Software Qyality Engineering

11 Nov, 2020

Assignment # 1, 2 &3

Abdullah chand BSE161027 Muhammad Ismail BSE173153

Table of Contents

Case study introduction ………………………………………………………………………3

Function 1 ……..….…………………………………………………………………….………….3

Function 2………..……………………………………………………………………….…………4

Function 3……………………………………………………………………………………………5

Strong Robust EC……………………………………..………………………………………….9

Function 1…..……………………………………………………………………………………….9

Function 2………………………………………………………………………………….……….10

Function 3 ………………………………………………………………………………………….11

Assignment 3……………………………………………………………………………..……… 13

Graphs …………………………………………....………………………………………………. 13

Decision Table …………………….……………………………………………………………. 14

Test cases ……………………………………………………………………….…….………….. 15

Graph for assignment 2 ………………………………………………………………..….. 16

Changes in Assignment 1 …………………………………………………………………..17

GitHub link of Assignment………………………………………………………………….18

**Banks Management System Case Study:**

An organization consists of five banks that operates an ATM system. Each bank holds many accounts and one account for each customer. An account will be accessed through cash card. A customer may have one or more cash cards. The main course of ATM system is to carry out the withdrawing and depositing transactions. Transaction will be carried out through a specific account. Before transaction bank will approve the credentials. When the transaction will be executed it will either do it successfully or it will give a denial message. Executing a transaction will occur in two steps : checking the account and processing the execution. The account checking process will result the approval for processing the transaction. If approval is denied the ATM will issue a denial notice else the transaction will take place using the account number. The checking of the account starts as the ATM validates the cash card details with the organization and the respective bank. If the card is valid, password is correct then the transaction will take place. the management of the banks will be provided by the organization and will follow the standard rules. All the credentials will be checked by the organization and will be matched with the credentials provided to the bank. If somehow any record mismatches or the record is missing, organization will recover the loss. Organization will take data from the bank details and add to its record.

**Functions:**

Balance-check (int accountnum )

let's assume we have 100 accounts in bank min account 1 and maximum is 100

**1 2 50 99 100**

**min=1**

**min+1= 2**

**normal =50**

**max-1 =99**

**max=100**

**n=1 so that according to worst case**

**5n= 5**

**total test cases =5**

**test no 1= Balaance\_check(1)**

**test no 2= Balaance\_check(2)**

**test no 3= Balaance\_check(50)**

**test no 4= Balaance\_check(99)**

**test no 5= Balaance\_check(100)**

**2nd function**

**void deposit(int account number ,double amount)**

let's assume we have 100 accounts in bank min account 1 and maximum i s 100

and amount we assume is min =1000 and max=50000

account number

**1 2 50 99 100**

**min=1**

**min+1= 2**

**normal =50**

**max-1 =99**

**max=100**

Amount

**1000 1001 25000 49999 50000**

**min=1000**

**min+1= 1001**

**normal =25000**

**max-1 =49999**

**max=50000**

**According to Worst case 5^2=25**

**test no 1=deposit(1,1000)**

**test no 2=deposit(1,1001)**

**test no 3=deposit(1,25000)**

**test no 4=deposit(1,49999)**

**test no 5=deposit(1,50000)**

**test no 6=deposit(2,1000)**

**test no 7=deposit(2,1001)**

**test no 8=deposit(2,25000)**

**test no 9=deposit(2,49999)**

**test no 10=deposit(2,50000)**

**test no 11=deposit(50,1000)**

**test no 12=deposit(50,1001)**

**test no 13=deposit(50,25000)**

**test no 14=deposit(50,49999)**

**test no 15=deposit(50,50000)**

**test no 16=deposit(99,1000)**

**test no 17=deposit(99,1001)**

**test no 18=deposit(99,25000)**

**test no 19=deposit(99,49999)**

**test no 20=deposit(99,50000)**

**test no 21=deposit(100,1000)**

**test no 22=deposit(100,1001)**

**test no 23=deposit(100,25000)**

**test no 24=deposit(100,49000)**

**test no 25=deposit(100,50000)**

**3rd function**

**withdraw(int accname ,char name, double amount)**

**account number i smin 1 and max 100**

**name is min 6 and max 18 character**

**amount min 1000 and max 50000**

account number

**1 2 50 99 100**

**min=1**

**min+1= 2**

**normal =50**

**max-1 =99**

**max=100**

Amount

**1000 1001 25000 49999 50000**

**min=1000**

**min+1= 1001**

**normal =25000**

**max-1 =49999**

**max=50000**

**max=100**

name

**6 7 9 11 12**

**min=6**

**min+1= 7**

**normal =9**

**max-1 =11**

**max=12**

**According to Worst case 5^3=125**

**Test no 1= withdraw(1,6,100)**

**Test no2 = withdraw(1,7,1000)**

**Test no 3= withdraw(1,9,1000)**

**Test no4= withdraw(1,11,1000)**

**Test no5= withdraw(1,12,1000)**

**Test no6 = withdraw(2,7,1000)**

**Test no 7= withdraw(2,9,1000)**

**Test no8= withdraw(2,11,1000)**

**Test no9= withdraw(2,12,1000)**

**Test no 10= withdraw(2,6,1000)**

**Test no11 = withdraw(50,6,1001)**

**Test no 12= withdraw(50,7,1001)**

**Test no 13= withdraw(50,9,1001)**

**Test no 14= withdraw(50,11,1001)**

**Test no 15= withdraw(50,12,1001)**

**Test no 16= withdraw(2,6,1001)**

**Test no 17= withdraw()2,7,1001**

**Test no 18= withdraw(2,9,1001)**

**Test no 19= withdraw(2,11,1001)**

**Test no 20= withdraw(2,12,1001)**

**Test no21 = withdraw(1,6,1001)**

**Test no 22= withdraw(1,7,1001)**

**Test no 23= withdraw(1,9,1001)**

**Test no 24= withdraw(1,11,1001)**

**Test no 25= withdraw(1,12,1001)**

**Test no226 = withdraw(100,6,1000)**

**Test no 27= withdraw(100,7,1000)**

**Test no 28= withdraw(100,9,1000)**

**Test no 29= withdraw(100,11,1000)**

**Test no 30= withdraw(100,12,1000)**

**Test no 31= withdraw(99,6,1000)**

**Test no 32= withdraw(99,6,1000)**

**Test no 33= withdraw(99,6,1000)**

**Test no 34= withdraw(99,6,1000)**

**Test no 35= withdraw(99,6,1000)**

**Test no 36= withdraw(50,6,1000)**

**Test no 37= withdraw(50,7,100)**

**Test no 38= withdraw(50,9,1000)**

**Test no 39= withdraw(5,11,1000)**

**Test no 40= withdraw(5,12,1000)**

**Test no 41= withdraw(99,6,1001)**

**Test no 42= withdraw(99,7,1001)**

**Test no 43= withdraw(99,9,1001)**

**Test no 44= withdraw(99,11,1001)**

**Test no 45= withdraw(99,12,1001)**

**Test no46 = withdraw(100,6,1001)**

**Test no 47= withdraw(100,7,1001)**

**Test no 48= withdraw(100,9,1001)**

**Test no 49= withdraw(100,11,1001)**

**Test no 50= withdraw(100,12,1001)**

**Test no51 = withdraw(20,12,100)**

**Test no52 = withdraw(20,12,1001)**

**Test no53 = withdraw(20,12,25000)**

**Test no54 = withdraw(20,12,49999)**

**Test no55 = withdraw(20,12,50000)**

**Strong Robust Equivalence Class Partitioning :-**

Strong robust testing is done by examining not only valid but also invalid inputs.

We go beyond the boundaries.

**Strong Robust for Function 1:**

**Balance\_check(**int accountnum**);**

Let’s assume that we have 50 accounts.

**Test cases will be within the limits and we will also take values out of boundaries.**

**Test cases:**

Test case 1: Balance\_check(-1)

Test case 2: Balance\_check(0)

Test case 3: Balance\_check(1)

Test case 4: Balance\_check(25)

Test case 5: Balance\_check(49)

Test case 6: Balance\_check(50)

Test case 1: Balance\_check(51)

**Strong Robust for Function 2:**

**Void deposit(**int accountnumber,double amount**):**

Let’s assume that account numbers are from 1 to 50 and amount is also from 1 to 50.

**Test cases:**

Test case 1: Void deposit(0,0)

Test case 2: Void deposit(0,1)

Test case 3: Void deposit(0,2)

Test case 5: Void deposit(0,20)

Test case 5: Void deposit(0,49)

Test case 6: Void deposit(0,50)

Test case 7: Void deposit(0,51)

Test case 8: Void deposit(1,0)

Test case 9: Void deposit(1,1)

Test case 10: Void deposit(1,2)

Test case 11: Void deposit(1,5)

Test case 12: Void deposit(1,49)

Test case 13: Void deposit(1,50)

Test case 14: Void deposit(1,51)

Test case 15: Void deposit(15,0)

Test case 16: Void deposit(15,1)

Test case 17: Void deposit(15,2)

Test case 18: Void deposit(15,20)

Test case 19: Void deposit(15,49)

Test case 20: Void deposit(15,50)

Test case 21: Void deposit(15,51)

Test case 22: Void deposit(49,0)

Test case 23: Void deposit(49,1)

Test case 24: Void deposit(49,2)

Test case 25: Void deposit(49,35)

Test case 26: Void deposit(49,49)

Test case 27: Void deposit(49,50)

Test case 28: Void deposit(49,51)

Test case 29: Void deposit(50,0)

Test case 30: Void deposit(50,1)

Test case 31: Void deposit(50,2)

Test case 32: Void deposit(50,5)

Test case 33: Void deposit(50,49)

Test case 34: Void deposit(50,50)

Test case 35: Void deposit(50,51)

Test case 36: Void deposit(51,0)

Test case 37: Void deposit(51,1)

Test case 38: Void deposit(51,2)

Test case 39: Void deposit(51,45)

Test case 40: Void deposit(51,49)

Test case 41: Void deposit(51,50)

Test case 42: Void deposit(51,51)

Test case 43: Void deposit(2,0)

Test case 44: Void deposit(2,1)

Test case 45: Void deposit(2,2)

Test case 46: Void deposit(2,40)

Test case 47: Void deposit(2,49)

Test case 48: Void deposit(2,50)

Test case 49: Void deposit(2,51)

**Strong Robust for Function 3:**

**Withdraw**(account\_id,serial\_no,amount)

Let’s assume we have 50 account ids and amount is also 50.

Name uses characters.(only two test cases: abc,123)

**Test cases:**

Test case 1: withdraw(0,abc,0)

Test case 2: withdraw(0,abc,1)

Test case 3: withdraw(0,abc,2)

Test case 4: withdraw(0,abc,30)

Test case 5: withdraw(0,abc,49)

Test case 6: withdraw(0,abc,50)

Test case 7: withdraw(0,abc,51)

Test case 8: withdraw(1,abc,0)

Test case 9: withdraw(1,abc,1)

Test case 10: withdraw(1,abc,2)

Test case 11: withdraw(1,abc,40)

Test case 12: withdraw(1,abc,49)

Test case 13: withdraw(1,abc,50)

Test case 14: withdraw(1,abc,51)

Test case 15: withdraw(2,abc,0)

Test case 16: withdraw(2,abc,1)

Test case 17: withdraw(2,abc,2)

Test case 18: withdraw(2,abc,35)

Test case 19: withdraw(2,abc,49)

Test case 20: withdraw(2,abc,50)

Test case 21: withdraw(2,abc,51)

Test case 22: withdraw(30,abc,0)

Test case 23: withdraw(30,abc,1)

Test case 24: withdraw(30,abc,2)

Test case 25: withdraw(30,abc,45)

Test case 26: withdraw(30,abc,49)

Test case 27: withdraw(30,abc,50)

Test case 28: withdraw(30,abc,51)

Test case 29: withdraw(49,abc,0)

Test case 30: withdraw(49,abc,1)

Test case 31: withdraw(49,abc,2)

Test case 32: withdraw(49,abc,25)

Test case 33: withdraw(49,abc,49)

Test case 34: withdraw(49,abc,50)

Test case 35: withdraw(49,abc,51)

Test case 36: withdraw(50,abc,0)

Test case 37: withdraw(50,abc,1)

Test case 38: withdraw(50,abc,2)

Test case 39: withdraw(50,abc,35)

Test case 40: withdraw(50,abc,49)

Test case 41: withdraw(50,abc,50)

Test case 42: withdraw(50,abc,51)

Test case 43: withdraw(51,abc,0)

Test case 44: withdraw(51,abc,1)

Test case 45: withdraw(51,abc,2)

Test case 46: withdraw(51,abc,25)

Test case 47: withdraw(51,abc,49)

Test case 48: withdraw(51,abc,50)

Test case 49: withdraw(51,abc,51)

Test case 50: withdraw(0,11,0)

Test case 51: withdraw(0,11,1)

Test case 52: withdraw(0,11,2)

Test case 53: withdraw(0,11,30)

Test case 54: withdraw(0,11,49)

Test case 55: withdraw(0,11,50)

Test case 56: withdraw(0,11,51)

Test case 57: withdraw(1,11,0)

Test case 58: withdraw(1,11,1)

Test case 59: withdraw(1,11,2)

Test case 60: withdraw(1,11,40)

Test case 61: withdraw(1,11,49)

Test case 62: withdraw(1,11,50)

Test case 63: withdraw(1,11,51)

Test case 64: withdraw(2,11,0)

Test case 65: withdraw(2,11,1)

Test case 66: withdraw(2,11,2)

Test case 67: withdraw(2,11,35)

Test case 68: withdraw(2,11,49)

Test case 69: withdraw(2,11,50)

Test case 70: withdraw(2,11,51)

Test case 71: withdraw(30,1,0)

Test case 72: withdraw(30,11,1)

Test case 73: withdraw(30,11,2)

Test case 74: withdraw(30,11,45)

Test case 75: withdraw(30,11,49)

Test case 76: withdraw(30,11,50)

Test case 77: withdraw(30,11,51)

Test case 78: withdraw(49,11,0)

Test case 79: withdraw(49,11,1)

Test case 80: withdraw(49,11,2)

Test case 81: withdraw(49,11,25)

Test case 82: withdraw(49,11,49)

Test case 83: withdraw(49,11,50)

Test case 84: withdraw(49,11,51)

Test case 85: withdraw(50,11,0)

Test case 86: withdraw(50,11,1)

Test case 87: withdraw(50,11,2)

Test case 88: withdraw(50,11,35)

Test case 89: withdraw(50,11,49)

Test case 90: withdraw(50,11,50)

Test case 91: withdraw(50,11,51)

Test case 92: withdraw(51,11,0)

Test case 93: withdraw(51,11,1)

Test case 94: withdraw(51,11,2)

Test case 95: withdraw(51,11,25)

Test case 96: withdraw(51,11,49)

Test case 97: withdraw(51,11,50)

Test case 98: withdraw(51,11,51)

**Assignment # 3**

**Causes Effects**

C1: Phone Number E1: Registration

C2: Account Number E2: Login

C3: Password E3: Confirm Withdrawal

C4: Withdraw Amount E4: Confirm Transaction

C5: Transaction

**Graphs:-**

**Decision Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test cases** | **T1** | **T2** | **T3** | **T4** |
| C1:Phone Number | **1** | **0** | **0** | **0** |
| C2: Account Number | **0** | **1** | **0** | **0** |
| C3: Password | **1** | **1** | **1** | **1** |
| C4: Withdraw Amount | **0** | **0** | **1** | **0** |
| C5: Transaction | **0** | **0** | **0** | **1** |
| E1: Registration | **1** | **0** | **0** | **0** |
| E2: Login | **0** | **1** | **0** | **0** |
| E3: Confirm deposit | **0** | **0** | **1** | **0** |
| E4: Confirm transaction | **0** | **0** | **0** | **1** |

**Identifying Test Cases:-**

|  |  |  |
| --- | --- | --- |
| **Test cases** | **Input(causes)** | **Expected output(effects)** |
| **1** | **Phone Number & Password** | **Registered** |
| **2** | **Account Number & Password** | **Login** |
| **3** | **Withdraw amount & password** | **Confirm withdrawal** |
| **4** | **Transaction & password** | **Confirm transaction** |

**Test Cases:**

We will use boundary value analysis to find our test cases.

In boundary value analysis we choose bounded values, and a value after lower boundary value and a value before the peak value and one value we can take from anywhere within the limits.

|  |  |  |
| --- | --- | --- |
| **Test cases** | **Input** | **Expected output** |
| 1 | 0300-0011223  (>=0000 && <=9999) | Registered |
| 2 | Xxyyzz  ((>=0000 && <=9999)) | Login |
| 3 | (>0&&<10,000)  (>=0000 && <=9999) | Confirm withdrawal |
| 4 | (>0&&<20,000)  (>=0000 && <=9999) | Confirm transaction |

**Graph for assignment 2:**

**Values above than 50 are representing strong robustness.**

**Similarly below 2 values are also robust values.**

**Changes in Assignment 1:**

**Function 1 :-**

**Equivalence Classes :**

Valid classes = classes <1 to 25> <25,50><50,75><75,100>

invalid classes = [1,0,-1 , ...]

invalid classes = [101,102,103,104, ...]

**Function 2 :-**

**Equivalence Classes :**

**Account number**

Valid classes = <1 to 25> <25,50><50,75><75,100>

invalid classes = [1,0,-1 , ...]

invalid classes = [101,102,103,104, ...]

**Amount**

Valid classes = <1000,10000><10000,20000><20000,30000><30000,40000><40000,50000>

Invalid classes = <10000,9999,9998,..>

Invalid classes = <50001,50002,50003,...>

**Function 3 :-**

**Equivalence Classes :**

**Account number**

Valid classes = <1 to 25> <25,50><50,75><75,100>

invalid classes = [1,0,-1 , ...]

invalid classes = [101,102,103,104, ...]

**Amount**

Valid classes = <1000,10000><10000,20000><20000,30000><30000,40000><40000,50000>

Invalid classes = <10000,9999,9998,..>

Invalid classes = <50001,50002,50003,...>

**Name (Char Length)**

Valid classes = <6,10><10,14><10,18>

Invalid Classes = <6,5,4,3,...>

Invalid Classes = <18,19,20,21,22,...>

**GitHub link of assignment 3:**

https://github.com/BSE173153/SQA-Assignment3