Project Two

Garrett Dunn

CS 320

6/23/2022

**Summary**

The software testing done for this project was created to align with the requirements made by the customer. The testing was completed in the initial stages of the development cycle to ensure any defects were squashed out in accordance with the customer’s requirements. Peer review was implemented for important feedback. The software code was combed over multiple times for any defects or issues needed to be corrected. Dead code was removed after analyzing the control flow. The data structure was also checked and tested for additional information of the flow.

The JUnit tests conducted allowed us to figure out which versions of the software were defective. These JUnit tests proved to be extremely useful for finding specification bugs and dead code in the software. Along with this, the tests involved generating input values for the software program, which was very comprehensive and exhaustive. By implementing coverage through components and integration, we were able to make sure that the code was as technically sound as possible.

To further ensure that the code was efficient, experience-based techniques were used. Things like error guessing techniques were utilized with structured methods to siphon out any weaknesses. We pushed the program to the limits to test the strength and integrity of the program.

**Reflections**

The software techniques that were employed in this project include boundary value analysis, equivalence class partitioning, and decision-based table testing. The utilization of boundary value analysis allowed us to test the minimum and maximum error values within and outside the code boundaries. Many of the errors found with this method occurred on the defined input boundary values. The decision-based table was utilized to test combinations of events. This allowed us to better decipher functionalities of the program in a shorter time.

Another technique utilized was a cause-effect table. This was used to identify any conditions that had the possibility of being overlooked. Finally, using equivalence class partitioning, we were able to divide specific test conditions into separate compartments we thought to be similar. This allowed us to identify inaccurate data sets within the code of the program quickly and efficiently.

There were many different testing techniques that were not utilized in the development and testing of this software. Some of those include black box testing, white box testing, grey box testing, and state transition testing to name a few. Black box testing involves no prior knowledge of how the software functions, while inputting values into the program and checking the output from those input values. White box testing implies the functionalities of the software, and the code itself is evaluated to siphon out any issues or defects within the program. Grey box testing is executed with little knowledge of the program’s functionalities, however there is access to the design and database of the program to help create a test plan. Finally, state transition testing relies on the input of conditions sequentially to determine the program’s performance under stress using positive or negative test values. These testing techniques were not utilized in this project.

Maintaining discipline and limiting bias as a software developer requires many strategies. One particularly important strategy is to limit oneself to the inspection of no more than 400 lines of code per hour. After 400 lines, one should take a break and refresh one's mind and body. Other strategies include setting goals to ensure that the project is being completed in an efficient and timely manner. This can help ensure the accuracy and integrity of whatever project you are working on. Annotating the source code before review maintains quality and significantly lowers defect density. Also, by utilizing checklists, one can ensure a straightforward way to manage guidelines, avoid omissions, and report errors to improve the software being developed. By establishing and utilizing specific processes to uncover and fix bugs and defects, it will ensure the quality of work needed moving forward as a developing software engineer.