**7-2 Project Two**

Katrina S. Gilliland

Global Campus, Southern New Hampshire University

CS-320-R3312 Software Test Automation & QA

Professor Kalysa Wilson

February 25, 2024

**7-2 Project Two**

**Summary**

To ensure my testing approach aligned to the software requirements, I constantly referred to the rubric while creating my JUnit tests. In the contact application within this project, the objective was to create “Contact” and “ContactService” classes containing contact objects which could be added, updated, and delete; to meet the client’s requirements for this program, I created JUnit tests to verify that the contact service methods would function as expected. The tests I wrote for the contact application worked to confirm that a contact object’s contact ID, firstName, lastName, phone number, and address fields are not null and that they do not exceed a certain character limit. My tests were also written to ensure that the overall mobile application allows for contacts to be added, deleted, or updated using their uniquely assigned contact IDs. Further, they were written to confirm that all fields except for a contact’s contact ID could be updated. In the task application for this project, the purpose was to create a module that handles task services. As with the contact service application, my job was to ensure certain fields for a task object are not left null and do not exceed a certain character limit, as well as that tasks could be added, deleted, or updated using their unique task IDs. For the task service program, only a task’s name and description should be able to be updated, not its task ID.

Test coverage in JUnit testing is a useful way to find out which parts of a system have yet to be tested. A test coverage rate of 80% or higher is considered reasonable as it confirms an application’s code covers all the functional requirements and expected behaviors for the program (Garcia, 2017). While I did find it challenging as a new programmer to get my JUnit tests to result in a high coverage rate, I was able to use the coverage results to further enhance my code. Running coverage tests helped me refine portions of my code by highlighting which lines have been utilized in the tests and which have not as well as assisted me in creating additional test cases for the areas reporting low coverage. Test coverage helped me reach a point in my JUnit testing process where I felt my code was thoroughly tested and met the quality standards for this project.

I made sure my code was technically sound by confirming that there were no blatant errors in it before running the tests. For example, in the contacts service application, I had switched the variable for a contact’s phone number from “phone” in the “Contact” class to “phoneNumber” in the “Contact Service” class without even realizing; the small oversight in the variable name for this field caused a ton of errors in my code, which would have resulted in very unsuccessful JUnit tests. I also made sure my code was technically sound by utilizing the JUnit tests. In the task service application, I used JUnit testing to verify that tasks could be added using a task’s unique task ID (shared below). My JUnit testing also verified that a user would not be able to add the same task twice, confirming that the task ID feature is implemented correctly and working as expected.

@Test

**void** testAddTask() {

TaskService taskService = **new** TaskService();

Task task = **new** Task("0123456789", "finish homework", "finish task service project for milestone");

*assertTrue*(taskService.addTask(task));

*assertFalse*(taskService.addTask(task));

}

I ensured that my code was efficient by using industry standard best practices, such as including clear comments and utilizing loops and branches. Providing comments is important in software development because it offers others who may interact with a piece of code a greater explanation of its purpose and overall intended functionality. Clearly communicating a program’s intentions will increase code’s readability and ultimately, make it easier to maintain. Employing loops and branches is also a great way to ensure code efficiency; for example, in the “Contact Service” class I created for the contact application, I included a for loop and if branch to search a contact list for a specific contact ID (shared below). If the contact ID was found in the contact list, that specific contact could then have their first name, last name, phone number, or address updated. If the contact ID was not found, the program would “return false,” meaning the contact would not be able to be updated as it does not exist.

**public** **boolean** updateContact(String contactID, String firstName, String lastName, String phoneNumber, String address) {

**for** (Contact contactList : contacts) {

**if** (contactList.getContact().equals(contactID)) {

contactList.setFirstName(firstName);

contactList.setLastName(lastName);

contactList.setPhoneNumber(phoneNumber);

contactList.setAddress(address);

**return** **true**;

}

}

**return** **false**;

}

**Reflection**

The software technique I primarily employed in this project was conducting a form of white-box testing known as unit testing. The unit tests I created isolated each respective section or functionality within the Contact, Task, and Appointment applications and allowed me the opportunity to ensure that each block of code was running as expected and per my client’s software requirements. Using Java’s unit testing framework, “JUnit,” I created unit tests for all three applications. In the task application, for example, my JUnit tests revolved around testing specific features in the program such as the ability to add tasks using a unique ID, which cannot be null or more than 10 characters long, and the ability to apply a description for the task, which cannot be null or longer than 50 characters. Through performing JUnit tests, I was not only able to verify that the functionalities I added to each program were operating correctly and in-line with the client’s vision, but I was also able to catch any errors and defects in my code. Testing the individual units of source code gave me the chance to identify issues early on and avoid having them be too deeply embedded into my programs. In a real-world scenario, this would save a project both time and money, as opposed to if the defects were found later in the development process.

Software testing techniques that I did not use for this project that I think would be useful to apply in many software development scenarios would be performance testing and user acceptance testing (UAT). Performance testing involves the evaluation of a system to determine its level of responsiveness and stability when put under a certain workload. Performance testing is an effective testing technique for confirming that software provides users with a positive experience as it focuses on speed, system response times, and scalability. Two types of performance tests that can be conducted are load tests and stress tests; a load test simulates many users accessing an application to make sure its architecture can handle large load conditions and a stress test evaluates how a system behaves when it reaches peak user activity (Tricentis Staff, 2022). Performance testing would have been a benefit to my programs if I had been given more information from the client on what they are expecting regarding application usage and scalability. Another testing technique that I did not use in this project that I believe is significant in software development is user acceptance testing. User acceptance testing, also known as end-user testing, is normally done at the end of a software development life cycle (SDLC) project. UAT checks whether the completed program fits the expected outcome of the client and is ready to be launched to the public. In a real-world scenario, this would be a great way to ensure an application is released with no or very minimal defects and bugs.

All sorts of SDLC projects and situations could benefit from the testing techniques mentioned above. Unit testing, for example, is essential to identifying weaknesses in a program. Since unit testing tests the smallest functional units of code in an application, it can be easier for a developer to detect what may need to be added, modified, or removed, to ensure the code’s quality. As noted earlier, since unit testing is conducted early in the SDLC, defects are found much sooner than if caught at the end of development and therefore, can be addressed before they become larger, more complex issues. Employing unit testing in any SDLC project will ultimately save the client a significant amount of money and the developers a significant amount of time. Performance testing, which is also done early in the SDLC, can reduce time and costs as well. Since performance testing utilizes varied scenarios to highlight performance issues that may negatively impact user satisfaction (speed, scalability, stability, etc.), it is a great way to ensure that the final software product meets client and end-user expectations and can efficiently handle different levels of input. User acceptance testing, which is done at the end of the SDLC, can benefit any software development project by confirming that a program is compatible with real-world conditions. Comparing the final software to the business requirements shared by the client ensures product readiness by confirming that the program is error-free, user-friendly, and effectively fulfills user needs.

**Mindset**

During my work as a software tester for this project, I adopted a very detail-oriented mindset. Since my main objective for this project was to deliver a high-quality application, I put all my efforts into ensuring my tests covered all the required functionalities within the contact, task, and appointment classes. It was crucial for me to employ caution when writing the JUnit tests for this program as I needed to make sure they thoroughly assessed each individual code component for its quality and overall correctness. Failing to write JUnit tests with a large amount of coverage over the application may have result in bugs or defects being found much later in its development; not only could these errors lead to a negative user experience, but it could also leave the program vulnerable to a myriad of security issues. It was important for me to appreciate the complexity and interrelationships of the code I was testing as it significantly influenced my approach to writing the JUnit tests. I found that by identifying ways to simplify the code within the units I was testing, I was able to write tests with more coverage. For example, when running a test coverage analysis on a particular JUnit test class, the highlighted red sections indicated portions of code with limited utilization. I recognized these red highlighted sections as a sign that I either need to expand on the test cases for that specific code component or refine lines of code to optimize the functionality and make it more efficient.

I tried to limit bias in my review of the code by constantly referring to my client’s requirements for the program. Keeping their perspective of how the program should operate in mind while developing the code and writing JUnit tests encouraged me to make decisions based on their objectives and desired outcome rather than my own personal preferences. On the software development side, I can imagine there being a concern of bias if a software developer were to write and test their own code as they may let their confidence in their programming skills allow them to put blinders on when it comes to potential issues or defects in specific areas of the program. For example, in this project, I made every effort to align the code I wrote with the client’s requirements as well as to conduct thorough testing to ensure its compliance; however, without being a true end-user for the application, I realize I may be missing some key insights and perspectives that are vital to a truly successful project. Having another developer give their feedback, tasking a dedicated testing team with analyzing the code, or having end-user’s beta test a program can offer a great advantage to any software development project by helping to identify possible programming oversights or areas that may benefit from improvement.

As a software engineering professional, it is important to be disciplined in commitment to quality and not cut corners when it comes to writing or testing code as taking shortcuts can have a significantly negative impact on a project’s success. One negative repercussion of cutting corners in a software development project is it can lead to unintended bugs and errors. If code defects are not caught quickly enough, they can build up over time and greatly affect software reliability and stability. For instance, not incorporating proper error-handling into a program can result in major system crashes and application performance issues, thus leading to an unsatisfactory user experience. Additionally, cutting corners in code can lead to an accumulation of technical debt, which can make or break a project by delaying or completely halting further program developments. As a practitioner in the field, to avoid technical debt, I plan on doing my best to adhere to industry standards and best practices, implement extensive testing procedures, and pledge myself to continuous learning. By adhering to industry standards and best practices I can ensure that my code is clean and maintainable and through comprehensive testing I can validate its reliability. Committing myself to continuous learning will allow me the opportunity to stay up to date on new technologies and will give me the chance to grow and enhance my programming skillset so I can contribute to the development of high-quality programs and applications.

**References**

Garcia, B. (2017). Mastering software testing with junit 5: a comprehensive, hands-on guide on unit testing framework for java programming language. *Packt Publishing*, 1. <https://eds-p-ebscohost-com.ezproxy.snhu.edu/eds/ebookviewer/ebook/bmxlYmtfXzE2MjY5NTBfX0FO0?sid=8610dbe3-3e94-481f-96c5-a71ddcb7a418@redis&vid=0&format=EB&rid=1>

Tricentis Staff. (2022, April 14). Performance testing, best practices, metrics & more. *Tricentis*, 1. https://www.tricentis.com/learn/performance-testing