Dipesh Patel

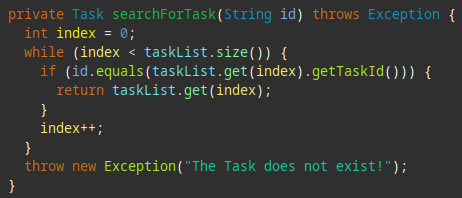
CS 320

April 16, 2022

Project Two

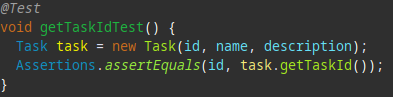
The testing approach is fully aligned to the software requirements due to the fact that the approach tests the required methods in the main code base. With the Contact service, the categories of the id, first name, and last name had a limitation of being no greater than ten characters long each. For each category a unit test was implemented to ensure that any input over ten characters would throw an invalid argument exception. The appointment service had a requirement in which the date could not be in the past. A unit test was implemented to ensure that the appointment could not be set to before the current day. The task service required a method to add new tasks with an unique ID. Using the unique ID utility, I had Java generate a new ID for new tasks and a unit test was created to ensure that the tasks could be created with a unique ID and as well as other data. After many iterations, and trial and errors, I was able the have a coverage of 86.0 percent. The higher percentage of coverage showed how much the tests covered of the functions.

To ensure my JUnit test code was technically sound I created code that was modular, so it could be quickly changed and isolated for more testing or greater coverage. I also made sure the code was syntactically correct, so that the test would function as intended. Furthermore, I utilized the most appropriate data structure for the code I was testing for.



In this example, using a while loop, the code compared the index value with the size of the task list and looks for the ID given to equal to the task list ID for the given index, if not found then an exception is thrown. By utilizing appropriate data structure and syntax the code is able to find the ID of the task if it exists, and if it does not exist then it gives an exception stating as much.

To keep the code efficient I created code that was straight to the point and filled the requirements with as few lines of code as possible.



As seen in this example, a new task is created with ID, name, and description and then is tested to determine whether the IDs of the task given match the ID returned. All this is just done with two lines of code.

The software testing techniques I employed for this project was white box testing. White box testing allows for inspection and verification of the code, infrastructure, and integrations with external systems. Since we know the requirements and what to check for white box testing is ideal for this project. The tests that were implemented checked for and ensured that the output was the same as we were expecting. If it didn’t result in the expected output then a exception was thrown stating as much.

The other software testing technique that I did not use for this project was black box testing. Black box testing is used when there is no prior knowledge of the internal working of the system being tested. Since this project was being developed and test by myself I didn’t use the black box testing technique.

Black box testing is closer to how a consumer would use the product. The consumer has no idea of the inner working of the product and neither does the tester. The tester can test the product as the consumer would use it including any edge cases that may come up. White box testing on the other had requires knowledge of the inner working of the product and specific testing to make sure each component works as intended. This can be automated and can have a complete code coverage. White box testing is used during development while black box testing is typically used after the product is completed.

When working on this project I adopted the growth mindset. Since I am fairly new to testing and coding in general the growth mindset allowed me to learn more and kept me motivated. Caution was used to make sure I didn’t stray off the requirements and to make sure I was thinking ahead about the functionality of the program. Usually you’re working on code one piece at a time so stepping back and appreciating the complexity and interrelationship of the code allows you to see how far you have come and to see the bigger picture of the code you’re working on. For example, the Contact.java set up all the data variables while the ContactService.java used those variable to make a complete data set. By stepping back and appreciating the complexity and interrelationship of the code you see how those two classes are related and how they work together.

To limit my bias I tried to think of ways to test and break my code. I tried to change my way of thinking from developer to tester when writing the unit test to avoid bias. The reason I tried to change my way of thinking was, so I didn’t write test that would conform to the code being tested, in other words write tests that would manipulate the data to give me the results I wanted to see. Pride can be a dangerous thing when it comes to development of code, and can to lead to unwillingness to change code that needs to be fixed to pass the test.

Discipline in development in quality code as a software engineer is greatly important. First of all if you don’t have the discipline to develop not only your code but also yourself, you’re not going to make it very far in your career or life in general. Cutting corners while developing code and tests is detrimental to all involved, from the company that you’re developing for but also the end users. Cutting corners also hurts the development process as someone has to review and fix all those cut corners later in the process, costing a great deal of time and money. I plan on avoiding technical debt by constantly learning the latest techniques and technologies, so I can develop with modern techniques with the latest development and security information.