ECE 322 Lab 4 Report

Introduction (5)

Q1: Write a Statement of problem and identify of applications under test

Answer:

The objective of this was to apply both Black and White box testing techniques at different levels of testing, i.e., unit testing and integration testing. This testing was performed on a simple database program provided to us, written in Python3 and using unittest & unittest.mock library's. First all the modules were unit tested in isolation using stubs and drivers. The coverage criteria was minimum 100% statement coverage. With all the modules tested individually integration testing was performed using the Big Bang approach (i.e., non-incremental). With both unit and integration testing performed, the errors found are reported on here in the lab report. Furthermore the effectiveness of integration testing is discussed.

Q2: Identification of the methodologies being used

Answer:

Both Black Box and White Box testing techniques are used at different levels. At the unit testing level White Box testing is used to individually test each module. Stubs and Drivers are used to simulate the behavior of lower and higher-level modules respectively to isolate the module under test. Black Box testing is used at the integration testing level where all the modules are tested together at once in a so-called Big Bang approach. This Big Bang approach is a non-incremental form of integration testing, while the other form is incremental testing, where bottom up or top down approach can be taken.

Definition of testing methods (10)

Q1 : Explanation of the purpose and realization of integration testing strategies

Answer:

Integration testing is the next level of testing after unit testing. Once modules have been tested at an individual level they can start to be integrated together. The purpose of integration testing is to test that these modules behave as expected when they start to interact with each other. With unit testing complete we are relatively certain our modules function individually as expected. Then once our system is starting to be built we want to test that modules interface with each other as expected too. This is realized through two main approaches, the first, as seen in this lab is non-incremental testing (or Big Bang integration). First all the modules are tested at an individual level via unit testing. Then all the modules are integrated at once. The second approach is incremental testing, where again the modules are tested at an individual

level, however instead of combining all modules together at once, modules are incrementally added one at a time for testing. Once testing has been completed for a selected set of modules another one is then added, and the new set of selected modules is re-tested. The incremental approach can use a bottom up or top down method of combining modules which is self-explanatory. Each of these approaches comes with their own inherent benefits and drawbacks.

Q2: Explain the purpose of stubs and drivers

Answer:

Both stubs and drivers serve to simulate other modules that the isolated module under test interact with. A driver is simply a module that is higher on the hierarchy and calls the isolated module. A stub is a module lower on the hierarchy than the isolated module and is called by the isolated module.

Part 1 (80)

Q1: Definition of the problem, including what the program does, what testing method is used

Answer:

A database system constructed in a modular fashion that stores records of (name, phone number) pairs is given to us to test. The database has a hierarchy structure where higher level modules depend/ use the functionality implemented in lower-level modules. Commands are passed to the database via the command line and information is retrieved from the database also via the command line. The database is a .txt file and we interact (read, write) with the database through a command line program. Our task is to test the individual models (via unit testing) and the integration of all the models using non-incremental testing (i.e., Big Bang integration). The two lowest level Modules F and G display the database stored in the program to the terminal and write data stored in the program to a file respectively. The Modules on the next level of the hierarchy are B, C, D and E. Module B opens a file and retrieves the data and stores it in the program. Module C sorts the data stored in the program by name in alphabetical order. Module D inserts, updates, and deletes data stored in the program while Module E simply exits the program. Lastly Module A at the top of the hierarchy binds all the other modules together and contains a run function that the user interacts with via commands.

Q2 : List the modules ,order , stubs and drivers you're going to use in order to apply Big Bang integration

Modules	Order	Stubs	Drivers
A	3	B, C, D, E	Yes
B	2	F	Yes
C	2	F	Yes

D	2	F, G	Yes
E	2	None	Yes
F	1	None	Yes
G	1	None	Yes
A, B, C, D, E, F, G	4	None	Yes

Q3 Test Case Table

ID	Input	Modules under test	Description	Expected Result	Actual Result
1	**Index to delete**	**A**	**Testing parseDelete method**	**ModuleD.deleteData method is called and with correct arguments**	**deleteData is called with incorrect arguments, index variable**
2	None	A	Testing displayHelp method	Output printed to terminal & returns True	As expected
3	Filename	A	Testing parseLoad method	ModuleB.loadFile method is called	As expected
4	Filename, name/number pair	A	Testing parseAdd method	ModuleD.insertData method is called & data is added to list	As expected
5	None	A	Testing runSort method	ModuleC.sortData method is called & data is sorted in DB	As expected
6	**Filename, updated name/ number**	**A**	**Testing parseUpdate method**	**ModuleD.updateData method is called and with correct parameters and DB is updated with new data**	**updateData is called with incorrect parameters, index variable**
7	**None**	**A**	**Testing runExit method**	**ModuleE.exitProgram method is called & program quits**	**exitProgram is called with newly instantied E obj, not the E obj that belongs to Module A obj**
8	“” command & “help” command	A	Testing run method with no command and help command	Message is printed to terminal	As expected

9	"add" command w/ data	A	Testing run method with add command	ModuleD.insertData is called	As expected
10	"load" command and filename	A	Testing run method with load command	ModuleB.loadFile is called	As expected
11	**"update" command with data**	**A**	**Testing run method with update command**	**ModuleD.updateData method is called and with correct arguments**	**updateData is called with incorrect parameters, index variable**
12	**"delete" command with index**	**A**	**Testing run method with delete command**	**ModuleD.deleteData is called and with correct parameters**	**deleteData is called with incorrect parameters, index variable**
13	**"exit" command**	**A**	**Testing run method with exit command**	**ModuleE.exitProgram is called**	**exitProgram is called with newly instantied E obj, not the E obj that belongs to Module A obj**
14	.txt file and data	B	Testing loadFile method retrieves data from file properly	List of Entry objects contains correct data	As expected
15	Non existent file	B	Testing loadFile method throws exception when non existent file is read from	Exception thrown/ handled and message printed to terminal	As expected
16	Name/ number pairs	C	Testing sortData method sorts list of Entry objects based on name in alphabetical order	Sorted data list	As expected
17	Data	D	Testing insertData method takes data, creates Entry object and adds it to list of data	Data added to Entry data list	As expected
18	**Data**	**D**	**Testing updateData method updates list at specified index with new data**	**Data Entry list is updated at specified index w/ correct data**	**Data Entry list is updated at wrong index with correct data**
19	Index	D	Testing deleteData method removes specified record from DB	Data Entry list is updated	As expected
20	None	E	Testing exitProgram method raises SystemExit exception when method is called	System exit	As expected
21	None	F	Testing displayData method is called when the method is mocked and prints data to terminal	Data is printed to terminal	As expected

22	Data	G	Testing updateData method correctly writes data list to file	File is updated with most current data list	As expected
23	No command	A, B, C, D, E, F, G	Testing system when no command is given	System prints message to terminal	As expected
24	“help” command	A, B, C, D, E, F, G	Testing system with “help” command	System prints message to terminal	As expected
25	“load command”	A, B, C, D, E, F, G	Testing system with “load” command	System reads from DB file and updates data list in program	As expected
26	“add” command	A, B, C, D, E, F, G	Testing system with “add” command	Change in data list is reflected in DB file	As expected
27	“delete” command	A, B, C, D, E, F, G	Testing system with “delete” command	Change in data list is reflected in DB	As expected

Q4 Discussion:

Answer:

In this lab integration testing served as a necessary and effective step in testing the provided program alongside unit testing. Without integration testing it is unknown if the modules interface with each other as expected. Several errors were discovered in the program. The majority of these errors were indexing errors in Modules A and D. I believe there was some confusion in the design of the program, for example in Module D the delete & update methods perform their actions at a given index, while the update method performs its action an index+1. Also, in Module A there were indexing errors when calling the aforementioned methods in Module D. I believe this had to do with the fact that the records in the Database started indexing at 1, instead of 0. I believe that handling the indexing all in Module A is the most elegant solution instead of having it spread across both Module A and D. The other major type of error was when commanding the program to exit, Module A created a new Module E obj to call the exitProgram method, when it already had a Module E obj as an attribute, while the program still functioned correctly it seems this was still an error. One solution to this would be to simply remove Module E as an attribute of A. For unit testing, isolating the modules using stubs and drivers was effective using Mock objects and mocking methods. This might not be as straightforward in other languages where library’s like Mock are not available. I think Big Bang integration testing was the appropriate choice for a program of this size however I do not believe that Big Bang integration testing would scale well to large systems since it can be difficult to trace errors if they are found, for a larger system I believe incremental testing would be more beneficial, because errors with interfacing could be fixed as they are found.

Code component (50)

See other directories for code. Or can be found at https://github.com/mattheas

As a note, whenever running TestRunner.py test suite or any TestModuleX.py individually I used the following command from inside the main ECE_322_Lab4 directory:

"PYTHONPATH=$(pwd) python3 tests/TestRunner.py"

Conclusion (5)

The purpose of the lab was to gain an introduction to integration testing as the next level of testing performed on software compared to unit testing. Exposure to Black Box testing working in a complementary fashion to White Box testing was another focal point of the lab. The use of unit testing and integration testing to discover errors gives the student more insight into the overall testing process from start to finish of an application. I believe that the theory of integration testing was only explored in a limited way, i.e., only briefly discussing the two approaches of integration testing (incremental vs. non-incremental), with more of an emphasis placed on the application of one of these approaches.