COMP20230: Data Structures & Algorithms **2018-19 Semester 2**

Lecture 8: Feb 13

Lecturer: Dr. Andrew Hines *Scribes: Natasha Gamboa, Robert Young*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

8.1 Outline

In this session, the importance of test driven development was introduced, as well as a recap on some of the key Object Oriented Programming (OOP) concepts. Test driven development allows for better design and implementation of our algorithms and data structures. The rest of this document will go into greater detail on the following points:

- The relevance of testing in relation to data structures and algorithms.
- Different testing strategies that can be implemented.
- The process of test driven development.
- Unit testing as a testing strategy.
- A revision of key OOP concepts.

8.1.1 Why Should We Test?

To solve problems, we need good code. How can testing help us with this? First, it is worthwhile going over the key ingredients of good code:

- Efficiency.
- Clarity.
- Maintainability.
- Correctness.

But how does testing help with this? It is important to note that testing will not turn bad code into good code. But testing will help confirm that your code works as intended and will improve the design and development process. Testing, when used appropriately, enables the better definition of your codes intent, and assists in creating a more decoupled architecture. [1] A decoupled architecture is one that facilitates the execution of components or layers independently while still interfacing with each other.

8.2 Testing Strategies

Testing, while not the cure to all coding problems, is undeniably a good idea. So how do we test? What strategies are available to us?

- Regression Testing - the re-running of both functional and non-functional tests to ensure that software, after changes have been made, still performs. Any running of the software which fails must create a test case to use for all future executions of the software.
- Unit Testing - testing the components algorithms and APIs actual output matches the expected output.
- Oracles - source of information that determines if the output of a program is correct or not. An oracle is not a test runner, but a test runner could use an oracle as a source for the correct output. [2]
- Automated Testing - required for oracles. Suited for large projects that require repeated testing over a certain area. In manual testing, a human is responsible for testing the functionality as a user would use the program. Automated testing automates this process. [3]
- Code Coverage - checks to see if the lines of code are actually executed by the prescribed tests.

8.3 Test Driven Development

Test driven development is an approach to development where tests are carried out on the code at the beginning, throughout, and at the end of the development process.

Tests should be written before starting to write any substantial amount of code, and if it passes the tests, then we start to write our production code. As the code is written, tests are being performed throughout. If the tests succeed, we can continue to write code, if not, we must go back and refactor. This involves cleaning up the code, error checking, and also ensuring clarity throughout. It is also important to develop our tests as we go along by adding test cases for the bugs that are caught, so that the same mistakes are not made twice.

Before finishing off, it is important to clean up the code by going back over it, and making sure to clarify anything that could be construed as an ambiguity in weeks to come. This extra clarity will prove essential to understanding the code that was written so that it can be improved/debugged if needed.

Essentially, test driven development is a strategy for breaking down programming problems into smaller, more approachable chunks so that we are always making small steps forward.

8.4 Precise Language and Communication

Precise language and communication is crucial for data structures. Computers, for the most part, do not make mistakes, humans do. When communicating intentions, you need to be sure that what you say is what someone else is actually hearing. Any resulting output from an executed program bar the desired or expected one is a failure, and can by attributed as a mistake in the programmers thinking.

8.5 Unit Testing - A Closer Look

Unit testing can play a central role in a development strategy that is focused on testing. Your test conditions should be written early, and should evolve and be updated regularly as your core code base changes and develops. There are even benefits in writing the test cases before writing the code. The following are just a few of the benefits of such an approach [4]:

- By starting with unit testing, you are made to clearly define the requirements for your program.
- Unit tests reduce writing too much code. If it passes the test case, the function is complete.
- Unit tests can confirm that new code behaves the same as previous versions.
- Identifies bugs, which makes the debugging process less painful.
- Unit testing verifies the program is working as intended.

Steps to unit test:

1. First define your test case.
2. Check that the output produced matches the expected output.
3. Use assertions for True and False.
4. Check the actual behaviour is the same as the expected behaviour.

8.5.1 Python Unit Test Example

Below is a simple example of running a unit test in Python. To get up and running you need to:

1. Import the unittest module to be able run the unit testing framework.
2. Inherit TestCase's methods as a subclass.
3. Use a method from this class, such as assertTrue() or assertEqual() to test your output.

Listing 1: Python Unit Test Example

```
import unittest

def fun(x):
    return x + 1

class MyTest(unittest.TestCase):
    def test_equal(self):
        self.assertEqual(fun(3), 4)
```

Some points to remember when managing test cases [1]:

- Start the method name with 'test' to make clear these methods are unit tests.
- Organise your tests by placing them into a module[4].

8.6 Object Orientated Programming [5]

Class
Classes can be thought of as blueprints for how objects that are instantiated (which is the creation of an instance of a class) will behave, and their states. They provide structure, not content. E.g. A Card () class might define the behaviours and states of Card objects that are created for a game of Poker.

Object
An object is a specific instance of a class that has actual values. A program can create as many of these instances as needed. E.g. A deck of cards might consist of 52 Card objects.

State
The state of an object is the current values that constitute its attributes. E.g. the rank and suit of a Card object would constitute the state of that card. This would be unique for other Card objects created.

Behaviour
The behaviours of an object are defined in its class' methods, which are made up of functions. They determine the functionality of the class objects. E.g. in a game of Poker, there could be a shuffle() method to randomise the cards.

Mutators and Accessors
Mutators (also known as setters) are methods that allow us to change the state of an object, while accessors (also known as getters) are methods which simply return information about its state. E.g. A shuffle() method would be an example of a mutator, while a getRank() method would be an example of an accessor.

Constructor
Each class will have its own constructor. In Python, this is known as the __int__ method, which is automatically called each time a new instance is created.

Encapsulation
Encapsulation allows us to hide the implementation of functions and data by packing them into a single component. This is important as it means that users cannot attempt to alter the state of the data in a class through the predefined methods.

Everything is a class
In Python, we say that everything is a class because all data types are object data types, meaning that they are defined by classes.

References

[1] Kenneth Reitz, *Testing Your Code* , https://docs.python-guide.org/writing/tests/, Accessed 2019-02-18.

[2] Stack Overflow, *What is a test oracle, and what is it used for?* , https://stackoverflow.com/questions/23522166/what-is-a-test-oracle-and-what-is-it-used-for, Accessed 2019-02-18.

[3] SmartBear, *What is a automated test?* , https://smartbear.com/learn/automated-testing/what-is-automated-testing/, Accessed 2019-02-18.

[4] Dive into Python 3, *Unit Testing* , https://www.cmi.ac.in/ madhavan/courses/prog2-2012/docs/diveintopython3/unit-testing.html, Accessed 2019-02-18.

[5] Real Python, *Object-Oriented Programming (OOP) in Python 3* , https://realpython.com/python3-object-oriented-programming/, Accessed 2019-02-18.