# **Team Details:**

**NANDANA MANOJ PES2UG20CS215**

**NEERAJ SUDHEER PES2UG20CS221**

**PRATHAM MANJA PES2UG20CS253**

**PREETHIKA AJAY KUMAR PES2UG20CS256**

## 

## **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.

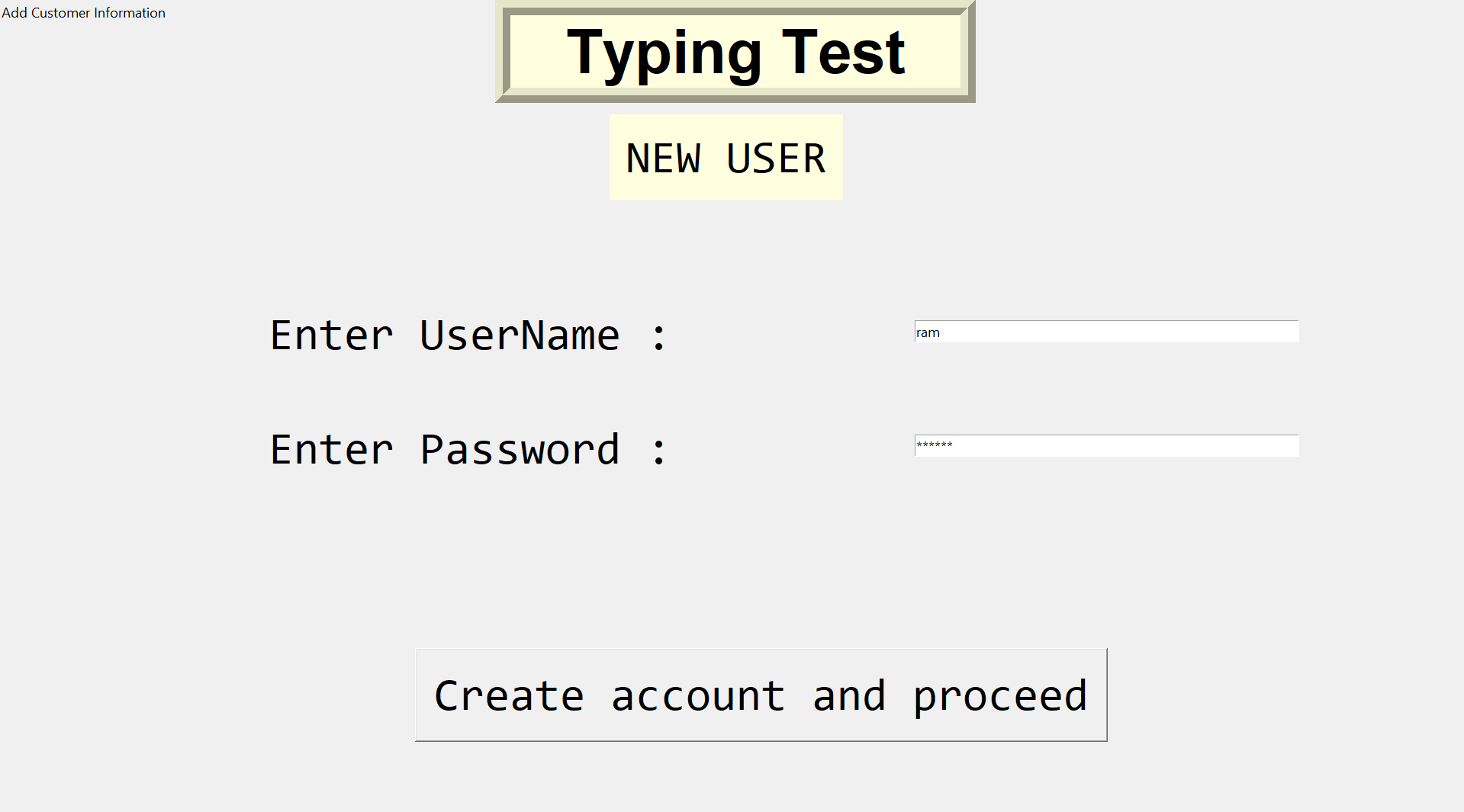
· Discuss with your teammates and demarcate units in your code base

o Note: discuss why the code snippet you have chosen can be classified as a unit

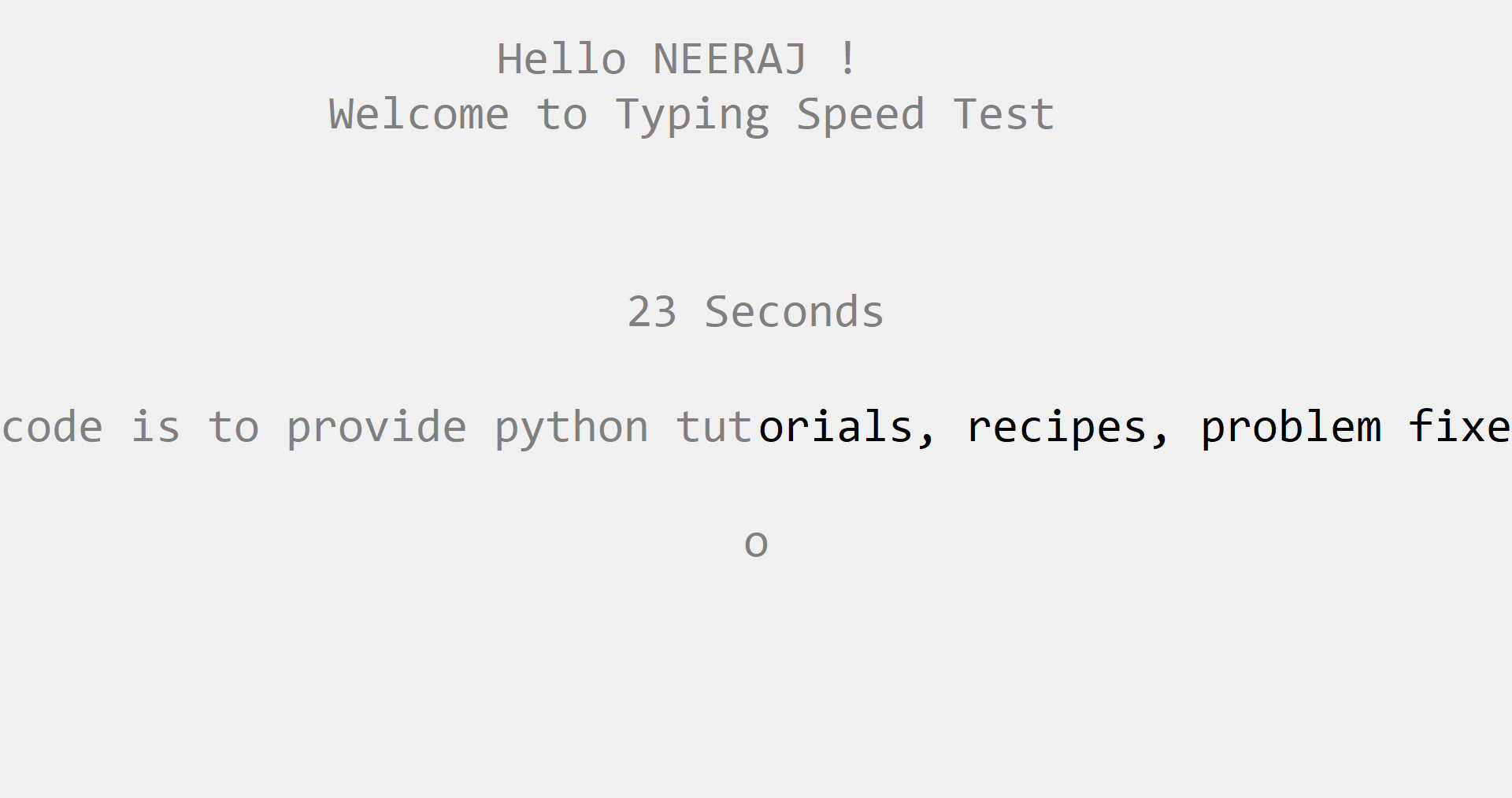
· Develop test cases for both valid and invalid data

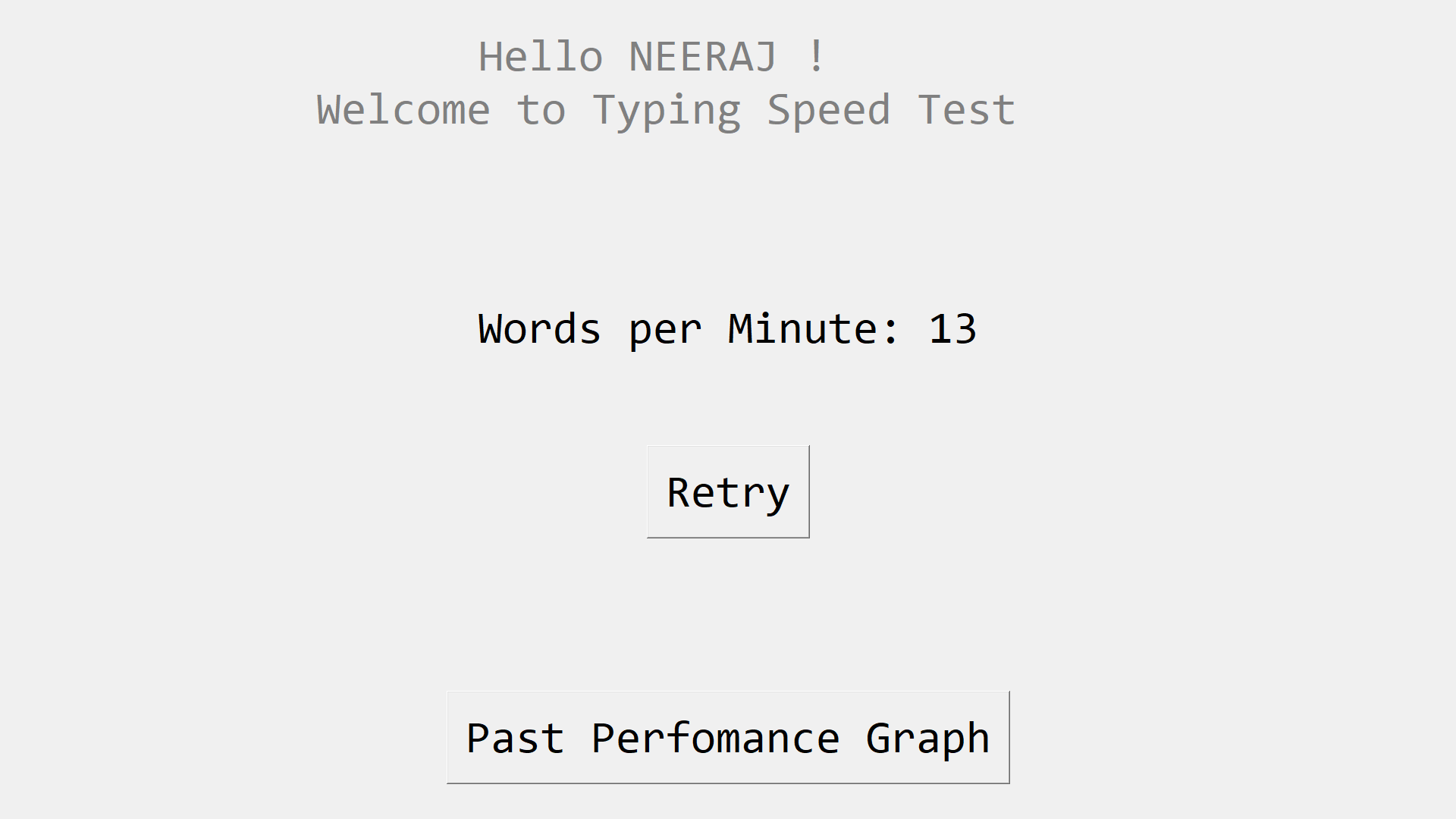
· Ideate how you could further modularize larger blocks of code into compact units with your teammates

| **Login Page Test Cases** | **Status** |
| --- | --- |
| Verify if a user will be able to login with a valid username and valid password. | Passed |
| Verify if a user cannot login with a valid username and an invalid password. | Passed |
| Verify the login page for both, when the field is blank and the Submit button is clicked. | Passed |
| Verify the messages for invalid login. | Passed |
| Verify if the data in the password field is either visible as asterisk or bullet signs. | Passed |
|  | |
| **New User Creation Test Cases** | **Status** |
| Verify if the user cannot proceed without filling all the mandatory fields. | Passed |
| Verify if a user can sign-up successfully with all the mandatory details. | Passed |
| Verify if user can create an account having a username less than 6 characters long | Passed |
| Verify if user can create an account having a username greater than than 13 characters long | Passed |
| Verify if a user can log in with the valid details. | Passed |
| Verify if data entered by the user is registered in the database | Passed |
| Verify if the format of the data entered by the user is registered in the database is correct | Passed |

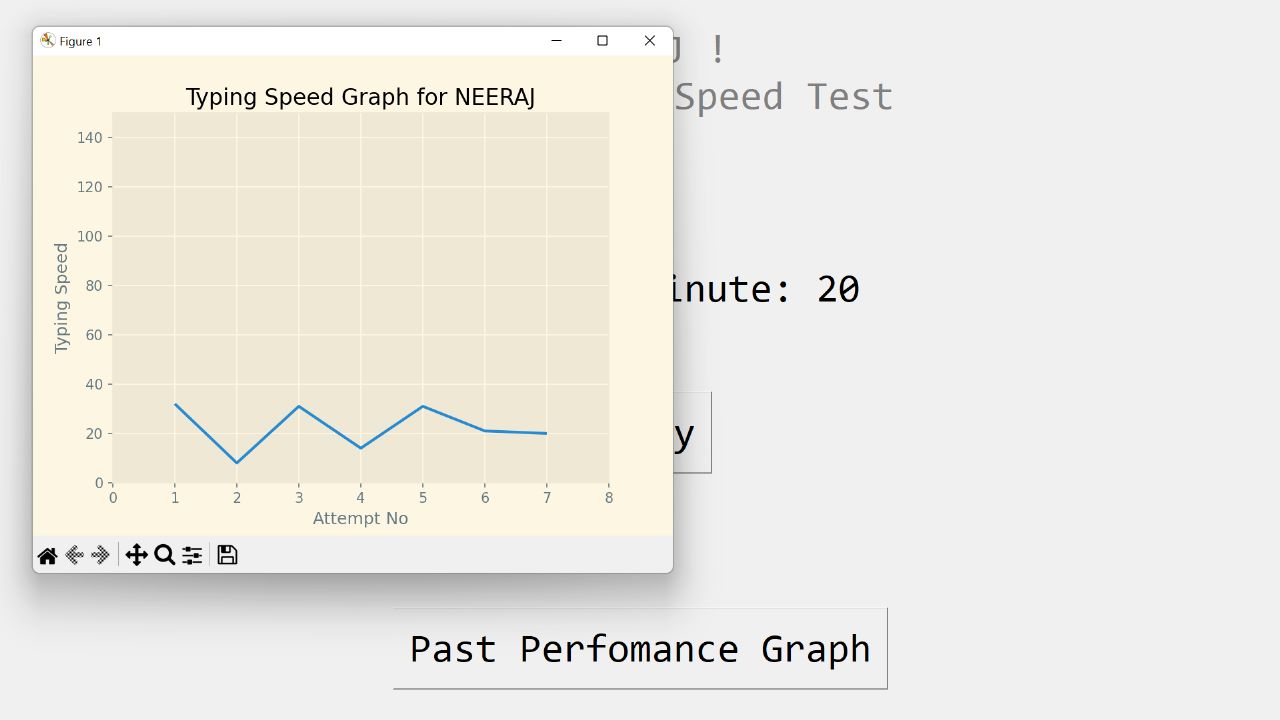


| **TypingTest Test Cases** | **Status** |
| --- | --- |
| Verify if the user’s name is displayed after login in the typing test page | Passed |
| Verify if a input provided by the keyboard is taken by the page | Passed |
| Verify if word entered is only registered if its correct | Passed |
| Verify that the time runs for exactly 60 seconds | Passed |
| Verify if the typing speed result is accurate and matching with the manually calculated score | Passed |





| **Result and Graph Test Cases** | **Status** |
| --- | --- |
| Verify if the user’s name is displayed on the graph generated after completing the test is matching with the logged in user | Passed |
| Verify if the data retrieved for generating the graph is for the same user. | Passed |
| Verify if the score generated is inserted into the database after the test is completed | Passed |
| Verify if the retry button works and all the functionality work the same in case of its usage | Passed |



## **Problem Statement – 2: 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.

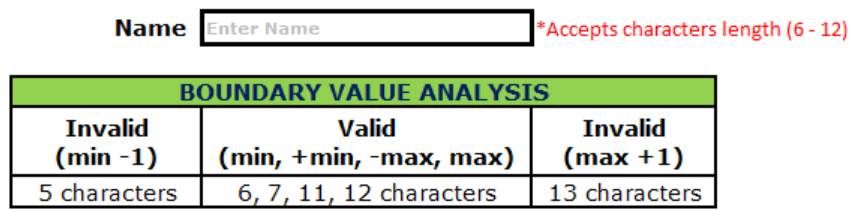
## **Problem Statement – 2.a: Boundary Value Analysis**

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?

o 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

**New User Creation [Mutant with >13 instead of <13 for checking the number of characters entered for username]**

| **New User Creation Test Cases** | **Status** |
| --- | --- |
| Verify if the user cannot proceed without filling all the mandatory fields. | Passed |
| Verify if a user can sign-up successfully with all the mandatory details. | Passed |
| Verify if user can create an account having a username less than 6 characters long | Passed |
| Verify if user can create an account having a username greater than than 13 characters long | Failed |
| Verify if a user can log in with the valid details. | Passed |
| Verify if data entered by the user is registered in the database | Passed |
| Verify if the format of the data entered by the user is registered in the database is correct | Passed |

**New User Creation [Mutant with <6 instead of >6 for checking the number of characters entered for username]**

| **New User Creation Test Cases** | **Status** |
| --- | --- |
| Verify if the user cannot proceed without filling all the mandatory fields. | Passed |
| Verify if a user can sign-up successfully with all the mandatory details. | Passed |
| Verify if user can create an account having a username less than 6 characters long | Failed |
| Verify if user can create an account having a username greater than than 13 characters long | Passed |
| Verify if a user can log in with the valid details. | Passed |
| Verify if data entered by the user is registered in the database | Passed |
| Verify if the format of the data entered by the user is registered in the database is correct | Passed |

**Database connection attributes values mutated**

| **Login Page Test Cases** | **Status** |
| --- | --- |
| Verify if a user will be able to login with a valid username and valid password. | Failed |
| Verify if a user cannot login with a valid username and an invalid password. | Failed |
| Verify the login page for both, when the field is blank and the Submit button is clicked. | Passed |
| Verify the messages for invalid login. | Failed |
| Verify if the data in the password field is either visible as asterisk or bullet signs. | Passed |
| **New User Creation Test Cases** | **Status** |
| Verify if the user cannot proceed without filling all the mandatory fields. | Failed |
| Verify if a user can sign-up successfully with all the mandatory details. | Failed |
| Verify if user can create an account having a username less than 6 characters long | Failed |
| Verify if user can create an account having a username greater than than 13 characters long | Failed |
| Verify if a user can log in with the valid details. | Failed |
| Verify if data entered by the user is registered in the database | Failed |
| Verify if the format of the data entered by the user is registered in the database is correct | Failed |

| **TypingTest Test Cases** | **Status** |
| --- | --- |
| Verify if the user’s name is displayed after login in the typing test page | Failed |
| Verify if a input provided by the keyboard is taken by the page | Passed |
| Verify if word entered is only registered if its correct | Passed |
| Verify that the time runs for exactly 60 seconds | Passed |
| Verify if the typing speed result is accurate and matching with the manually calculated score | Passed |

| **Result and Graph Test Cases** | **Status** |
| --- | --- |
| Verify if the user’s name is displayed on the graph generated after completing the test is matching with the logged in user | Failed |
| Verify if the data retrieved for generating the graph is for the same user. | Failed |
| Verify if the score generated is inserted into the database after the test is completed | Failed |
| Verify if the retry button works and all the functionality work the same in case of its usage | Failed |

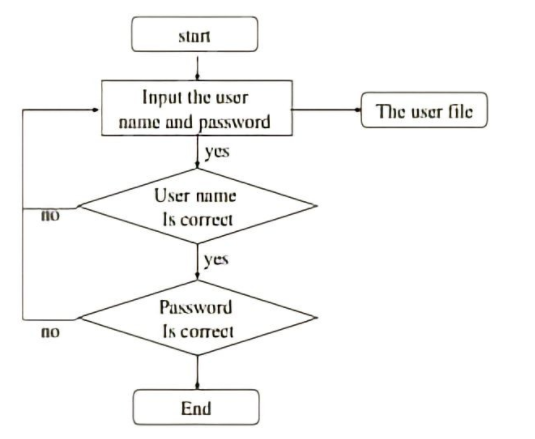
## **Problem Statement – 3: Static Testing**

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



**Cyclomatic Complexity**

M=E-N+2P

where,

E = the number of edges in the control flow graph

N = the number of nodes in the control flow graph

P = the number of connected components

E = 7 N = 6 P = 1

M = 7-6+2x1

M = 3

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

Assume your neighboring 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

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

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

* The software product was according to the given functional requirements. It was responsive, interactive and gave an overall smooth GUI experience.

* Although the project performed well and was working as it should, it could be improved by some extra functional features like testing typing speed with 30s,15s time interval or calculating by giving sentences of predefined length to further improve user experience.

## **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

o Ideate how to refactor the code and the portion of the code base you would have to change

o 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

Reverse engineering can deconstruct the software to extract their

design information.

● It allows us to identify how the developer design a particular part

of code so that we can recreate and create a replacement part

of the product the code the engineers copy or mimic the a

design without the original blueprint

● The process of restructuring existing code changes the factoring

without changing its external behavior.

● Change the code without breaking current functionalities

● For changing the code base it involves some steps that is

inventory analysis, document Reconstruction, reverse

engineering, code reconstruction , data reconstruction and

forward engineering

● In our neighboring team’s code almost everything fine except a

few changes, these are the functions to be declared as small as

possible and consolidate the code into functions

● Also, the team can increase the readability of the code

to increase the maintainability and reusability.

● After the indicated changes are done the code’s space

complexity reduces since we are consolidating the code also the

time complexity reduces because we are reducing the number of

variables therefore the time taken for execution considerably

reduces.