**CS-320 Journal Module 5:**

**Software Testing Techniques**

Damean Murphy-Short

September 26, 2023

Southern New Hampshire University

Over the course of the previous projects undertaken by the author, several software testing techniques were learned and applied to the assignment at hand, while others were not used. Of the techniques excluding from testing, many did not show a benefit when it came to delivering the requirements directly in line with the project in relation to their time commitment when compared to those that were involved in the testing process.

The most utilized technique employed to test the Contact, Task, and Appointment classes was boundary value testing and input partitioning, sometimes called ‘equivalence partitioning.’ In the experience of the author, off-by-one errors are the most common cause of software defects. This makes boundary value testing an invaluable tool, as it forces one to analyze the result of inputs on a function that exist on this boundary, above it by one, and below it by one. Since the fields in the classes above often had strict length requirements, this was the most logical choice of technique to verify whether this requirement was actually being met in a meaningful way. This type of testing is most important in languages where dynamic memory management is handled by the programmer. Memory allocation can often cause a myriad of difficult-to-debug issues that may not become apparent in most cases and may seem “random” when this off-by-one value is experienced in production. In many cases, these errors go undetected for a long time until becoming vulnerabilities (“Memory Safety”, 2023).

The equivalence partitioning technique played a similar role in that it separated valid input and invalid input into two partitions, those being “valid length values” and “null values” which were tested against an expected outcome. Since the boundary value testing already tested “values of invalid lengths” we can say that upon this secondary testing, all possible partitions had a chance to be examined for undefined or unwanted behaviour.

The code was also statically analyzed before undergoing dynamic tests. It is important for a programmer to ensure that their code is sound structurally by walking through it manually, and also ensuring that the code they have written is adherent to the requirements by taking a step back and examining what it will do once executed, so that one can be assured they have written the correct code for the project. In this particular case, the code did not undergo a more sophisticated static analysis like group review. This complemented the coverage analysis done by JUnit, which verifies what the static testing expects to see when walking through each statement.

For sake of brevity, we will now leave the techniques that were used and review some useful techniques that were not used. Primary of these are both integration and system testing. Since the author is unaware of how these projects will eventually work together and integrate with one another, these types of testing are not possible at the current stage of development. They play an important role in ensuring that the entirety of a product is as strong as each of its components, and should take place during any more-finished project.

Performance tests were seen as less important for such simple segments of code where very established data structures, like ArrayLists, and very simple algorithms were used. In a project where one is designing a large, non-volatile database, for example, a test of the performance would be paramount to nearly any other tests because of how integral performance is in the architecture of data structures and searching algorithms. For similar reasons, security testing was excluded from the roster of techniques employed because it was simply less worth the time than the other forms of testing used. As per the laid out requirements, we are more concerned that the description field is not null than that it does not accept as input some kind of SQL injection.

In conclusion, each project must decide what kind of testing is worth the time commitment based on its own requirements and priorities. All testing techniques have value, but some provide value more meaningfully to certain units, as exemplified by the importance of value-based testing in this case.

References.

Memory safety. (2023). *Android Open Source Project*. https://source.android.com/docs/security/test/memory-safety