Project Two – Summary and Reflections

CS-320-T4205 Software Test Automation & QA 23EW4

Scott Dixon

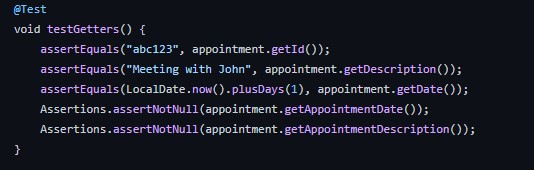
April 15, 2023

As a software engineer working with Grand Strand Systems, I would like to describe the development and testing of the back-end services we have been working on to bring the mobile application from concept to reality. The initial concept of the mobile application was straightforward and relied on the various modules to work together in unison so the end-user would be able to input the needed information, the application would then parse the information, return the answer, and then output the result to the user. Our task, as developers and testers of the application, was to create and perfect the code base by writing smaller chunks of the program and then testing as we went along, trying to bring the application to its final stages within the three weekly sprints that the project allowed us. In this paper I describe the process of aligning the requirements that the end user desires with the properties that the finished product has to offer.

During the process of building the application which was broken down into three modules; Contact Service, which uses in-memory data structures to support storing contacts, allowing the user to store and retrieve information such as first name, last name, Id number, and address. The second module, Task Service, allows the user to add, update, and delete task objects within the application. The third module, Appointment Service, allows the user to add and delete appointments within the context of the application. The final deliverable application was evaluated at every step along the way to provide an excellent user experience along with a design that the end user will find pleasing and efficient. The project was designed with the requirements in mind and the aspect of the finished code models the output that is necessary for the fully functional application.

Effective testing is extremely important when it comes to delivering a useful and functional software product. In terms of unit testing, the approach I have adopted is aligned with the software requirements and focuses on assessing the individual components of the software in isolation. The test cases covered all scenarios and edge cases to ensure that the code is working as expected. The use of automated testing frameworks such as Junit 5 helped to ensure the repeatability and consistency of the test results. The effective testing principles that I have chosen to use include white box testing since I knew the code and the configuration of the application and utilized the Junit 5 testing scenario. The other available forms of testing include; black box testing, where the tester has no knowledge of the code or the structure of the program, gray box testing which combines the benefits of both white box and black box and relies on the goal of identifying defects induced by improper usage or the structure of the application. (Dhore, et al., 2023)

To ensure the overall quality of JUnit tests, I have learned that it is important to measure the code coverage percentage. Code coverage measures the percentage of code that has been evaluated by the unit tests. A high code coverage percentage indicates that a substantial portion of the code has been assessed and reduces the likelihood of undetected bugs. However, a high code coverage percentage alone does not guarantee that the tests are effective. In developing the application for the end user, the quality of the test cases and their ability to detect software defects were also considered.

The codebase must be efficient and contain technically sound coding principles that lend them to be easily tested. The ability to evaluate software effectively relies on the fact that the code being developed is technically sound and is designed with efficiency in mind. For example, the code that I have written for the Appointment Service module includes a test routine that assesses the “Getter” part of the Getter and Setter code methods used to protect the data and make the code more secure. The method is important in the code structure since it hides sensitive data by using Java encapsulation. In this case, a method to access and update the value of the “private” variable and test if the assertions of the statements are equal or not null. This software testing technique has practical uses in other software applications by insuring clean code standards and object-oriented programming design. (Zaw, et al., 2019)

The technique that I employed to test the application is Junit 5. I had no experience performing testing with a structured, test-driven development environment, but am very happy that I have been introduced to JUnit 5.

The Junit 5 testing is accurate and efficient since it can run numerous tests in batches that can reduce the amount of time spent evaluating a code routine. The process of testing gives me the possibility to understand if the code verification is going as planned or "am I building the software product right?” and allows me to gauge if the code validation is in line with customer expectations, which is to say, “am I building the right product?” (Garcia, 2017)

The other techniques of testing that I will consider as the project review goes into the post-release stage will be functional testing of the finished product which verifies the functionality of the software application. It involves evaluating the application against functional requirements and specifications. Another type of testing will include the security testing phase which evaluates the security of a software application to identify vulnerabilities or weaknesses. This will include testing for authentication, authorization, encryption, and other security-related aspects. Finally, the application will be assessed for useability. The useability testing includes a focus on the user interface and the user experience aspects of the finished application. This will involve testing how difficult the application may be to use and whether the product meets the needs of the end user.

In software testing the Grand Strand application, it has been crucial to adopt a cautious mindset and evaluate the software thoroughly to identify any defects in the code. It is important to consider the complexity of the code being tested and to make sure that all functionalities are tested thoroughly so that any potential bugs are isolated. For instance, a simple change in one function can affect the behavior of the entire system, so I need to consider the implications of my choices in both writing and evaluating the code base. In the case of writing my own routines for the application and then attempting to test the same code, a certain bias appears that could introduce a lot of unneeded errors into the application. Bias can be a concern since I am responsible for evaluating the code which I feel is correct by logic standards in the first place. For example, I may tend to overlook potential defects or prioritize certain features over others, leading to biased testing results. Here are code snippets that support the fact that the code needs to be correct and structurally sound before it is tested:

Text

Description automatically generated

In this example, the assertions were simple enough but the code that was to output the Local Date was not structurally sound, so it was returning incorrect data back to the test. After looking at the test parameters of the code I was forced to rewrite the code, so the data was accurate and yielded a test that passed. This is one aspect where bias may come into play when the developer is creating and testing the code by themselves. (Zivkovic, 2021)

Looking back over the coding experience of the last few weeks, I have found that the software program needs to be organized into the most logical design and that the testing needs to be structured using tests that are well formed so the assertions test for any incongruent statements in the code, therefore, creating the most effective, efficient, safe, and bug-free application.

References,

Dhore, P., Wadhwa, L., Shinde, P., Chaudhri, D., & Vyas, P. (2023). Brief Review on Different Manual Software Testing Approaches & Procedure. Journal of Pharmaceutical Negative Results, 14, 455–464. <https://doi-org.ezproxy.snhu.edu/10.47750/pnr.2023.14.S02.56>

Zaw, K. K., Zaw, W., Funabiki, N., & Kao, W. C. (2019). An informative test code approach in code writing problem for three object-oriented programming concepts in java programming learning assistant system. IAENG International Journal of Computer Science, 46(3), 445-453.

Boni Garcia. (2017). Mastering Software Testing with JUnit 5 : A Comprehensive, Hands-on Guide on Unit Testing Framework for Java Programming Language. Packt Publishing.

Medium.com [https://blog.devgenius.io/9-bad-java-snippets-to-make-you-laugh Accessed 04/16/2023-Milos Zivkovic 82f5018e06e5](https://blog.devgenius.io/9-bad-java-snippets-to-make-you-laugh%20Accessed%2004/16/2023-Milos%20Zivkovic%2082f5018e06e5)