**7-2 Project 2:  Grand Strand Systems**

Danielle Williams

Southern New Hampshire University

CS-320: Software Test, Automation QA

Professor Sherif Antoun

October 16th, 2025

While writing and reviewing my unit tests, I applied testing techniques for test cases designed to evaluate the individual components of each class to verify their functionality and ensure they behave as expected. I utilized assertions such as true, equals, and null.

Each of my requirements were clear and achievable, so I ensured that I implemented tests to catch any errors or invalid operations to meet them. For instance, in each class, one requirement was to ensure that an input did not exceed a certain length or was not a duplicate. I created tests to verify that any inputs exceeding the requirements were not accepted and provided an error message to indicate the amount entered or to catch a duplicate.

I conducted tests with coverage and achieved scores of 97% and then 100%. These results indicate that the JUnit tests were effectively implemented, highlighting the quality of the tests.

These are useful for identifying any issues or failures before deployment or implementation. These approaches help uncover bugs, confirm that features function as intended, and assess how well the software performs under pressure.

I made sure my code was sound and efficient by applying the ‘Three C’s of Software Quality: Competency, Completeness, and Correctness.’ Correctness means the software can perform its intended functions without errors or bugs. Completeness refers to how well the software meets all its specified requirements and performs all its intended tasks. This is important because it ensures the software satisfies user’s needs. Lastly, Consistency is about how uniformly the software behaves across different platforms and environments.(2023, May 2)

@Test  
void testValidAppointmentCreation() {  
 Appointment appt = new Appointment("apt\_1", getFutureDate(), "Valid description");  
 *assertEquals*("apt\_1", appt.getAppointmentID());  
 *assertEquals*("Valid description", appt.getDescription());  
 *assertNotNull*(appt.getAppointmentDate());  
}

This code snippet from the project serves as an example of the requirements being met by validating the future date is equal to the expected outcome as a date in the future- the description is as expected and ensures the appointment date is present. The test passes, is correct and complete.

I used Functional Testing to verify that each function of the software application works according to the requirements and specifications. This technique involves testing each functionality by providing appropriate test inputs, expecting specific outputs, and then comparing the actual outputs with the expected results. Additionally, I conducted Unit Testing, which involves testing the smallest parts of the code, allowing me to verify the correctness of each piece by running them individually. This approach is crucial for improving quality, as it tests each unit in isolation. If a test fails, I can quickly identify and fix the issue. This method ensures better test coverage and allows me to focus more on coding rather than manual testing.

A technique I did not use in the project is complete Manual testing. This is an important part of software development that involves a person actively using the software to find bugs and issues. This approach helps to ensure the software works as intended and meets user needs. It begins by understanding what the software should do. First by looking at the Functional Requirements, for example, the login page must verify user credentials correctly. Then, Non-Functional Requirements which focus on performance, usability, and security. An example is ensuring the login page loads in under 2 seconds. Then review User Stories and Design Documents to see how users will interact with the software and meet their needs.(2021, February 22).

I found a simple yet effective way to appreciate the interrelationships in the code I tested by examining the file names and their imports. In my previous journals, I mentioned that I encountered issues with having lowercase file names, which I needed to update to capitalize. This change was especially important for the final project when I was consolidating everything and aimed for a consistent system that would be more team-friendly.

The complexity of the code became particularly evident in the latest milestone assignment, where I was required to check for dates. I used caution when employing the calendar import to retrieve past dates. It was essential to ensure that the function was accurate in providing the correct test data; otherwise, my tests could have failed, leading to confusion about whether the issue lay with the input I used or the actual test setup. This is shown in the snippet above with “getFutureDate()”

In a short journal entry written by 'Coders Stop,' I found the mention of 'the curse of knowledge' particularly resonant. I often struggle to see potential issues, even when they are right in front of me. This bias can linger, so I've developed a strategy to combat it: I pretend that I'm reviewing someone else's work. To do this, I close my application and take a short break before starting back into my project files. When I reopen the application, I approach the review as if I'm assessing another person's work. This allows me to conduct tests more objectively and walk through any failures more carefully. During one test run, I encountered failed tests where the expected outputs didn’t match the actual results. To resolve the discrepancies, I needed to ensure that my inputs were correctly specified, as I had mistakenly set them as the same which they should have been different.

In this module's discussion, it is clear that cutting corners can be detrimental to finances, the environment, and advancements. While the topics shared may have been extreme, they are necessary to understand how one simple error can lead to events that result in a loss of trust, money, and safety. I wrote about the northeast blackout that occurred in 2003 and responded to classmates who shared the events of a failed NASA launch and another outage. Each of these events was due to rushing and missing the errors. In less extreme instances, cutting corners when writing or testing code can cause damage to relationships between clients and companies, ultimately failing to meet their expectations. Being entrusted to create or be part of a team is an opportunity for growth, contribution, and fulfillment. Certain expectations need to be met, and if you don’t meet them, someone else will (and probably for less a price).

Citations:

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). Software testing : An istqb-bcs certified tester foundation guide - 4th edition. BCS Learning & Development Limited.

Boni Garcia. (2017). Mastering Software Testing with JUnit 5 : Comprehensive Guide to Develop High Quality Java Applications. Packt Publishing.

Redirecting. (2025). Snhu.edu. https://learn.snhu.edu/content/enforced/2019682-CS-320-10361.202581-1/course\_documents/CS%20320%20Module%20Four%20Tutorial%20Using%20JUnit.pdf?ou=2019682

‌ Jakubiak, N. (2018, August 2). JUnit Tutorial With Examples: Setting Up, Writing, and Running Java Unit Tests. Parasoft. https://www.parasoft.com/blog/junit-tutorial-setting-up-writing-and-running-java-unit-tests/

Kitakabee. (2023, May 2). 15 Techniques to Improve Software Quality. BrowserStack. https://www.browserstack.com/guide/how-to-improve-software-quality

‌GeeksforGeeks. (2021, February 22). Software Testing Techniques. GeeksforGeeks. https://www.geeksforgeeks.org/software-testing/software-testing-techniques/

Coders Stop. (2025, April 18). The Cognitive Bias That’s Ruining Your Code (And How to Fix It). Medium. https://medium.com/@coders.stop/the-cognitive-bias-thats-ruining-your-code-and-how-to-fix-it-2885178c63ac