Department of Computer Engineering

Academic Term: First Term 2023-24 Class: T.E

/Computer Sem – V / Software Engineering

Practical No:	**9**
Title:	**Designing Test Cases for Performing White Box Testing**
Date of Performance:	**03/10/2023**
Roll No:	**9628**
Team Members:	**Mann Patel**

Rubrics for Evaluation:

Sr. No	Performance Indicator	Excellent	Good	Below Average	Total Score
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Theory Understanding(02)	02(Correct)	NA	01 (Tried)	
3	Content Quality (03)	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Questions (04)	04(done well)	3 (Partially Correct)	2(submitted)	

Signature of the Teacher:

Department of Computer Engineering

Academic Term: First Term 2022-23 Class: T.E

/Computer Sem – V / Software Engineering

Signature of the Teacher:

Lab Experiment 09

Experiment Name: Designing Test Cases for Performing White Box Testing in Software Engineering

Objective: The objective of this lab experiment is to introduce students to the concept of White Box Testing, a testing technique that examines the internal code and structure of a software system. Students will gain practical experience in designing test cases for White Box Testing to verify the correctness of the software's logic and ensure code coverage.

Introduction: White Box Testing, also known as Structural Testing or Code-Based Testing, involves assessing the internal workings of a software system. It aims to validate the correctness of the code,
identify logic errors, and achieve maximum code coverage.

Lab Experiment Overview:

1. Introduction to White Box Testing: The lab session begins with an introduction to White Box Testing, explaining its purpose, advantages, and the techniques used, such as statement coverage,
branch coverage, and path coverage.
2. Defining the Sample Project: Students are provided with a sample software project along with its source code and design documentation.
3. Identifying Test Scenarios: Students analyze the sample project and identify critical code segments, including functions, loops, and conditional statements. They determine the test scenarios
based on these code segments.
4. Statement Coverage: Students apply Statement Coverage to ensure that each statement in the code
is executed at least once. They design test cases to cover all the statements.
5. Branch Coverage: Students perform Branch Coverage to validate that every branch in the code, including both true and false branches in conditional statements, is executed at least once. They design test cases to cover all branches.
6. Path Coverage: Students aim for Path Coverage by ensuring that all possible execution paths through the code are tested. They design test cases to cover different paths, including loop iterations and condition combinations.
7. Test Case Documentation: Students document the designed test cases, including the test scenario,
input values, expected outputs, and any assumptions made.
8. Test Execution: In a test environment, students execute the designed test cases and record the results, analyzing the code coverage achieved.
9. Conclusion and Reflection: Students discuss the significance of White Box Testing in software quality assurance and reflect on their experience in designing test cases for White Box Testing.

Learning Outcomes: By the end of this lab experiment, students are expected to:

Understand the concept and importance of White Box Testing in software testing.
Gain practical experience in designing test cases for White Box Testing to achieve code coverage.

Learn to apply techniques such as Statement Coverage, Branch Coverage, and Path Coverage in test case design.
Develop documentation skills for recording and organizing test cases effectively.
Appreciate the role of White Box Testing in validating code logic and identifying errors.
Pre-Lab Preparations: Before the lab session, students should familiarize themselves with White Box Testing concepts, Statement Coverage, Branch Coverage, and Path Coverage techniques.
Materials and Resources:
Project brief and details for the sample software project
Whiteboard or projector for explaining White Box Testing techniques
Test case templates for documentation
Conclusion: The lab experiment on designing test cases for White Box Testing equips students with
essential skills in assessing the internal code of a software system. By applying various White Box Testing techniques, students ensure comprehensive code coverage and identify logic errors in the software. The experience in designing and executing test cases enhances their ability to validate code
behavior and ensure code quality. The lab experiment encourages students to incorporate White Box
Testing into their software testing strategies, promoting robust and high-quality software development. Emphasizing test case design in White Box Testing empowers students to contribute to software quality assurance and deliver reliable and efficient software solutions.

White-box testing is a method of software testing that examines the internal structures or workings of an application. When it comes to a gas detection system using Arduino Uno, white-box testing can be performed to ensure that the code is functioning as expected. Let's consider a simplified example of a gas detection system that uses Arduino Uno. The system is designed to detect the presence of flammable gases and activate an alarm when such gases are detected. Here's an example of how you can conduct white-box testing for the system:

Code:

```
const int gasSensorPin = A0; // Analog pin for gas sensor
const int alarmPin = 13; // Digital pin for the alarm

void setup() {
  Serial.begin(9600);
  pinMode(alarmPin, OUTPUT);
}

void loop() {
  int sensorValue = analogRead(gasSensorPin);
  Serial.print("Gas Sensor Value: ");
  Serial.println(sensorValue);

  if (sensorValue > 500) { // Threshold value for gas detection
    activateAlarm();
  } else {
    deactivateAlarm();
  }

  delay(1000); // Delay for stability
}

void activateAlarm() {
  digitalWrite(alarmPin, HIGH);
  Serial.println("Gas Detected! Activating Alarm.");
}

void deactivateAlarm() {
```

```
  digitalWrite(alarmPin, LOW);
}
```

White-Box Testing Steps:

Code Review: Review the source code to understand its structure, including variables, functions, and conditional statements.

Static Analysis: Check for any potential issues in the code without actually running it. Ensure there are no syntax errors, variable conflicts, or other coding issues.

Unit Testing:

Test the activateAlarm and deactivateAlarm functions to ensure that they are functioning properly.
Test the loop function with different input values for sensorValue, including values greater than 500 and values less than 500, to confirm that the system responds accordingly.
Path Testing: Ensure that all possible paths through the code are tested, including the if-else conditions.

Boundary Testing: Test the system with values on the threshold (500 in this case) to verify if the alarm is triggered appropriately.

Integration Testing: Check if the hardware and software components are working together as expected. Simulate the gas sensor input and verify that the alarm is activated and deactivated correctly.

Performance Testing: Run the code for an extended period to check for any memory leaks or performance issues.

Error Handling Testing: Intentionally introduce errors or unexpected input to check if the system can handle such situations gracefully.

Gas Sensor
|
V
A0 Pin of Arduino Uno

Digital Pin 13 of Arduino Uno
|
V

Buzzer/Alarm

GND Pin of Gas Sensor and Buzzer connected to GND Pin of Arduino Uno

Test Case ID	Test Description	Test Steps	Expected Result	Actual Result	Pass/ Fail
1	Test activateAlarm Function	Set input > threshold Call activateAlarm function Check if alarm is activated	Alarm Activated	Alarm Activated	Pass
2	Test deactivateAlarm Function	Set input < threshold Call deactivateAlarm function Check if alarm is deactivated	Alarm Deactivated	Alarm Deactivated	Pass
3	Test Sensor Integration with Arduino	Simulate gas presence by providing input signal Monitor system response	System detects the Presence of the gas	System detects the Presence of the gas	Pass