1 Introduction

The purpose of this lab is to become familiar with integration white-box testing. The goal of this lab was to give us experience with using Python and Python unittest unittest.mock for integration testing.

Integration testing serves as a logical extensions of unit testing. There are two general approaches to integration testing – non-incrementing testing (Big Bang) and incremental testing (Top down/Bottom up). In non-incremental testing, each module is tested individually and then the whole system is tested as a whole. In integration testing, we combine the next module to be tested with the set of previously tested modules before running tests. This can generally done in either a bottom up or top down method. A bottom up method involves testing the lowest level modules in isolation and then incrementally adding higher and higher module levels. A top down method involves testing the highest level modules in isolation and then incrementally adding lower modules.

Integration testing usually involves using various stubs and drivers. Stubs, in integration testing, is used as a stand in for lower level modules that are not currently under test. A stub returns a dummy value or makes an assertion so that the test case can ensure it was called. Drivers are a piece of testing code which makes it possible to call a submodule of an application independently.

2 Conclusion and Discussion

In this lab, we were introduced to the white-box testing strategy. We focused on control flow testing and coverage criterion. Specifically, statement coverage, branch coverage, condition coverage, and path coverage. Because we had the source code, we were able to make a control flow graph and test cases by inpsecting the source code. We were also able to form tests that would check for correctness of the program, but also check for branch and line coverage. Statement and branch coverage do not indicate good tests, but indicate lines or branches tested. This gives the developer a good idea of what has been tested. In the second part of the lab, we learned pairwise testing and used tools like allpairspy to help generate test cases. We found a similar orthogonal array and generated test cases. Then we used allpairspy to generate test cases. Doing this, we found that both methods were equally effective at generating and reducing test cases. We then discussed some pros and cons of pairwise testing where a con is that some edge cases may still occur when more than two inputs interact with eachother.