PES UNIVERSITY

AUGUST – DECEMBER 2022 SEMESTER 5

SOFTWARE ENGINEERING LAB TASKS

Professor-in-charge: Dr. Jayashree R

NAME	SRN
DIVIJA L	PES1UG20CS134
G DHATHRI	PES1UG20CS141
GAURAV DNYANESH MAHAJAN	PES1UG20CS150
JALLURI HARSHITHA DEVI	PES1UG20CS173

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.

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

  Considering our members' discussions, we made the decision to designate one of the app's units as the login page, which is the primary utility that users are expected to utilise.
  The login feature is the first utility that can be evaluated to determine whether an application is functioning out of all the other features it offers.

- Develop test cases for both valid and invalid data
  Some of the test cases that we could collaboratively come with were:
  FOR VALID DATA:
  - ➔ Verify if a user will be able to login with a valid username and valid password.
  - ➔ Verify if the data in password field is either visible as asterisk or bullet signs.
  - ➔ Verify if the font, text colour, and colour coding of the Login page is as per the standard.

  FOR INVALID DATA:
  - ➔ Verify if a user cannot login with a valid username and an invalid password.
  - ➔ Verify the login page for both, when the field is blank and Submit button is clicked.
  - ➔ Verify the login page for both, when the field is blank and Submit button is clicked.
- Ideate how you could further modularize larger blocks of code into compact units with your teammates
  - ➔ Upon integration of the login unit with the home page, the entire home screen and other utilities can be tested.

<u>Problem Statement – 2: Dynamic Testing</u>

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

<u>Problem Statement – 2.a: Boundary Value Analysis</u>

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

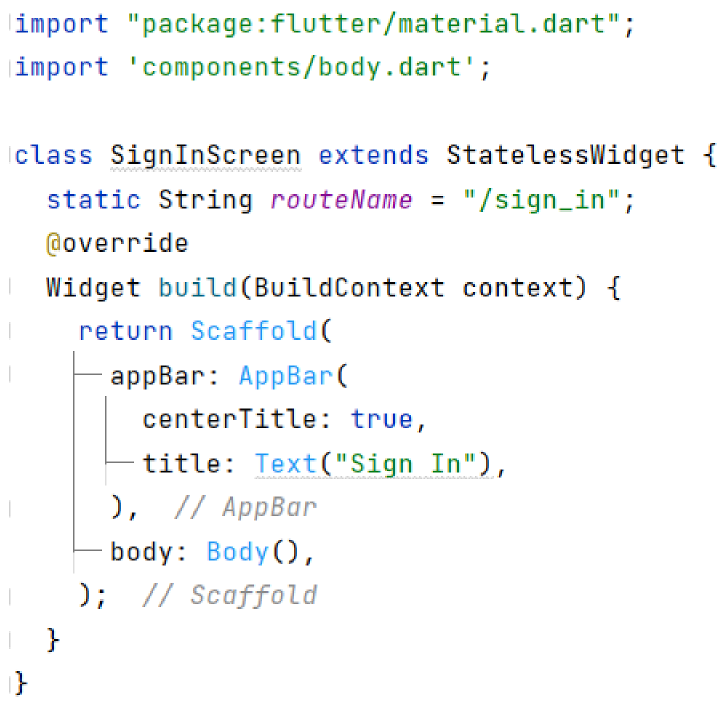
Problem Statement – 2.b: Mutation Testing

- 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

Problem Statement – 3 Static Testing

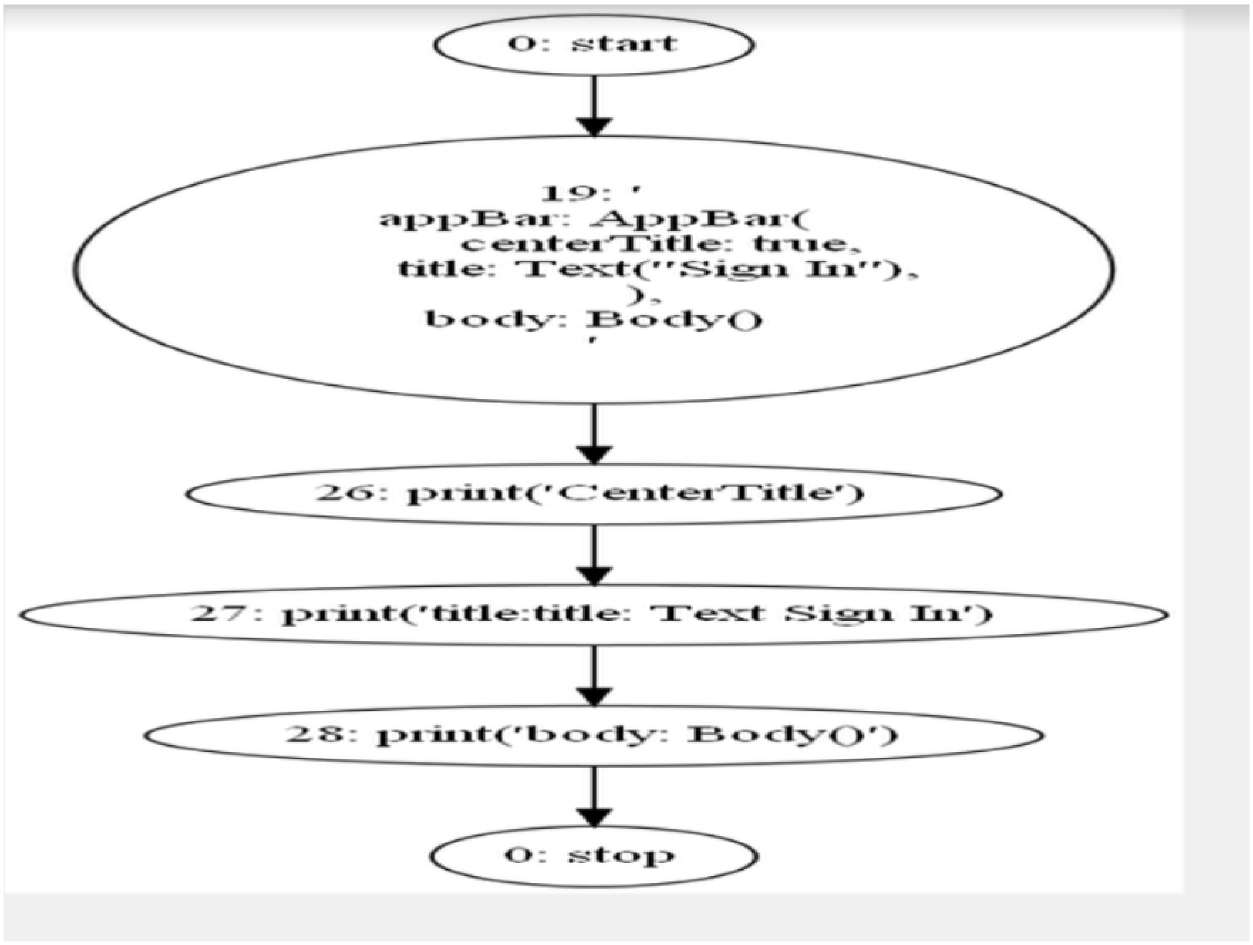
Static testing involves validating your code without any execution. Under this problem statement, you will be expected to analyse and calculate the cyclomatic complexity of your code.

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



```
import "package:flutter/material.dart";
import 'components/body.dart';

class SignInScreen extends StatelessWidget {
  static String routeName = "/sign_in";
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        centerTitle: true,
        title: Text("Sign In"),
      ),  // AppBar
      body: Body(),
    );  // Scaffold
  }
}
```



- Using the Control flow graph, calculate the cyclomatic complexity of your code.

  The number of nodes is 6

  The number of edges is 5

  The connected component with the exit node is 1

  Using the formula : M - N + 2P

  The cyclomatic complexity will be 3.

- Using the cyclomatic complexity as an indicator, Ideate and code your unit again to reduce complexity

  Since we have already used a reduced number of components in this unit , cyclomatic complexity of 3 seems to be good.

<u>**Problem Statement – 4 Acceptance Testing**</u>

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 other's projects to see if it meets user requirements

  We exchanged our code base with Diya's [PES1UG20CS137] team who are developing a timetable generator using python.
- If you identify a bug in the project you are testing, inform the opposing team of the bug

  We identified a minor bug regarding classroom size. The team did not recognise the classroom size while allotting the timetable. We suggested them to consider classroom size so that the timetable allotted to users did not overcompensate the capacity.
- As a team, based in clients experience, ideate modifications to the existing project that could improve client experience

  <u>They could add a single LOC considering the size l</u>ike a len() function.