Thomas Ngo
Professor Ning Chen
CPSC463-01 Software Testing
Project 1
Due February 20, 2018

Review of Unittest

1. explain in a nutshell what is the motivation of a unit test?

The motivation of a unit test is about reducing bugs in new features, reducing bugs in existing features, creating a good documentation, reducing the cost of change, improving the design, allowing refactoring, constraining features, defending against malicious attacks from other programmers, making programmer becomes more creative, forcing developer to slow down and think, making development faster and reducing fear. Unit test is a way to make sure that individual parts of software or units of source code are working fine as expected. Without the unit test, it is very hard for developer to build new functions for a product since the cost of maintenance is going to be very high.

Generally speaking, unit test helps developers and programmers to reduce bugs. Every time, we write new codes. There always a chance that we may produce a few bugs in our existing code. Those bugs may or may not be seen by our eyes and logic immediately. By writing unit test, we can have a way to catch those bugs and reduce a significant amount of time in debugging while adding new features. This also means unit test can reduce bugs in existing features. With a well-designed unit test code, adding new features rarely breaks any existing functionality. If the new feature that is added to the existing code, existing unit tests give errors. That said we are having a problem with our product in which we have not anticipated yet. Unit test allows us to pinpoint the unseen problem and fix it. Imagining that without unit test, that particular bug can produce more minor bugs and eventually creates a humongous and dramatic issue to the system. In addition, unit test could be a good documentation for the developer to review code and learn faster. For example, when the programmers want to learn to build an API, they are usually searching for the code examples to execute immediately rather than reading a long written documentation. Unit test is one of the best code example that programmers may find and study since they are concise snippets of code that are used to practice public APIs. Moreover, Unit test reduces the cost of change and maintenance. Nowadays, the world of technology is moving so fast that the need to make changes in existing product are inevitable. With that said, making changes also means the cost of change is going up if we do not have the uni testing plan. Having the unit test tool ready to use make it easier to change the software because the developer do not need to worry about the system get broken as new things are added. For instance, if developers have good test coverage for the product, adding new design or ideas will not produce too much bugs. Unit test provides guarantee and confidence for them to explore new functionalities. On the other hand, if the existing product are poorly tested, it will become extremely expensive to change and maintain from time to time. Adding new features to poorly-tested software is equivalent to increase risks in the system. As the system of product becomes more complex, it is more likely that new changes may breaks some parts of product that are used to work before.

Furthermore, unit test helps programmer to improve design, allow refactoring and constrain features. First of all, writing unit test forces programmer to rethink about the design and make his or her code more testable. For example, some programmers are highly relied on dubious patterns such as declaring global variables in which consumes a lot of memory in computer. By writing unit test, they can learn how to create classes more often and utilize classes to create their own advantage to save resources. Note that codes require complex initialization are usually hard to test rather than the ones with less complex initialization. Besides, unit test forms what so called safety net for developers to refactor the code at any time they need. By providing the system with unit test tool, we are giving ourselves more opportunities to change the code over time in order to accommodate the need of customers. Unit test allows us to refactor without breaking existing code so that we can constantly makes change in our design and architecture of the program. Also, most of the time, programmers get used to the way that they are required to build impressive frameworks rather than deliver the requirement customers want. With the test-first approach, developers will begin their job by writing unit tests for current function. Then, they implement the features that are asked by customers. When all unit tests are passed, developers can stop and move on to next feature or next test like integration test. Thus, that demonstrates that unit tests constrain features of product. A well-tested application is far easier to extend. Meaning as developers, we do not have to predict what customer will eventually want.

Finally, unit test defends against other programmers, forces us to slow down and think clearer, makes development faster and reduces fear. How can unit test helps defending against other programmers? Open source is a place where everything can happen at once. Some want to create application that others can use them. Some want to take advantage of that application and modify it into a malicious software so that they can get advantage of it. Therefore, by providing unit test, we can defend our software from unintended purpose of the attacker. Suppose that there is a payroll calculation routine removes a zero from employees' salary in the database. However, the routine of the system only runs at 11:50 PM on New Year's Eve. Now, imagine that the bug fix involves a single line code change. Without the unit test, another programmer may come and change that code. Instead of running the application as usual, the system will be compromised and it cause coutless bounced checks next year. Hence, we can ensure that program or system run properly and it is not broken by providing a decent unit test to informs us about the changes. Additionally, sometimes developers should slow down and think about what they are intending to do. By adding new features and refactoring an existing code, unit test forces developers to think ahead about what the code is supposed to accomplish in the long run. When creating class for unit test, we will think about how to use the public API or a specific function so that we can get the ultimate result should be. That said we are forced to slow down, clean up the code and make a simple design as long as we use unit test. And it is a good thing because with the simple design and cleaned code, we can expect the software to compile properly. Next, unit test can make development faster. Why? Normally, we will think adding class unit test consumes a lot of time to implement and design in the short run. But in the long run, even though it takes time to think about and create good tests, the payback for producing unit test will be promising and worth it. As time goes on, our overall speed of developing will increase since we are not often worrying about break the existing source code when we try to add new features. The last motivation of unit test can bring us is about reducing fear. Making changes to the existing solution and hoping that will not break the system is the biggest fear of the programmers. It will be a nightmare for a program to spend hours and hours in debugging the new features and fixing the codes at the same time. By having a unit test for each individual part, programmers will know exactly where the bug is and how to fix it. They will not waste much of time in figuring out the bugs because unit test guides them to the heart of issue. That means unit test allows programmers to remove the fear of making chances and give them the opportunity to become more creative.

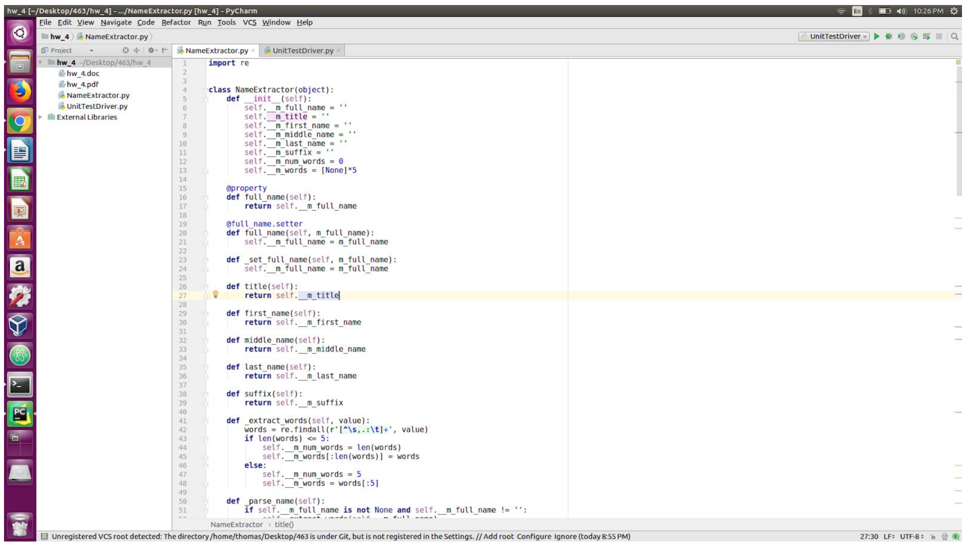
2. explain in general how do you perform a unit test?

    In order to perform a unit test, you have to select a unit testing framework and its programming language. So, what is exactly unit testing framework? It it a layered structure containing many different testing methods or functions that we can use to test each part of source code individually. Framework includes a set of predefined functions and prototypes that developers want to use so that they can perform unit test on a specific function easily. Also, some frameworks even provide users different modules with different associated control data, usage procedures and operating procedure so as to help them determine whatever resources are fit to utilize. For the sake of simplicity, we will use unit testing framework name “unittest” in which is written in Python language to illustrate in general how we perform a unit test.

    First, we need to identify which part of a project that we want to do the unit test. Then we select some or all functions, supposed in a class, to perform unit test on each of them. In addition, we need to create another python script say UnitTest.py to implement the class unit test. Next, we will use that class to write the test cases to check if a particular project unit’s design and functionality works as expected or not. Also, we will need to learn how to run python script in linux using command line.

3. demonstrate the procedure of using the unit test tool

Step 1: Having a project includes some units for the testing.



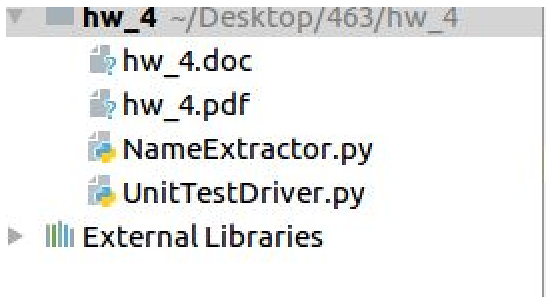
```python
import re


class NameExtractor(object):
    def __init__(self):
        self.__m_full_name = ''
        self.__m_title = ''
        self.__m_first_name = ''
        self.__m_middle_name = ''
        self.__m_last_name = ''
        self.__m_suffix = ''
        self.__m_num_words = 0
        self.__m_words = [None]*5

    @property
    def full_name(self):
        return self.__m_full_name

    @full_name.setter
    def full_name(self, m_full_name):
        self.__m_full_name = m_full_name

    def _set_full_name(self, m_full_name):
        self.__m_full_name = m_full_name

    def title(self):
        return self.__m_title

    def first_name(self):
        return self.__m_first_name

    def middle_name(self):
        return self.__m_middle_name

    def last_name(self):
        return self.__m_last_name

    def suffix(self):
        return self.__m_suffix

    def _extract_words(self, value):
        words = re.findall(r'[^\s,.:\t]+', value)
        if len(words) <= 5:
            self.__m_num_words = len(words)
            self.__m_words[:len(words)] = words
        else:
            self.__m_num_words = 5
            self.__m_words = words[:5]

    def _parse_name(self):
        if self.__m_full_name is not None and self.__m_full_name != '':
```

Unregistered VCS root detected: The directory /home/thomas/Desktop/463 is under Git, but is not registered in the Settings. // Add root Configure Ignore (today 8:55 PM)

27:30 LF÷ UTF-8÷

In the above picture, we are using a project named hw_4. Inside the project, we have an existing source "NameExtractor.py". We are going to perform unit test on that class.

Step 2: Create another python script to import unittest and design class unittest

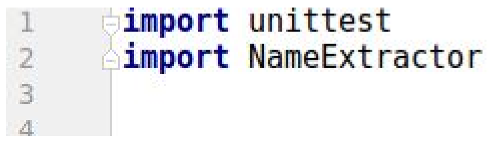


```
hw_4 ~/Desktop/463/hw_4
    hw_4.doc
    hw_4.pdf
    NameExtractor.py
    UnitTestDriver.py
External Libraries
```

A new python script is created and named UnitTestDriver.py. We will use this to implement our unit test class for each individual methods of class NameExtractor

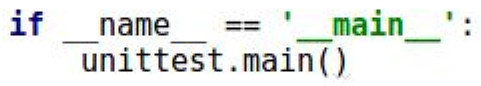
Step 3: Import unittest framework in python 2.7

Use keyword "import" to use the unittest framework and NameExtractor



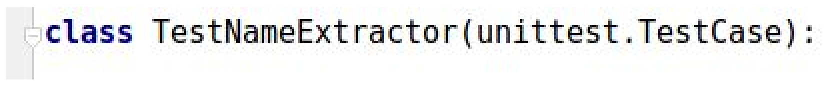
```python
import unittest
import NameExtractor
```

In addition, we need to include the following at the end of the script so as to run unittest directly on IDE



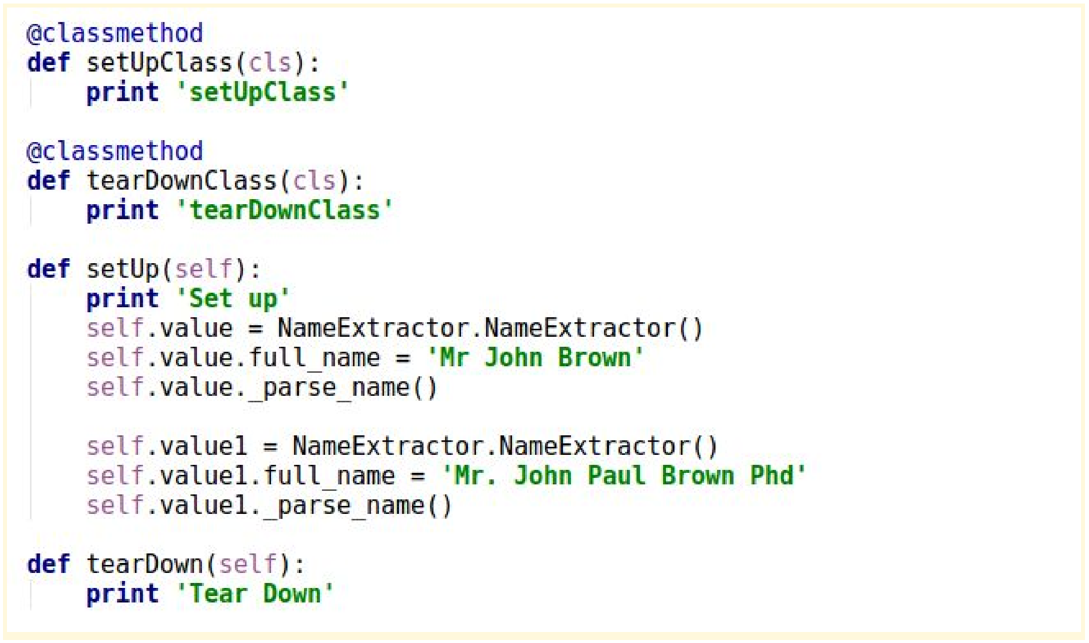
```python
if __name__ == '__main__':
    unittest.main()
```

Step 4: Place unittest.TestCase inside parenthesis of testing class



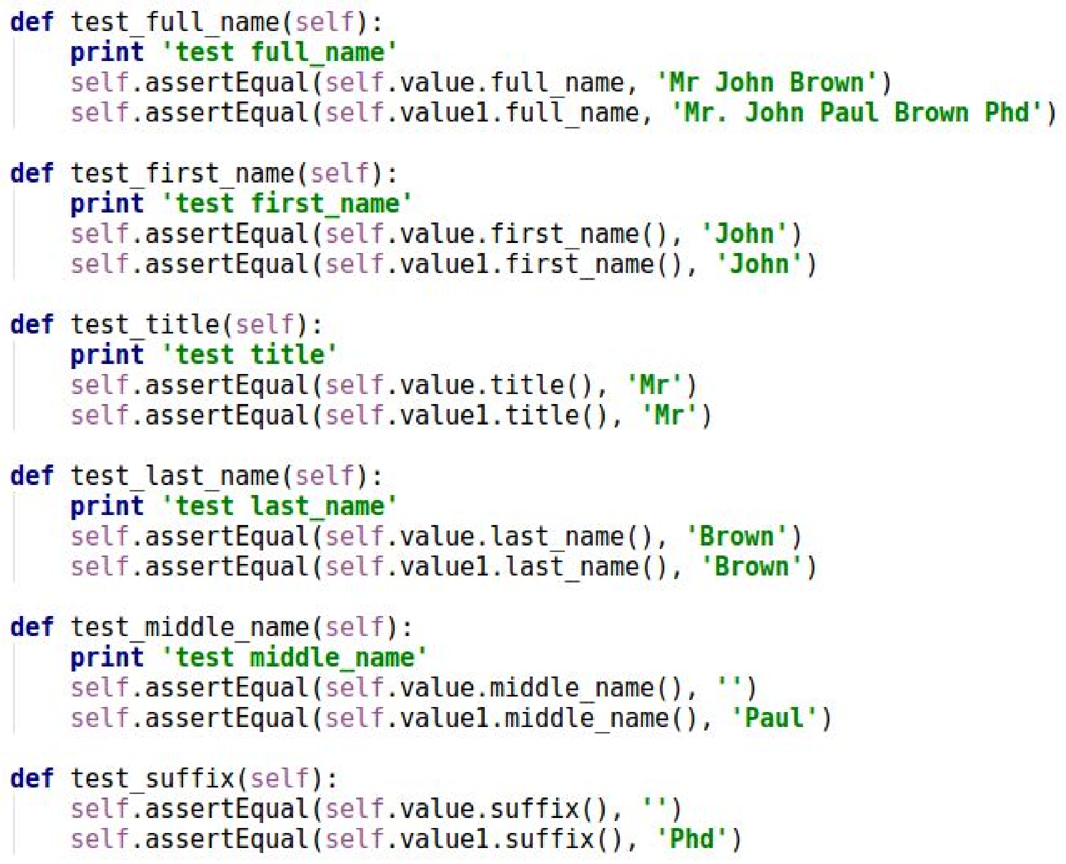
```python
class TestNameExtractor(unittest.TestCase):
```

Step 5: Include the setUp and tearDown in order to provide initialization and cleanup for the fixture. setUp() will run prior to each test and tearDown will invoke after each test



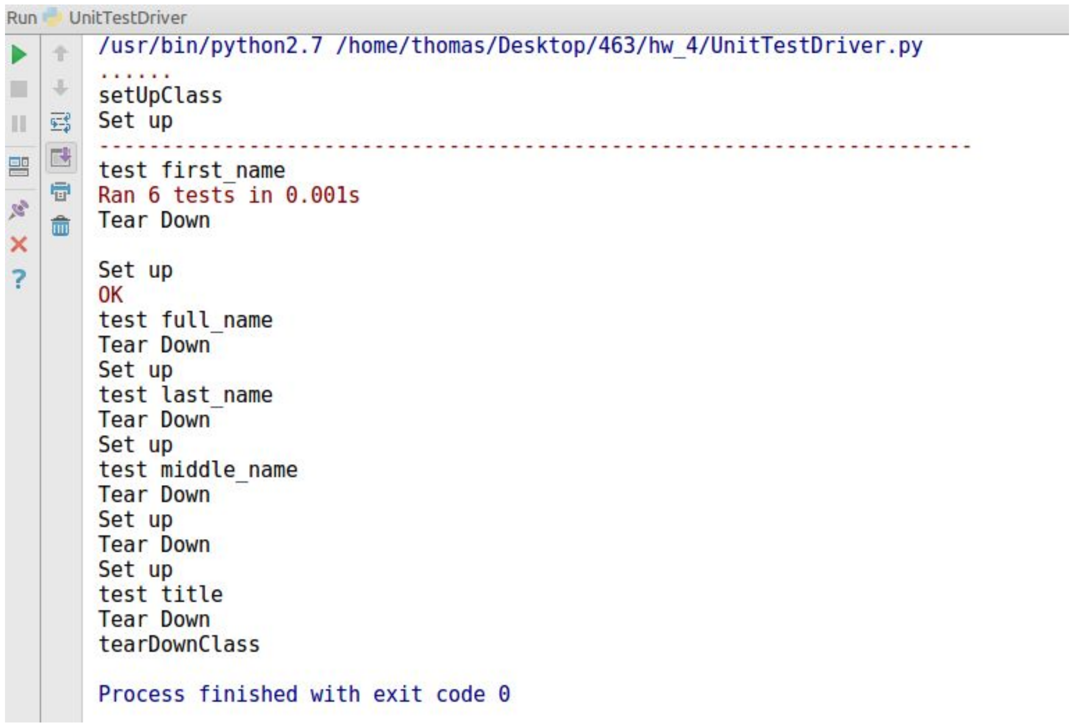
```python
@classmethod
def setUpClass(cls):
    print 'setUpClass'

@classmethod
def tearDownClass(cls):
    print 'tearDownClass'

def setUp(self):
    print 'Set up'
    self.value = NameExtractor.NameExtractor()
    self.value.full_name = 'Mr John Brown'
    self.value._parse_name()

    self.value1 = NameExtractor.NameExtractor()
    self.value1.full_name = 'Mr. John Paul Brown Phd'
    self.value1._parse_name()

def tearDown(self):
    print 'Tear Down'
```

Step 6: Define testing function corresponding to particular units that you want to test its behavior or correctness. Then, Add test case using predefined functions from the framework unittest such as ‘assertEqual’, ‘assertRaises’, etc to perform unit test



```python
def test_full_name(self):
    print 'test full_name'
    self.assertEqual(self.value.full_name, 'Mr John Brown')
    self.assertEqual(self.value1.full_name, 'Mr. John Paul Brown Phd')

def test_first_name(self):
    print 'test first_name'
    self.assertEqual(self.value.first_name(), 'John')
    self.assertEqual(self.value1.first_name(), 'John')

def test_title(self):
    print 'test title'
    self.assertEqual(self.value.title(), 'Mr')
    self.assertEqual(self.value1.title(), 'Mr')

def test_last_name(self):
    print 'test last_name'
    self.assertEqual(self.value.last_name(), 'Brown')
    self.assertEqual(self.value1.last_name(), 'Brown')

def test_middle_name(self):
    print 'test middle_name'
    self.assertEqual(self.value.middle_name(), '')
    self.assertEqual(self.value1.middle_name(), 'Paul')

def test_suffix(self):
    self.assertEqual(self.value.suffix(), '')
    self.assertEqual(self.value1.suffix(), 'Phd')
```

Step 8: Launch unittest in terminal or using IDE like Pycharm to run.



```
/usr/bin/python2.7 /home/thomas/Desktop/463/hw_4/UnitTestDriver.py
......
setUpClass
Set up
----------------------------------------------------------------------
test first_name
Ran 6 tests in 0.001s
Tear Down

Set up
OK
test full_name
Tear Down
Set up
test last_name
Tear Down
Set up
test middle_name
Tear Down
Set up
Tear Down
Set up
test title
Tear Down
tearDownClass

Process finished with exit code 0
```

Or you can use Ctrl + T to open terminal on Ubuntu 16.04 and type the following commands:

```
python -m unittest UnitTestDriver.py
```

4. explain pros/cons of the tool of your choice

Based on what I have experienced with unittest, I think the advantage of this unit testing framework is that it is part of the Python standard library. This makes unittest becomes very friendly to python programmers since there is no need to install any dependencies. On top of that, unittest is very easy to use. It does not take much time for a beginner to catch up and learn all methods in this framework. That said writing actual test is quite easy with unittest.

On the other hand, if we have large project with humongous code base, unittest is not a typical ideal framework to work with. Because it requires users to be tedious in collecting test suites and test cases to do it. Besides, it is also not as easily extensible as other frameworks.

5. summary and your thoughts/insights

In conclusion, practicing in writing unit test becomes a very crucial activity for a programmer or developer. It does not only reduce the cost of maintenance and provide us helpful warning when there are bugs, but also it helps us to become more creative and careful in design our software. In my opinion, applying test-first approach brings many benefits to me as a developer since it forces me to think ahead and slow down my thinking but still pushing the development of product faster.