias is looking at the coverage percentages along with the requirements to ensure that the work that is getting reviewed and submitted is not only good in my mind but in a non-bias way as well.

It is important to maintain a level of discipline when it comes to quality as a software engineering professional as you are working as a professional to deliver your best work to ensure that the end-user and client are all happy with the work you are making. For that reason, cutting corners would simply be showing that you as a software engineer does not care about the end-user meaning you can let bugs that ruin that experience through or can generally harm the usability of the finished product since you wanted to make something a little easier for yourself. If you keep your mind on the client's requirements and run different types of tests throughout the project, we can avoid a technical debt as we are finding the problems earlier into development and not having to go through at the end and spend alot more time making changes.