

ECE 322 Lab Assignment 4

Integration Testing

Overview

The objective of this lab is to become familiar with integration white box testing techniques tools. This lab makes use of Python and Python unittest, unittest.mock for integration testing. The objective is to understand the difference between incremental and non-incremental integration testing.

Integration Testing:

Integration testing serves as a logical extension of unit testing. There are two general approaches to integration testing:

1. Non-incremental testing (Big Bang): Test each module independently then test the system as a whole. If there is dependency between modules, group and test them together.
2. Incremental Testing (Top down/Bottom up): Combine the next module to be tested with the set of previously tested modules before running tests. Generally done in either a bottom up or top-down way:
 - a. Bottom Up: Test the lowest level modules in isolation, then incrementally add higher and higher level modules.
 - b. Top Down: Test the highest level modules in isolation stubbing out lower level functionality, incrementally add lower modules.

Generally, integration testing requires the use of various stubs and drivers.

1. Stubs: In integration testing a stub is used as a stand in for lower level modules not currently under test. Generally, a stub returns a dummy value or simply makes an assertion so that the test case can ensure it was called. **In python a stub is a mock which has a predefined implementation that returns static data (fixed data).**
2. Drivers: A driver is simply a piece of testing code which makes it possible to call a submodule of an application independently. Driver code may require stub setup, object initialization and so on.

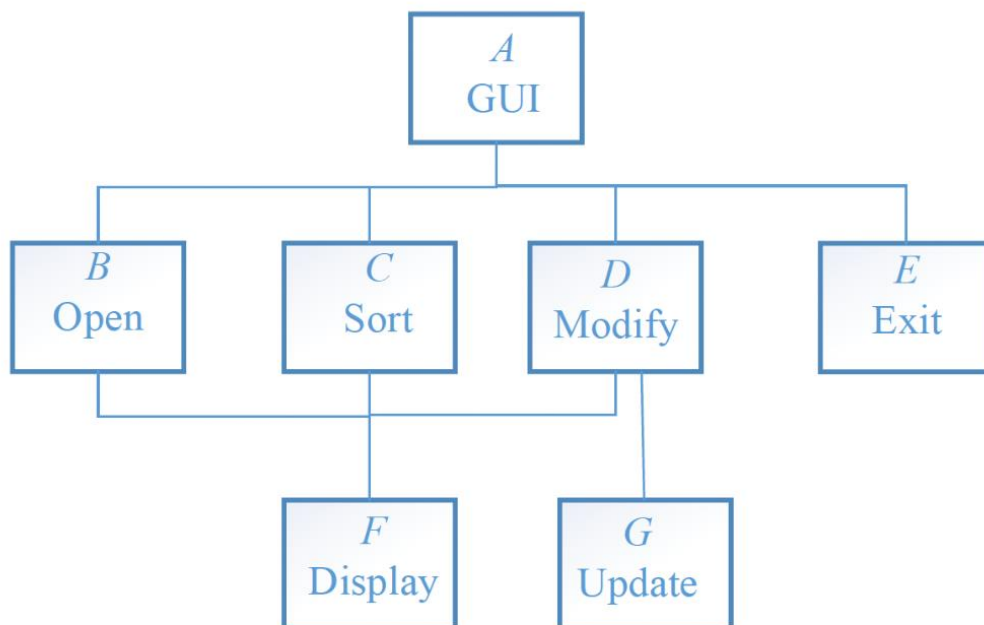
Mocking Frameworks:

Mocking frameworks can be extremely helpful in abstracting away much of the extra work involved in creating and maintaining stubs and drivers. These frameworks allow for the easy creation of Mock Objects. In this lab, we will use `unittest.mock` package.

Task (100 marks)

Prepare test cases for the tasks you will be working on in the lab session. Make sure you have PyCharm and Python `unittest` installed on your account. The code for this lab is available on the class website and can be imported into PyCharm as an existing project.

Consider a simple database system which has been constructed in a modular fashion: Module A invokes Module B, Module C, Module D, and Module E. Module D invokes Module F, and Module G. Modules B and C also invoke Module F.



Entries in the database are composed of two elements, a String name and a String Phone number. Module A is a Command Line Interface which processes command strings and delegates functionality to sub modules:

1. Open a data file (Module B)
2. Sort records (Module C)
3. Modify a record (Module D)
4. Exit (Module E)

Files for this database should contain one entry per line, and elements of an entry should be comma separated. Sort function sorts records by first name. The modify function (Module D) additionally uses the Display function (Module F) and the update function (Module G).

Your task is to prepare and run test cases, stubs, and drivers for:

1. Non-incremental testing (Big Bang integration)
2. Incremental testing (choose one to implement):
 - a. Bottom up: Test the lowest level modules in isolation, then incrementally add higher and higher level modules.
 - b. Top down: Test the highest level modules in isolation stubbing out lower-level functionality, incrementally add lower modules.

Unit tests should cover the full functionality of each module; there should be at least one test for each piece of functionality and method. You do not need main function of the provided code as it implements a simple command loop.

Your test cases can be embedded in a test suite. Your test cases should be easily run against the provided application code for verification.

Run the test cases and record your observations and results in your report, as in the previous labs test cases must be presented in your report as a test case table with meaningful descriptions, expected and actual results. Failed test cases must be highlighted.

Comment on the effectiveness of integration testing in isolating individual modules. Are drivers and stubs an effective method of isolating modules? Compare the non-incremental and incremental testing method. Which type of testing is suitable for a large-scale system? Why? Which is the easiest to maintain? Consider these and similar questions in your discussion of integration testing to show you have a strong understanding of the underlying concepts and motivations of this testing technique.

Your report should include

1. Two test case tables, one for non-incremental integration testing, one for incremental type testing you choose. Test case reuse is okay where applicable; however, results should be fully listed in each table for clarity.
2. A discussion of your results regarding the application under test. This should include a discussion of any errors you found in the code, and fixes to make your tests pass
3. A discussion of the above questions regarding integration testing technique.
4. Code for testing suits. Please separate the non-incremental and incremental testing code.