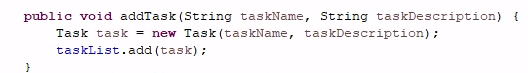
ShiAnn Oliver

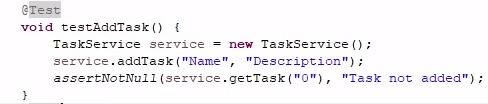
CS-320 Project Two

04/17/2024

Because the software requirements were clearly stated before the project began. This helped me follow all necessary program features and properly structure my testing functions to ensure all requirements were met and were working appropriately. For example, the task service class required that “The task service shall be able to add tasks with a unique ID.” That requirement was met with this code:

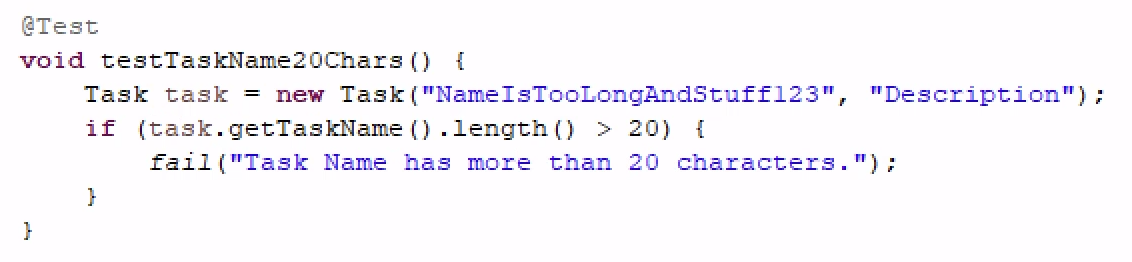


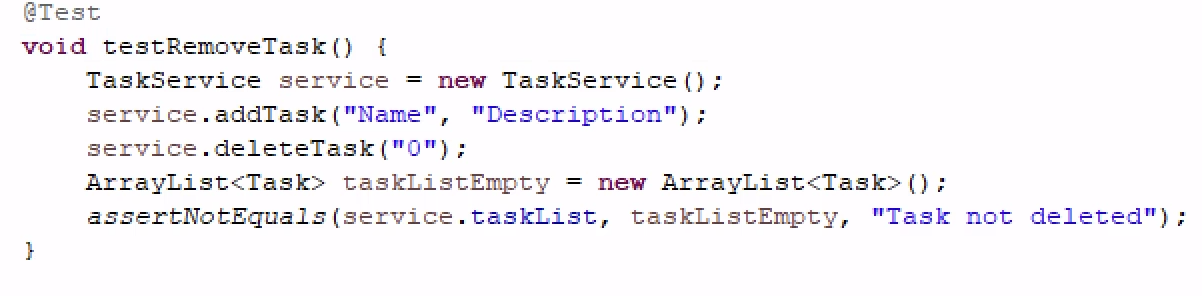
And tested with this code:



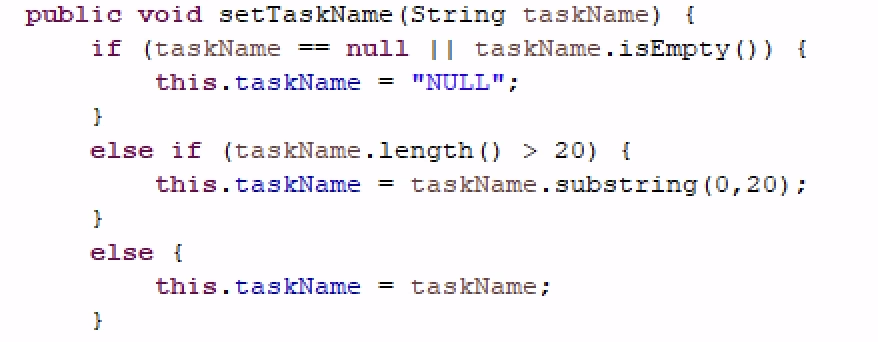
The JUnit tests showed no failures in the Contact, Appointment, or Task classes, and showed over 80 coverage in the ContactService, AppointmentService, and TaskService classes. This shows that the JUnit tests were effective in identifying most code quality issues. Each test will either complete with or without failure, but a lack of failed tests does not guarantee the overall quality of the JUnit tests. It must have high coverage percentage as well in order to have trust in the program quality.

It is important to make sure your code is practical, scalable, and effective at meeting requirements. Code must be easy to read and well commented to make sure anybody reviewing the program can do so effectively. If the software is being tested by a different developer than wrote the program, then it is especially important that the code has comments explaining each item to help others understand. I personally made sure my code was technically sound by ensuring that each requirement was met and using the built in JUnit functionality that allows for efficient tests and error handling. An example of technically sound applications of JUnit test functions:





I ensured my code was efficient by keeping each function created simple and repeatable. Each function I created also served purposes in addition to completing the task it needed to complete. For example, having a constructor that efficiently validates data or the following example where the function sets a task name while validating data so it will fail if the task name is invalid per the requirements.



In this project, I used component, or unit, testing to test each block of code against the requirements of the program. Component testing is composed of individual blocks of code in a system that are tested in isolation (Khan, 2023). This helps each function in the code works as intended by isolating the potential problems. This helps ensure the usability and reliability of the program. Some software testing techniques I did not use were usability and integration testing. These types of testing did not apply to this project. Usability testing, or user experience testing, is mainly used to ensure the end users can effectively use the program and ensure their experience meets the project's needs. I did not use this testing technique because the program did not have a User Interface yet and could not be tested this way. This is an especially important part of testing any program as the end user is usually the customer and determines its success. This is often achieved by conducting user acceptance testing to agree on the project's outcome. Integration testing occurs after successful unit testing to ensure the separate pieces work together once integrated. This form of testing is important for programs that are complex and have multiple layers of functionality that communicate with each other (BairesDev, 2023).

My mindset for this was primarily to be cautious and mindful of the requirements established for the project. I wanted to ensure the requirements were met and my code made sense. I also spent some time researching and finding resources to help employ the best possible solutions to each part of the project. One of the items I had to research extensively to find the best solution to was how to generate an ID code for each package in the project. I ended up testing several different methods including manual and automatic generation and weighing the drawbacks of each. I also wrote different JUnit tests to determine which made the most sense for the project.

I think bias is definitely a problem when it comes to code. I am sure when people are super deep into creating a program, that they may have the belief that it couldn't have issues, or they may miss the bigger picture of how the code works. If someone else tests your code, they can step back and look at it objectively. I think I had an issue with this, as I missed some aspects of testing my code and could've done a better job if I had been able to step back and look at it with a different perspective. I think the ability to look at a problem when coding from multiple perspectives truly just comes with experience. There are multiple ways to achieve each requirement, and some may be better than others.

Discipline as a developer is important because code must be well documented, efficient, and well executed. The quality of code can easily suffer if you are not disciplined and can cause bugs or a poor end user experience. Cutting corners when testing code can cause issues to slip by and be published into the production environment. This can decrease trust in the team and end up being a costly oversight. Making sure testing is completed from an experienced and objective developer and fully executed with attention to detail will prevent bugs, lower overall costs, and ensure project success.

Works Cited

Khan, S. (2023, December 9). *Component testing tutorial: a comprehensive guide with examples and best practices*. <https://www.lambdatest.com/learning-hub/component-testing>

*Java Integration testing explained with examples | BairesDev*. (2023, September 8). BairesDev. <https://www.bairesdev.com/blog/java-integration-testing/>