Software Testing Project Reflection

Uriel Lujan & Elvin Palushi

The work that we did on this project was overall a valuable experience at first hand software testing that reinforced both our understanding of the theory of software testing such as the control flow graph section of the project as well as the application of software testing principles. The overall project can be thought of as a careful and methodical process that begins with modeling our control flow graphs, block tables, and capturing our baseline source code that eventually ends with the creation and use of carefully crafted test cases. Our perception of end-to-end coverage was drastically changed because of our experience in trying to achieve it.

Constructing control flow graphs for each method could be compared to figuring out how to carefully put a puzzle together. The process required us to go line by line of the source code and make a map out of every possible branch that could affect the flow of execution. This allowed us to visualize the structure of the code that was abstracted away due to the high-level nature of object-oriented languages like Java. Careful examination of the flow of execution from a testing standpoint is very different from a development perspective. This is because as a developer you are mostly trying to syntactically and semantically achieve a goal, but as a tester you are eliminating as many mistakes in the semantics as possible that might have been overlooked when developing the program while also working under the limitations set by the structure in which the program was developed.

Following the creation of the control flow graphs we moved onto listing all the test paths and translating them into applicable test cases. Our analytical skills were further enhanced because we had to take many factors into consideration and reason through program behavior with given inputs to deliver the outputs that we expected. We were able to use a helpful testing tool, the oracle, which gave us insight in how the program was expected to work. Furthermore, the test paths allowed us to design test cases that were effective and maximized edge coverage.

The execution of our test methods was an exciting manifestation of all the hard work we did with the control flow graphs and the test paths. We were able to quickly find some faults in the code, but others we're more obscure and hidden. We made sure to make our testing suite as robust as we could to achieve a high coverage of all statements and branches. For our testing to be effective we concluded that testing isn't linear, it involves many iterations and a mindset of thinking outside of the box.

Generating the HTML coverage reports gave us a high-level, visual representation of the coverage achieved by our test cases. Seeing visual confirmation of our edge coverage progress, both at the unit and through end-to-end testing levels, was deeply satisfying. These reports not only served as deliverable artifacts but are also proof of our thoroughness put into our testing process.

Ultimately, this project taught us more than how to follow a testing process or achieve coverage goals. This project taught us how to analyze code line by line and further understand semantics, while also working effectively under the existing structure of the program. We can confidentially say that we have improved our testing skills.