Software Testing Online Test – Winter 2021

Question 1

Version 1

During the lectures, we talked about a set of testing principles. Consider the following scenario, and explain what testing principles has been followed, and what testing principles have been violated. For each case, you need to **mention the testing principle,** and **explain how it has been followed/violated.** If no explanation is provided, no mark will be given.

David works as a software developer in BankToGo. He has just finished developing code to automate the process of withdrawing money from an Automated Teller Machine (ATM). Such code needs to be tested. Hence, David created JUnit tests that covered two scenarios:

(i) Allowing a customer to withdraw 40 dollars from his account that has a 100 dollars balance.

```
public void testWithdraw_scenario1()
{
      Customer John = new Customer(100);
      //where 100 is the initial balance

      int newBalance = John.withdraw(40);
      assertEquals(newBalance, 60);
}
```

(ii) Allowing a customer to withdraw 100 dollars from his account that has a 100 dollars balance.

```
public void testWithdraw_scenario2()
{
      Customer John = new Customer(100);
      //where 100 is the initial balance

      John.withdraw(100);
}
```

Followed principles:

- Used JUnit which is a reusable approach to write tests.

Violated principles:

- He wrote the source code, and tested for it. This violates the principle that one should not test his own code.
- He wrote test cases for the valid and expected inputs (i.e., no handling for negative amounts, or for withdrawing extra money for instance). This violates the principle that test cases should consider both the valid and invalid inputs/cases.

- The absence of any definition/assertion of the expected result within the second test case.

Version 2

During the lectures, we talked about a set of testing principles. Consider the following scenario, and explain what testing principles has been followed, and what testing principles have been violated. For each case, you need to **mention the testing principle,** and **explain how it has been followed/violated.** If no explanation is provided, no mark will be given.

BankHelper is some software that is being developed by a set of developers including *Nader*. The software automates money withdrawal from a user's account, money deposit into a user's account, and money transfer from one user account to another user's account. *Nader* has just finished developing the money transfer functionality, and has created the JUnit tests that cover two scenarios, as follows:

i. Allowing a customer to transfer 70 dollars from his account that has a 100 dollars balance, to another customer whose balance is 100 dollars.

```
public void testTransfer_scenario1(){
          Customer John = new Customer(100);//100 is the initial
          balance
          Customer Dave = new Customer(100);//100 is the initial
          balance
          John.transfer(70,Dave);
          assertEquals(Dave.getBalance(),170);/*getBalance()
          returns the current balance of a customer*/
}
```

ii. Allowing a customer to transfer 50 dollars from his account that has a 100 dollars balance, to another customer whose balance is 0 dollars.

```
public void testTransfer_scenario2(){
     Customer John = new Customer(100);//100 is the initial
     balance
     Customer Dave = new Customer(0);//100 is the initial
     balance
     John.transfer(50,Dave);
}
```

Followed principles:

• Used JUnit which is a reusable approach to write tests.

Violated principles:

• He wrote the source code, and tested for it. This violates the principle that one should not test his own code.

• He wrote test cases for the valid and expected inputs (i.e., no handling for negative amounts, or for withdrawing extra money for instance). This violates the principle that test cases should consider both the valid and invalid inputs/cases.

• The absence of any definition/assertion of the expected result within the second test case.

Question 2

Version 1

Consider a SuperList class that can hold objects:

- public SuperList (int size); // The highest number of allowed elements
- public boolean isEmpty ();
- public boolean isFull ();
- public void insertAtTheEnd (Object X); //adds an object at the end of the list
- public Object removeFromTheEnd (); //removes an object from the end of the list

For example, if the list has the content {"Hi","there"}, and you called insertAtTheEnd("Man"), the content will be {"Hi","there","Man"}

For example, if the list has the content {"Hi","there"}, and you called removeFromTheEnd(), the content will be {"Hi"}

(a) Identify all the variables to consider for input space partitioning.
The 3 variables are: The superlist object, the size argument that represents the size of the list, Object X that would inserted in the Super list.

(b) Identify characteristics that suggest partitions and define the blocks in the partition for each characteristic. Choose the smallest possible number of partitions and blocks.

The partitions and blocks are as follows:

- Whether list is empty.
  a1:true
  a2:false **Base**
- Whether list is full.
  b1:true
  b2:false **Base**
- Current Size of list.
  c1:0
  c2:1
  c3:more than 1 **Base**
- Value of size (where size refers to the highest number of allowed elements).
  d1:negative
  d2:0
  d3:1
  d4:more than 1 **Base**
- Whether x is null.
  f1:true
  f2:false **Base**

(c) Define a test set that satisfies Base Choice (BCC) coverage for the insertAtTheEnd(); only the test set is needed, not the test values.

- Marking one block **for each** of the above partitions as the base choice (as done using the green word **Base**)
- One test case would include all the base choices combined (a2, b2, c3, d4, f2)
- The eight variations are:
  (a1; b2; c3; d4; f2);
  (a2; b1; c3; d4; f2);
  (a2; b2; c1; d4; f2);
  (a2; b2; c2; d4; f2);
  (a2; b2; c3; d1; f2);
  (a2; b2; c3; d2; f2);
  (a2; b2; c3; d3; f2);
  (a2; b2; c3; d4; f1)

Version 2

Consider a DuperList class that can hold objects:

- public DuperList ( );
- public boolean isEmpty ();

- public void insertAtTheFront(Object X); //adds an object at the beginning of the list
- public Object removeFromTheFront(); //removes an object from the beginning of the list

For example, if the list has the content {"Hi","there"}, and you called insertAtTheFront("Man"), the content will be {"Man","Hi","there"}

For example, if the list has the content {"Hi","there"}, and you called removeFromTheFront(), the content will be {"there"}

(a) Identify all the variables to consider for input space partitioning.
The 3 variables are: The duperlist object, the size argument that represents the size of the list, Object X that would inserted in the duper list.

(b) Identify characteristics that suggest partitions and define the blocks in the partition for each characteristic. Choose the smallest possible number of partitions and blocks.

The partitions and blocks are as follows:

- Whether list is empty.
  a1:true
  a2:false **Base**
- Current Size of list.
  c1:0
  c2:1
  c3:more than 1 **Base**
- whether the list contains null entries
  d1:true
  d2:false **Base**

- Whether x is null.
  f1:true
  f2:false **Base**

(c) Define values for the blocks
For example, whether the list contains null entries
true (Possible list values [null, "cat", null]
false (Possible list values ["cat", "hat", "rat"])

(d) Define a test set that satisfies Base Choice (BCC) coverage for the insertAtTheFront method; only the test set is needed, not the test values.
Marking one block **for each** of the above partitions as the base choice (as done using the green word **Base**)

- One test case would include all the base choices combined (a2, c3, d2, f2)
  Mentioning correct base choice variations

Question 3

Version 1

Consider the following source code:

```
int a = b;
while (a < 100){
    if (a < b)
    {
        a += 1;
        break;
    }
    for(int c = 1; c< a; c++)
        a = a + c;

    if (a > 5)
        b += 1;
    else
        b += 2;
}
print(a, b);
```

(a) Draw the control flow graph of the above source code.
(b) Decorate the control flow graph with all the Def-use data for all the variables.
(c) Write down the DU-paths of for all the variables.
(d) What does infeasible test path mean? Illustrate using an example from the above source code.

Version 2

```
public int myMethod(int a) {
    int c = 0;
    while(c<15) {
        for(int d=0; d<a; d++) {
            if( d % 2 == 0) {//The % operator is the remainder after division operator
```

```
                    c = c + d;
                }
                else {
                    c = c + 3;
                }
            }
        }

        return(c);
}
```

a) Draw the control flow graph of the above source code.
b) Show the test requirements for edge coverage.
c) Create a set of test cases that achieve edge coverage.
d) Show the test requirements for prime path coverage.