Individual Assignment: Software Unit Testing Maturity Models

Introduction:

For the purpose of this assignment, I will write a simple test function in, 'Python' using the, 'Unit-test' module to implement a unit test based on the depicted short scenario of:

> 'A customer recently moved funds between her bank accounts. She has a total of $200 in her savings account, and she moved $75 from the savings account to a chequing account with an initial balance of $0. Based on this straight forward function, perform a simple unit test that ensures that the online financial transaction provides the desirable output, i.e.: The savings account now has a balance of $125 after the transaction; The chequing account now has a balance of $75. Write a simple test case or script to execute the above functions'

I will delineate how this is implemented in addition to elucidating on what qualifies it to meet the description of a unit test to highlight the value of this unit test to software development.

Mock-up of Described Source-Code:

Executed in IDLE, Python's Integrated Development and Learning Environment.

```
#Bank Account transfer
# File Name: Bank_transfer_programme.py
#Joseph Karl Magnussen - Student ID: H00069811
# Module: CKIT-535 Week 3 - Individual Assignment – Programme Source-Code

class account:
    def __init__(self, initial_amount):
        self.balance = initial_amount

    def transfer (self, amount, target_acc):
        if self.balance < amount:
            print ("Sorry, you have insufficient funds.")
            return False
        elif amount < 0:
            print ("You cannot take money from other accounts")
            return False
        else:
            self.balance -= amount
            target_acc.balance += amount
            return True
```

My Unit Test Depiction:

Executed in IDLE, Python's Integrated Development and Learning Environment.

```
# File Name: Unit_test.py
#Joseph Karl Magnussen - Student ID: H00069811
# Module: CKIT-535 Week 3 - Individual Assignment - Unit Testing Code

import unittest
import Bank_transfer_programme as btp

class KnownValues (unittest.TestCase):
    def test_acceptable_transfer (self):
        savings_account = btp.account (200)
        Cheq_acc = btp.account (0)
        savings_account.transfer (75, Cheq_acc)
        self.assertEqual (savings_account.balance, 125)
        self.assertEqual (Cheq_acc.balance, 75)

    def test_wrong_transfer (self):
        savings_account = btp.account (200)
        Cheq_acc = btp.account (0)
        savings_account.transfer (1000, Cheq_acc)
        self.assertEqual (savings_account.balance, 200)
        self.assertEqual (Cheq_acc.balance, 0)

    def test_exact_transfer (self):
        savings_account = btp.account (200)
        Cheq_acc = btp.account (0)
        savings_account.transfer (200, Cheq_acc)
        self.assertEqual (savings_account.balance, 0)
        self.assertEqual (Cheq_acc.balance, 200)

    def test_negative_transfer (self):
        savings_account = btp.account (200)
        Cheq_acc = btp.account (0)
        savings_account.transfer (-200, Cheq_acc)
        self.assertEqual (savings_account.balance, 200)
        self.assertEqual (Cheq_acc.balance, 0)

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

What qualifies this to meet the description of a unit test, other than the, 'Unit-test' module which offers an array of tools for developing and implementing tests, is its expanse in relation to the source code. The intent of which is to test the source code to ensure that it operates as required. An example of which would be the first parameter of, 'Unit_test.py' within the, 'test_acceptable_transfer' function, designed to test the programme against the $200 in savings account > move $75 from the savings account to a chequing account with an initial balance of $0 > The savings account now has a balance of $125 after the transaction scenario:

```
def transfer(self, amount, target_acc):
    if self.balance < amount:
        print ("Sorry, you have insuficcient funds.")
        return False
    elif amount < 0:
        print("You can not steal money from other accounts")
        return False
    else:
        self.balance -= amount
        target_acc.balance += amount
        return True
```

$200 - $75 = $125

$0 + $75 = $75

```
class KnownValues (unittest.TestCase):
    def test_acceptable_transfer(self):
        savings_account = btp.account(200)
        Cheq_acc = btp.account(0)
        savings_account.transfer(75, Cheq_acc)
        self.assertEqual(savings_account.balance, 125)
        self.assertEqual(Cheq_acc.balance, 75)
```

The test code titled, 'test_acceptable_transfer' shown above on the right tests the source code on the left against the exampled procedure, transferring $75 from a $200 savings account to the $0 chequing account and changing the saving account's balance to $125 and the chequing account to $75. What qualifies it to meet the description of a thorough unit test the depth and scrutiny of the trial that the source code is measured against. I have executed several tests exampled as objects of the, 'KnownValues' class which example the kind of questions one may ponder upon a unit test such as this:

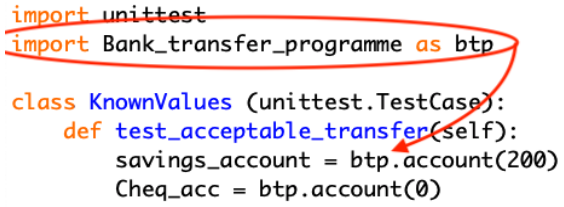
```
    def test_wrong_transfer (self):
        savings_account = btp.account(200)
        Cheq_acc = btp.account(0)
        savings_account.transfer(1000, Cheq_acc)
        self.assertEqual(savings_account.balance, 200)
        self.assertEqual(Cheq_acc.balance, 0)

    def test_exact_transfer (self):
        savings_account = btp.account(200)
        Cheq_acc = btp.account(0)
        savings_account.transfer(200, Cheq_acc)
        self.assertEqual(savings_account.balance, 0)
        self.assertEqual(Cheq_acc.balance, 200)

    def test_negative_transfer (self):
        savings_account = btp.account(200)
        Cheq_acc = btp.account(0)
        savings_account.transfer(-200, Cheq_acc)
        self.assertEqual(savings_account.balance, 200)
        self.assertEqual(Cheq_acc.balance, 0)

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

Let's create a test that checks whether we can transfer more than we have available in the account

Let's create a test that checks whether we are able to transfer the exact amount that's available in the account

Let's create a test that checks whether we are able to make a negative transfer

In efforts to be completely thorough on this process, I would like to add that the source code has been tethered to the unit test through an import and located within the same folder location as the unit test. Furthermore, I have given the source code import an alias of, 'btp' as an acronym for, 'bank transfer procedure' in efforts to mitigate the complexity of the unit test. To highlight the value of this unit test to software development, it should be noted that the process of implementing said test gives credence to a more critical and comprehensive thought process in relation to the source code. For example, writing the unit test caused me to augment my source code to include the, 'elif amount < 0:' procedure within the, 'transfer' protocol and successive, 'test_negative_transfer' within the unit test. Unit testing is a supplementary measure that promotes a thought process that's conducive to the robustness of the source code, bestowing a more mindful perception of the source code's behaviour and limitations on the developer. Finally, upon completion, the test will elucidate on the functionality of the test with a binary variable output of wat effectively pertains to pass or fail:




```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/josephmagnussen/Documents/The University of Liverpool/PGC in Sof
tware Engineering/CKIT - 535 Software Testing & Qual Assura/Week 3- Development T
esting/Individual assignment/Unit Test Project/Unit_test.py
..You can not steal money from other accounts
.Sorry, you have insuficcient funds.
.
----------------------------------------------------------------------
Ran 4 tests in 0.164s

OK
>>>
```

The 4 tests have been executed with no issues found.

Source code and unit test screenshots:

```python
#Bank Accont transfer
#Joseph Karl Magnussen - Student ID: H00069811
# Module: CKIT-535 Week 3 - Individual Assignment - Unit Testing

class account:
    def __init__(self, initial_amount):
        self.balance = initial_amount

    def transfer(self, amount, target_acc):
        if self.balance < amount:
            print ("Sorry, you have insuficcient funds.")
            return False
        elif amount < 0:
            print("You can not steal money from other accounts")
            return False
        else:
            self.balance -= amount
            target_acc.balance += amount
            return True
```

```python
#Bank Accont transfer
#Joseph Karl Magnussen - Student ID: H00069811
# Module: CKIT-535 Week 3 - Individual Assignment - Unit Testing

import unittest
import Bank_transfer_programme as btp

class KnownValues (unittest.TestCase):
    def test_acceptable_transfer(self):
        savings_account = btp.account(200)
        Cheq_acc = btp.account(0)
        savings_account.transfer(75, Cheq_acc)
        self.assertEqual(savings_account.balance, 125)
        self.assertEqual(Cheq_acc.balance, 75)

    def test_wrong_transfer (self):
        savings_account = btp.account(200)
        Cheq_acc = btp.account(0)
        savings_account.transfer(1000, Cheq_acc)
        self.assertEqual(savings_account.balance, 200)
        self.assertEqual(Cheq_acc.balance, 0)

    def test_exact_transfer (self):
        savings_account = btp.account(200)
        Cheq_acc = btp.account(0)
        savings_account.transfer(200, Cheq_acc)
        self.assertEqual(savings_account.balance, 0)
        self.assertEqual(Cheq_acc.balance, 200)

    def test_negative_transfer (self):
        savings_account = btp.account(200)
        Cheq_acc = btp.account(0)
        savings_account.transfer(-200, Cheq_acc)
        self.assertEqual(savings_account.balance, 200)
        self.assertEqual(Cheq_acc.balance, 0)

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