**CS 320: PROJECT TWO**

Ian Viguera

Southern New Hampshire University

CS 320

Professor. Norman

02/25/2024

The approach taken towards software requirements was a direct one. Customer needs and requirements were the basis of the complete system, mainly because of the specific problems that our customers wanted solved. For this reason, JUNIT tests were conducted to ensure that the functions that solve the specific customer problems were functioning correctly. Multiple instances of user inputs were considered to minimize errors and enhance program functionality. The most common instances of user inputs included situations like: Null first name, Null last name, Null address, and so on. The focus of these strategic JUnit tests was to reduce, or eliminate, possible escapements these functions might create.

To ensure that the code was technically sound and efficient, common coding practices were performed to ensure that, not only JUnit tests benefit from the code structure, but also be neat and readable. Although readable code does not seem like a focus point, it is important to consider when exporting your code to other developers. After meeting these standard practices, the technicality of the program functions is the most important step to follow. In the “Appointment Service” program, it was important to consider that new appointments should be done only after the current date. Using the java tools “Date” and “Calendar”, I created a set variable “date” with the program’s launch date and created the user functions around it. This prevents false instances or errors such as making an appointment before the established date. Having a combination between a technically sound code and an effective code ensures functionality of the program and eases usability.

The software testing techniques used for this application include most of the common techniques used for testing. This is because functional, unit, and system testing were the main techniques used for this application, but non-functional and integration testing were considered. The main testing element was JUnit testing to make sure that the code executes correctly under different automated tests. Another reason is that it prevents user instances that could potentially exploit the different program functions or individual code lines. For example, A JUnit test was conducted, on “ContactTest” class, to verify if the user inputted a “firstName” that swayed beyond its parameters. Once a long name test was set, the test returned as a failure, proving my hypothesis. Non-functional and integration testing were not considered due to the simplicity of the program and because of the different classes working independently from each other.

It is highly recommended to approach software testing with caution and discretion. To employ a suitable test the developer must approach with a user and customer mindset before approaching as a tester. This is because users tend to sway from the program’s parameters in its functions and could prompt the program to return false statements or shut down. As stated before, the JUnit tests that were conducted were done so with possible errors and/or unconventional inputs in mind to keep the application running even with unsupported arguments. The program’s code must be reviewed with as little bias as possible because other developers may examine and/or expand upon it. Bias and discipline act similar when it comes to testing as it is important to implement testing techniques with as little bias as possible without cutting testing corners.

Cutting corners in testing is the principal reason for catastrophic errors, exploits, and malware attacks. This statement is backed up with the real-life example of the “Ariane 5” aircraft that exploded 40 seconds after launch because of a wrong variable declaration. The developer “cut corners” and committed a slight error that cost nearly 500-million-dollar loss of an ongoing project. To avoid technical debt in the field, it is wise to review the code under every test case possible and with logical precautions. The combination of these two practices not only avoids program failures and illegal arguments, but also avoids developer miscalculation.

**References**

(December 06, 2023). *Software Testing Techniques*. Geeksforgeeks.com

A. Douglas (August 23, 2000). *The Explotion of the Ariane 5*. https://www-users.cse.umn.edu/~arnold/disasters/ariane.html