Nate Bennett

Jonathan Norman

Software Test Automation And QA

17 June 2023

7-2 Project Two Submission

The unit tests for ‘TaskService’ concentrate on adding, deleting, and modifying task names and descriptions. To ensure that tasks are added correctly, deleted tasks are removed, and task name/description adjustments are reflected, positive testing is employed. When handling situations like adding duplicate tasks or invalid input arguments, negative testing is used. The tests verify the functionality of the ‘TaskService’ methods and how they relate to the HashMap data structure that they interface with. Adding, removing, and retrieving contacts are often tested operations in ‘ContactService’ unit tests. Positive testing would demonstrate that contacts are correctly added, retrieved, and deleted. Negative testing would involve dealing with situations like adding duplicate contacts or trying to locate contacts that aren't there. The tests would make sure that the ‘ContactService’ functions behaved as expected and that the contact storing was reliable. For ‘AppointmentService’, the unit tests would include retrieving, removing, and adding appointments. Positive testing would confirm that appointments are added and retrieved appropriately and that deleted appointments are removed effectively. Negative testing could entail dealing with situations like adding incompatible appointments or attempting to locate nonexistent appointments. The evaluations would guarantee the accuracy and dependability of the ‘AppointmentService’ methods and the management of appointment data. I think that my JUnit tests were effective because I believe that my testing coverage percentage was over eighty percent as outlined in the project one requirements.

I utilized JUnit tests and made assertions to attempt and validate the intended behavior of the methods in an effort to make sure the code is technically sound. For an example in my code, by creating a task, adding it to the task service, and then claiming that the newly added task is the same as ‘(‘Assertions.assertEquals(task, taskService.getTask("T1"))’)’, I tested the ‘addTask’ function in the ‘TaskServiceTest class’. This test should assist in validating that tasks are appropriately added to the service via the ‘addTask’ function. I concentrated on the effectiveness of the fundamental processes to ensure code efficiency. Before adding a task, the ‘addTask’ method in the ‘TaskService’ class verifies that it is not null and that the tasks map does not already contain a task with the same ID ‘(if (task!= null &&!tasks.containsKey(taskId)))’. The efficiency of the ‘addTask’ operation is increased because this check makes sure that duplicate tasks are not created. Similar to how tasks.remove(taskId) gives a quick way to delete tasks by their ID, the ‘deleteTask’ method also removes the task directly from the map.

Unit testing, which focuses on testing distinct modules or components, was the testing technique used in this project. It enhances code quality, assists with future revisions, and aids in early bug detection. Other testing methods such as system testing, performance testing, security testing, and integration testing, among others, were not used. Unit testing verifies the accuracy of individual pieces of code and enhances the quality of the code. It encourages developer cooperation and reworking. System testing helps verify the behavior, user interactions, and compliance with requirements of the overall system. System testing also helps evaluate user expectations and system performance. Performance testing helps measure how responsive, stable, scalable, and resource-efficient a system is under varied workloads. Performance testing also helps enhance performance and tries to ensure stability of the system. Security testing identifies risks and weaknesses to safeguard sensitive information and stop unwanted access. Security testing can help with ensuring that security standards are followed and that the system is strong. Integration testing helps verify interactions between parts and or modules to make sure they function properly together. Integration testing also helps assure system functionality and finds integration flaws. The project's needs determine the testing methodologies to use. While the others offer broader coverage for functionality, stability, performance, and security in many software development scenarios, unit testing is absolutely necessary.

I think I developed a quality control mindset as a software tester for this project. I understood the significance of thoroughly analyzing the code and its relationships in order to ensure complete testing. Understanding the code's intricacy was essential since it let me spot possible problems and comprehend how alterations to one region of the code can affect other parts. For instance, I had to think about how changing the task name would impact how other methods behave or the system's overall consistency when I tested the updateTaskName method in the TaskService class.

To try and minimize bias in my code review, I tried to use a systematic approach that focused on reviewing the code through the lens of best practices, and the project's requirements. Bias could be a concern in certain situations if the same developer that created the code also tests it. Not always, but in certain cases a programmer could miss things in their review because they might not see the program the same as someone that did not develop it, and might look at it through a biased lens. Testers that did not develop the code, could potentially think of new and out of the box ways to test the program, because they potentially did not witness every piece of the program being developed, and might not have assumptions about things potentially being correct in the code.

Being disciplined in the area of quality can be very important as a Software Engineering professional. This is because creating quality code and tests can help minimize any potential bugs in the software before it gets released. Releasing low bug quality software is very important because it means that development time could be cut down if things were developed correctly the first time, and for certain companies, having quality code can lead to a decent brand reputation, leading to more business. Creating quality software can also help avoid less than ideal situations like a substantial scale project having malfunctions and costing the company a lot of time and money. There are so many reasons to produce quality code as a professional, so even if it takes a bit longer and or a bit more funding before release, it could be worth it. Things that I might try to do to create quality code are to follow industry best practice programming standards, and to thoroughly test my code.