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

Lecture 8: Testing and OOP

Lecturer: Dr. Andrew Hines *Scribes: Fionnuala Wall, Niamh O'Flaherty*

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

8.1 Outline

This lecture discussed the benefits of incorporating a testing strategy into software projects. Once an algorithm and data structure has been coded, it needs to be tested to ensure that it works and works in the intended way. There are many different testing strategies that can be adopted. Test Driven Development is one of these strategies. It involves writing tests before any code is written. Using test driven development can help to make for better design, development and testing. Test Driven Development is discussed in more detail in section 8.3. Unit testing is another testing strategy that can be adopted. It allows for testing of individual components or units of software. It can enable early detection of any bugs that may be in the code. This is discussed further in section 8.4.

It should be noted that while testing can be immensely helpful, it does not replace a good algorithm. The specifications also need to be right. A good data structure needs to be clear, robust, correct, maintainable, secure, of adequate speed and efficient.

The final section, 8.5, discusses the benefits of using precise language and communication to reduce mistakes. This section also includes a glossary of some standard testing terminology and a review of terminology used in Object Orientated Programming.

8.2 Testing Strategies

Testing is used to investigate and evaluate the quality of software. It allows for errors to be identified during the development phase and highlight where a programs performance may be inadequate. Implementing a testing strategy early in development is very important. There are many options when it comes to implementing a testing strategy. Some of the options are outlined below:

- **Regression Testing:** A regression test aims to ensure that changes to the software have not negatively affected it. No new tests are created for regression testing, previously created tests are re-executed to ensure the same functionality is maintained. It is often used to ensure that new versions of a software product perform the same functions as the older version.
- **Unit Testing:** Is the testing of individual units or components of code. A unit should be the smallest testable part of the software. Unit testing is discussed further in section 8.4.
- **Oracles:** A test oracle is used to check if the output of a program is correct using test cases where expected outputs are known. It makes a comparison between actual and expected outputs and determines if it passes or fails. An oracle can be a program, a program specification, a table of examples, a

body of data that specifies the expected output of a set of test data or a programmer's knowledge of how a program should operate.

- **Automated Testing:** Automated testing uses an automation tool to execute test cases. It removes the need for manual testing where possible and can be used to perform tasks that are repetitive, tedious or time consuming. An automatic test can be applied in an oracle testing approach to improve speed and reduce human error.
- **Code Coverage:** Code coverage provides a metric to demonstrate how much code has been tested during the testing phase. Code coverage can look for:
  - Function coverage: to check how many defined functions have been called
  - Statement coverage: how many statements have been executed
  - Branches coverage: how many branches (such as if statements) have been executed
  - Condition coverage: how many Boolean expressions have been checked for both true and false values
  - Line cover: how many lines of code have been tested.

8.3 Test Driven Development

Test driven development (TDD) is an approach to development which requires that tests be written before any code. It follows the process outlined below:

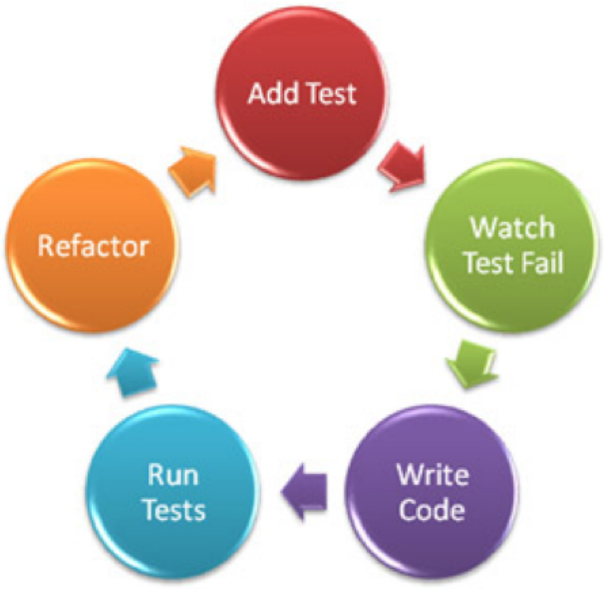

Figure 8.1: Process of test driven development. Taken from https://blog.unif.io/test-driven-development-of-infrastructure-code-9146d3d6c780

1. Every new feature in TDD begins with a test that fails (as there is not yet any code to pass the test). It allows a programmer to understand the required outputs before implementing the code.

2. All of the tests are run to ensure the new test still fails. This ensures the new test won't pass by mistake when code is implemented.
3. Code is then written to fulfil the requirements of the tests and the tests are run again to ensure that they pass.
4. Then any code written should be refactored to remove any duplicates and improve efficiency.
5. The process should be repeated with new tests.

8.3.1 Motivation for using TDD

There are many benefits to using TDD. The approach allows focus on smaller chunks of code as functionality is resolved one test at a time, rather than the whole application which may lead to unfinished code. TDD helps to reduce the number and likelihood of defects as everything is tested in the system as it is built. Having tests in place means that change can be accommodated more easily. Any issues can quickly be identified and rectified.

8.4 Unit Testing Frameworks

8.4.1 Why Unit Test?

Unit testing is an important part of any software project. It helps to improve the quality of the code being delivered by ensuring that bugs are found as early as possible. This in turn can lead to reduced project costs (bugs are typically more expensive to fix when they are discovered later in the Software Development Life Cycle).

When unit tests are automated, they can easily be re-used as regression tests after changes have been made to the program. This means that the existing functionality can be re-tested quickly and easily to ensure that it hasn't been affected by the change.

8.4.2 The Python Framework

`unittest` is a Unit Testing Framework for Python that provides a number of tools for creating and running unit tests.

The following code creates a simple unit test to test the existing `max` function in Python.

```
import unittest

class TestMaxFunction(unittest.TestCase):

    def test_max(self):

        self.assertEqual(max(1,5), 5)
```

The first line of code imports the `unittest` module.

This module provides a class, `TestCase`, that can be used to create unit tests. This is done by defining a new class as a subclass of the `TestCase` class.

The second line of code defines a new class, `TestMaxFunction`, in this way. This new class will inherit all the methods of the `TestCase` class.

The third line of code defines a method called `test_max`. This will test one aspect of Python's `max` function. Numerous methods can be defined within a `TestCase` subclass to test different aspects of functionality. The names of these methods should start with the word 'test' so that other team members can see that the methods are for unit testing.

The last line of code defines the behaviour of the `test_max` method. It makes use of the `assertEqual` method inherited from the `TestCase` class. This method checks for equality between its first and second arguments.

In the above code, the first argument given to the method is `max(1,5) -` this is the condition we want to test. The second argument given to the method is the output we would expect `max(1,5)` to return i.e. `5`.

When the test is run, the `assertEqual` method will compare the output from `max(1,5)` with the expected result that we entered. If the values are equal, then the test will pass. Otherwise, it will fail.

There are a number of different types of assertion that can be performed using the `unittest` module. These are outlined on the Python website (see section 8.6).

8.4.3 Using an IDE

An IDE can be a useful tool for unit testing. IDEs can be used to organise tests within modules and folders. They often also provide additional functionality that can be used to assist with debugging.

For example, a breakpoint can be placed at a specified line of code within a program. It will cause execution of the program to pause when that line of code is reached. The developer can then see the value of different variables at that point of program execution.

8.5 Language and Communication

In general, it's important to use precise language when communicating with team members and stakeholders. This helps to reduce confusion and miscommunication. This section outlines some standard terminology related to testing, and also revises some OOP terminology.

8.5.1 Testing Terminology

Some standard testing terminology is outlined below:

- **Test Case:** A test case consists of a set of input values, a set of execution steps, and a set of expected results. A test case will usually aim to verify a particular requirement or piece of functionality.
- **Test Execution:** This is the process of running a test case and obtaining the test results.
- **Defect:** A fault in a component or system that may cause it to fail to perform its intended function.
- **Fail:** A test is said to fail if the actual result after test execution does not match the expected result.
- **Pass:** A test is said to pass if the actual result after test execution matches the expected result.

8.5.2 OOP Terminology

Some standard OOP terminology is outlined below:

- **Class:** A class is a template for creating an object. It specifies the behaviour and attributes of the newly created object.
- **Object:** An object is a particular instance of a class.
- **Method:** A method is a function associated with a class.
- **Mutator:** A mutator is a method that changes the state of an object.
- **Accessor:** An accessor is a method that returns information about the state of an object.
- **Instantiation:** Instantiation is the creation of an instance of a class.
- **Constructor:** A constructor is a special type of method that is used to instantiate an object.
- **Encapsulation:** Encapsulation is the mechanism of hiding the implementation details of a class. This is done by restricting object access to a set of public methods.
- **Inheritance:** Inheritance occurs when a class is created as a subclass of another class (called a superclass). A subclass will inherit the methods and class attributes of its superclass.
- **Polymorphism:** Polymorphism occurs when a method is designed to have different behaviour when applied to different types of objects.

8.6 References

http://se.ethz.ch/ meyer/publications/testing/principles.pdf

https://docs.python.org/3.4/library/unittest.html

https://www.softwaretestinghelp.com/software-testing-terms-complete-glossary/

http://homepages.uc.edu/ thomam/Misc/OO_Terminology.html

https://en.wikipedia.org/wiki/Breakpoint