



PES UNIVERSITY

**AUGUST – DECEMBER 2022 SEMESTER 5**

# **SOFTWARE ENGINEERING LAB TASKS**

**Professor-in-charge: Dr. Jayashree R**

Teaching Assistant: Aanchal Narendran (Sem VII)

We hope at this point in your semester you have achieved considerable progress in your Software Engineering Project. All of the tasks in this lab manual are geared towards improving and testing your project prior to your submissions.

## **Team Details:**

Moulya Rajesh Shetty - PES2UG20CS204

Shreya Nadella - PES2UG20CS208

Naman Choudhary - PES2UG20CS209

Nivedita Venkat - PES2UG20CS232

Praneeth Kumar L - PES2UG20CS251

## **Problem Statement – 1: Unit Testing**

A unit is the smallest block of code that functions individually. The first level of testing is Unit testing and this problem statement is geared towards the same.

- Discuss with your teammates and demarcate units in your code base
  - Note: discuss why the code snippet you have chosen can be classified as a unit

- Develop test cases for both valid and invalid data
- Ideate how you could further modularize larger blocks of code into compact units with your teammates

### Solution:

In our project, the smallest part of the code is the login page

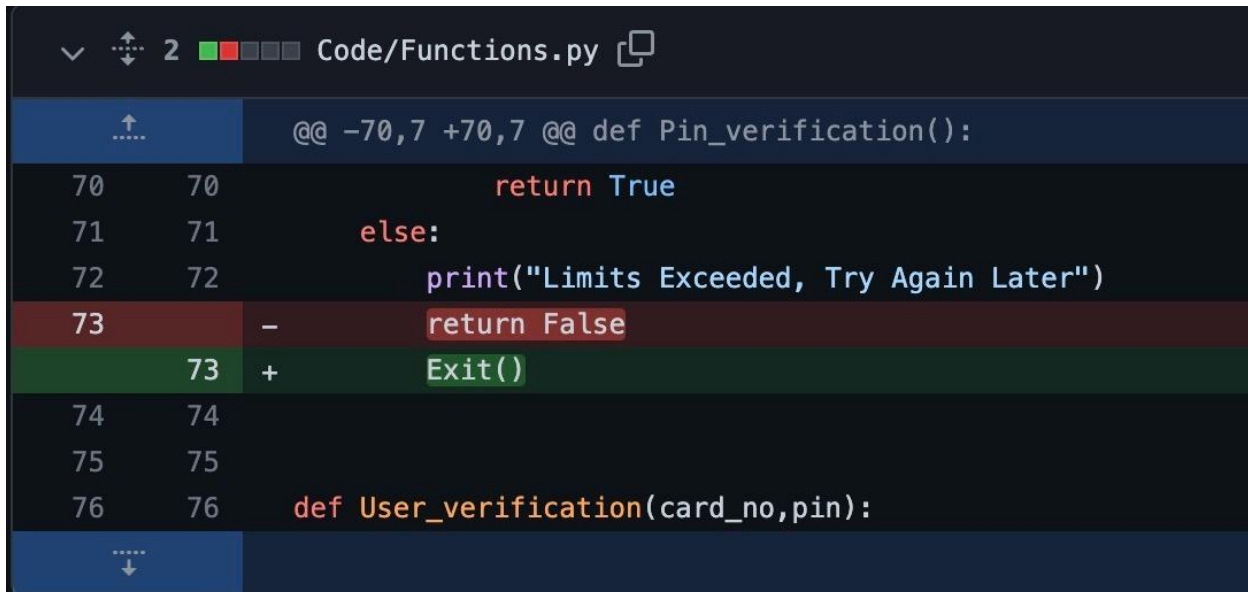
Two levels of access given:

- Customer level access - to perform the functional activities provided by the atm
- Administrator-level access - to help maintain and run the machine

The passcode is in masked form when typed in the passcode field.

The user ID and passcode are checked against our database.

If incorrect details are entered by the user, an error is displayed.



The screenshot shows a code editor window titled 'Code/Functions.py'. The code is as follows:

```

@@ -70,7 +70,7 @@ def Pin_verification():
70     70         return True
71     71     else:
72     72         print("Limits Exceeded, Try Again Later")
73     -         return False
73     +         Exit()
74     74
75     75
76     76     def User_verification(card_no,pin):
  
```

```

Code/Functions.py
@@ -39,6 +39,7 @@ def Card_search(card_no):
    39      39          if card_no == card_nos[i]:
    40      40              user_index = i
    41      41              return True
    42      42          print("Card Number not found")
    43      43          return False
    44      44
    45      45      def Card_no_verification():
@@ -150,8 +151,9 @@ def Check_Balance():
    150     151
    151     152      def Pin_change():
    152     153          global pins
    154     154          pin=pins[user_index]
    153     155          old_pin = int(input("Enter your old pin: "))
    154     156          if old_pin == pins:
    155     157          new_pin = int(input("Enter your new pin: "))
    156     158          if len(str(new_pin)) != 4:
    157     159              print("Pin should be 4 digits")

```

Serial No:	Test Cases	Valid/Invalid	Expected Output	Actual Output
1	Valid User ID and Passcode	Valid	Login Successful	Login Successful
2	Valid User ID and Invalid Passcode	Invalid	Login Unsuccessful	Login Unsuccessful (Re-enter Password)
3	Invalid User ID and	Invalid	Login Unsuccessful	Login Unsuccessful

	Valid Passcode			(Sign up page prompted)
4	Invalid User ID and Invalid Passcode	Invalid	Login Unsuccessful	Login Unsuccessful (Sign up page prompted)
5	Passcode entered and User ID missing	Invalid	Login Unsuccessful	Login Unsuccessful (Error: enter Username)
6	User ID entered and Passcode missing	Invalid	Login Unsuccessful	Login Unsuccessful (Error: enter Password)

**Ideate how you could further modularize larger blocks of code into compact units with your teammates**

- 1) Developing team breaks down the complete software into various modules.
- 2) Effective modular design can be achieved if the partitioned modules are separately solvable, modifiable as well as compilable.
- 3) In order to build a software with effective modular design there is a factor "Functional Independence" which comes into play.
- 4) The meaning of Functional Independence is that a function is atomic in

nature so that it performs only a single task of the software without or with

least interaction with other modules.

5) Functional Independence is considered as a sign of growth in modularity

i.e., presence of larger functional independence results in a software system

of good design and design further affects the quality of the software.

### **Dynamic Testing**

Dynamic testing involves execution of your code to analyse errors found during execution. Some common techniques are Boundary Value Analysis and Mutation Testing.

#### **Problem Statement – 2.a: Boundary Value Analysis**

When it comes to finding errors in your code base, they are often found at locations where a condition is being tested. Due to this, developers often use Boundary Value tests to reduce defect density.

- How would you define a boundary test?
  - Note: Simple relational conditions are a basic example
- Build your boundary test cases and execute them

#### **Problem Statement – 2.b: Mutation Testing**

- Using your isolated units from the first problem statement, ideate with your team mates on how to mutate the code

- Develop at least 3 mutants of the functioning code and test all 4 code bases using the test case from the first problem statement

```

Code/Functions.py
@@ -45,8 +45,8 @@ def Card_search(card_no):
45 45 def Card_verification():
46 46     n = 0
47 47     while(n<3):
48 -     card_no = int(input("Enter your card number: "))
49 -     if not (card_no>999999999 and card_no<=9999999999): #Make it !=
48 +     card_no=int(input("Enter your card number: "))
49 +     if len(card_no)==9:
50         print("Card No should be 9 digits, Try again\n")
51         n+=1
52     elif Card_search(card_no):
@@ -62,7 +62,7 @@ def Pin_verification():
62 62     n = 0
63 63     while(n<3):
64 64         pin = input("Enter your pin: ")
65 -         if len(pin) != 4:
65 +         if pin>999 and pin<=9999:
66             print("Pin should be 4 digits, Try again")
67             n+=1
68         elif int(pin) == pins[user_index]:

```

```

Code/Functions.py
@@ -86,8 +86,8 @@ def User_verification(card_no,pin):
86 86 def Account_type():
87 87     global acc_choice
88 88     print("Select your Account Type: ")
89 -     for i in range(len(Account_Types)):
90 -         print(i+1,Account_Types[i])
89 +     print("1. Savings")
90 +     print("2. Current")
91     acc_choice = int(input("Enter your choice: "))
92     if acc_choice == 1:
93         print("You have selected Savings Account")

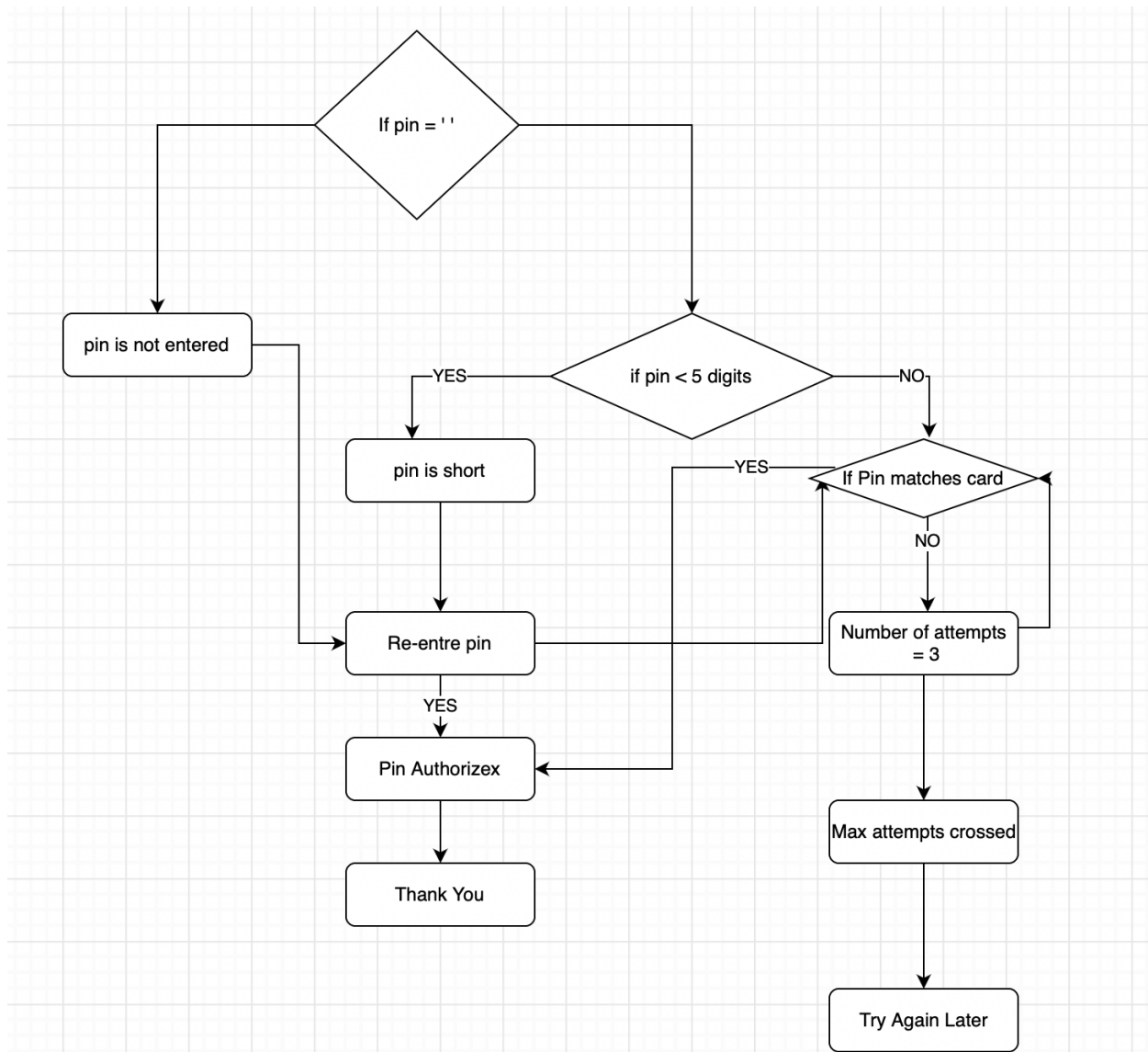
```

### **Problem Statement – 3: Static Testing**

Static testing involves validating your code without any execution. Under this problem statement, you will be expected to analyse and calculate the cyclomatic complexity of your code.

- Using the unit you selected in the first problem statement as an example, develop the control flow graph of your problem statement.

- Using the Control flow graph, calculate the cyclomatic complexity of your code.
- Using the cyclomatic complexity as an indicator, Ideate and code your unit again to reduce complexity



From the above graph,  
Number of Edges = 12

Number of Nodes = 11

$P = 2$

$M = E - N + 2P$

$M = 12 - 11 + 4$

$M = 5$

#### **Problem Statement – 4: Acceptance Testing**

Assume your neighbouring team is the client for your code. Give them an idea of what your product is and the software requirements for the product.

- Exchange your code base and test each others projects to see if it meets user requirements

The user requirements are met. These contain the login page where the user can sign in using his/her login credentials and enter the interface of his/her bank. It contains the options to withdraw cash from the ATM from the user's bank account. The user also is able to check his bank balance.

- If you identify a bug in the project you are testing, inform the opposing team of the bug

There was a small bug in the project, wherein , the user was still able to withdraw the cash from his bank account even after his/her balance was at 0, making his or her balance negative. There were no other major bugs found in the project while testing.

- As a team, based in clients experience, ideate modifications to the existing project that could improve client experience



A modification proposed by the team was to incorporate a feature wherein, the user is automatically logged out after 2 minutes of inactivity ,in order to improve privacy and security.

### **Problem Statement – 5: Maintenance Activities**

Once a product is completed, it is handed off to a service based company to ensure all maintenance activities are performed without the added expenditure of skilled developers. However, a few tasks are performed by the maintenance team to gauge the product better. In this problem statement, you will be asked to experiment with your code.

- Exchange code bases with your neighboring teams and reverse engineer a block of code in order to understand it's functionality
- After understanding the code block, Re-Engineer the code
- Ideate how to refactor the code and the portion of the code base you would have to change
- Discuss how the new changes would impact the time and space complexity of the project during execution
- After Reverse Engineering and Re-Engineering the code, perform acceptance testing between the teams

The team came up with an idea, that we should restructure the code in such a way that instead of creating a different set of variables for each user, it can be done dynamically, to save space.

The new changes helps in reduction of the space complexity by a huge factor, while the time complexity of the program remains the same.

This change was possible only because of reverse engineering the code and comparing it back to our SRS.