Department of Computer Engineering

Academic Term: First Term 2023-24

Class: T.E /Computer Sem – V / Software Engineering

Practical No:	9
Title:	**Design test cases for performing white box testing**
Date of Performance:	
Roll No:	9567, 9552
Team Members:	Shruti Patil, Mrunal Kotambkar

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:

EXPERIMENT NO:6
WHITE BOX TESTING

* Perform basis path Testing on the Program of printing grades of a student based on 5 subject marks.

Algorithm:-

Step 1: Input sub1, sub2, sub3, sub4, sub5 // marks of 5 subjects out of 100

Step 2: avg=(sub1 + sub2 + sub3 + sub4 + sub5) / 5

Step 3: If(avg>=90) Then

Step 4: Output = ”Grade: A”

Step 5: ElseIf(avg>=80 and avg<90) Then

Step 6: Output=”Grade: B”

Step 7: ElseIf(avg>=70 and avg<80) Then

Step 8: Output=”Grade: C”

Step 9: ElseIf(avg>=60 and avg<70) Then

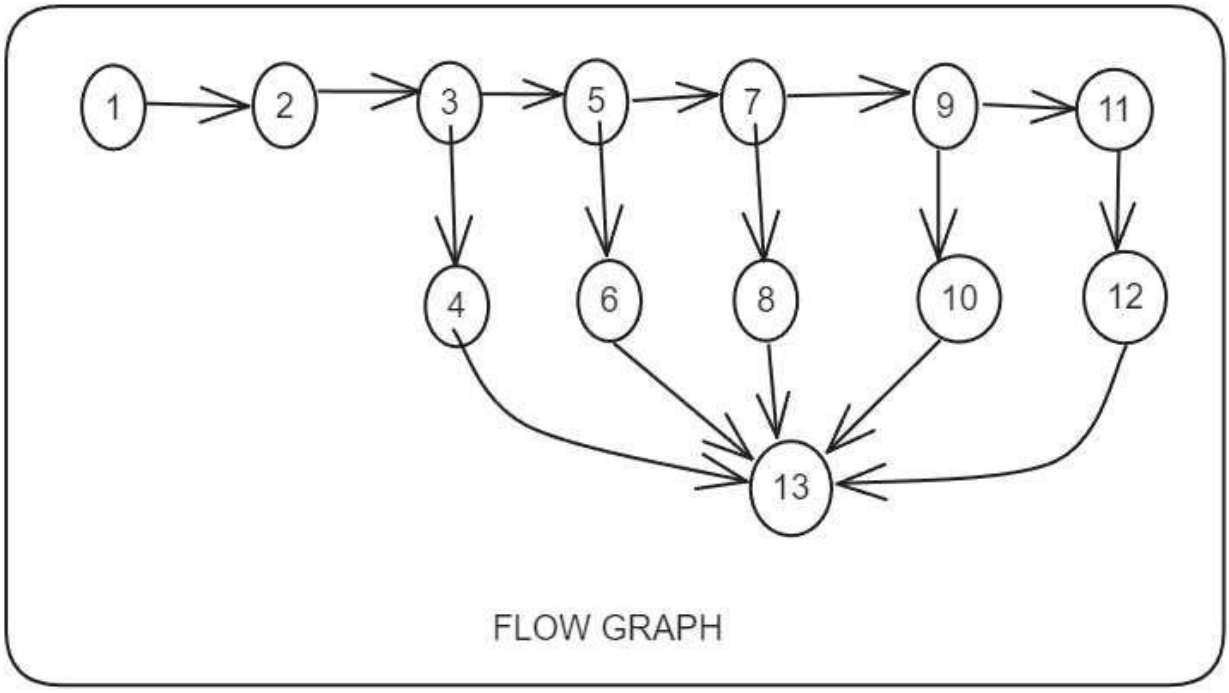
Step 10: Output=”Grade: D”

Step 11: Else

Step 12: Output=”Grade: F”

Step 13: Return Output

Step 1: Construction of Control Flow Graph



Step 2: Find Independent Program Paths

Path 1: 1-2-3-4-13

Path 2: 1-2-3-5-6-13

Path 3: 1-2-3-5-7-8-13

Path 4: 1-2-3-5-7-9-10-13

Path 5: 1-2-3-5-7-9-11-12-13

Step 3: Calculation of Cyclomatic Complexity

CC = E - N + 2
CC = 16 - 13 +2
CC = 5

Step 4: Design Test cases For Each Independent Path

Test Case 1: sub1=90, sub2=92, sub3=93, sub4=94 sub5=89
Path 1: 1-2-3-4-13

Test Case 2: sub1=80, sub2=92, sub3=91, sub4=84 sub5=86
Path 2: 1-2-3-5-6-13

Test Case 3: sub1=77, sub2=78, sub3=82, sub4=81 sub5=77
Path 3: 1-2-3-5-7-8-13

Test Case 4: sub1=70, sub2=69, sub3=70, sub4=69 sub5=65
Path 4: 1-2-3-5-7-9-10-13

Test Case 5: sub1=60, sub2=56, sub3=59, sub4=66 sub5=52
Path 5: 1-2-3-5-7-9-11-12-13

POSTLAB:

Compare and contrast white box testing with black box testing, highlighting their respective strengths and weaknesses in different testing scenarios.

White box testing and black box testing are two distinct software testing approaches, each with its own strengths and weaknesses. They are used in different scenarios and have different objectives. Here's a comparison of the two testing techniques:

White Box Testing:

1. Objective: White box testing, also known as clear box testing or structural testing, focuses on testing the internal logic, code structure, and paths within the software application. The primary objective is to evaluate the internal working of the software.
2. Tester's Knowledge: Testers have access to the internal code, and they use this knowledge to design test cases based on code paths and logic.
3. Strengths:
    - Effective at uncovering low-level defects, including logical errors, code vulnerabilities, and performance bottlenecks.
    - Can identify security vulnerabilities and potential issues in code efficiency.
    - Well-suited for testing algorithms and complex data structures.
4. Weaknesses:
    - Limited in assessing user experience and high-level functionality.
    - Test cases are often based on assumptions and the tester's understanding of the code, which may lead to biased testing.
    - Requires access to the source code, which may not be possible for third-party or proprietary software.

Black Box Testing:

1. Objective: Black box testing, also known as behavioral testing, focuses on testing the software's functionality without knowledge of its internal code. The primary objective is to validate whether the software meets user requirements and behaves as expected.
2. Tester's Knowledge: Testers do not have access to the source code or knowledge of the internal code structure. They rely on the software's specifications, user documentation, and input-output behavior.
3. Strengths:
    - Effectively assesses the software from an end-user's perspective, ensuring it meets user requirements and functionality.
    - Suitable for testing usability, compatibility, and integration with external systems.
    - Easily applicable to third-party software or when the source code is unavailable.
4. Weaknesses:
    - Limited in identifying low-level defects, such as code vulnerabilities or performance issues.
    - May not uncover complex logic errors hidden within the code.
    - Test cases might miss certain code paths or edge cases that are not evident from the external behavior.

Scenarios for White Box Testing:

- Critical Systems: White box testing is suitable for critical systems where low-level defects can have severe consequences, such as aerospace, medical devices, and financial applications.
- Security-Critical Applications: It is essential for applications handling sensitive data or requiring robust security, as white box testing can uncover vulnerabilities.
- Complex Algorithms: When the software relies on complex algorithms, data structures, or mathematical calculations, white box testing can validate their correctness.

Scenarios for Black Box Testing:

- Functional Validation: For most software applications, especially those aimed at end-users, black box testing is essential to validate the functionality and user experience.
- Third-Party Software: When testing third-party software or software components, black box testing is the only viable option, as you typically do not have access to the source code.
- User-Centric Applications: Applications where the primary concern is user satisfaction and adherence to requirements, such as e-commerce platforms, mobile apps, and websites.

In practice, a combination of both white box and black box testing is often used to achieve comprehensive software testing. The choice between the two depends on the specific testing objectives, the nature of the software, and the available resources.

Analyze the impact of white box testing on software quality, identifying its potential to uncover complex logic errors and security vulnerabilities.

White box testing plays a significant role in enhancing software quality by uncovering complex logic errors and security vulnerabilities. Its impact on software quality is substantial, as it helps ensure that software is not only functional but also robust, secure, and reliable. Here's an analysis of the impact of white box testing on software quality:

1. Uncovering Complex Logic Errors:
    - Depth of Testing: White box testing enables testers to examine the internal logic, code structure, and code paths of the software. This allows for a more in-depth evaluation of the application's behavior, making it effective in uncovering complex logic errors that might be hidden within the code.
    - Path Coverage: Testers can design test cases that specifically target various code paths, including both common and uncommon scenarios. This increases the likelihood of identifying hidden logic errors that might not be apparent in traditional black box testing.
    - Conditional Statements and Loops: White box testing can evaluate the behavior of conditional statements and loops within the code, helping to detect issues related to branching logic and iterations.

- Integration Testing: White box testing can also be used in integration testing to ensure that different components of the software interact correctly and that complex data flow and control logic work as intended.

2. Uncovering Security Vulnerabilities:
    - Access to Source Code: Testers, in white box testing, have access to the source code, making it a powerful tool for identifying security vulnerabilities. They can scrutinize the code for potential weaknesses and threats.
    - Code Analysis: White box testing includes static code analysis and dynamic analysis to identify vulnerabilities such as buffer overflows, injection attacks, improper authentication, and data leakage.
    - Authentication and Authorization Testing: Testers can assess how authentication and authorization mechanisms are implemented, ensuring that only authorized users have access to specific functionality and data.
    - Secure Coding Practices: White box testing encourages the application of secure coding practices. It helps ensure that the code adheres to security standards and best practices, reducing the likelihood of vulnerabilities being introduced.
3. Software Quality Assurance:
    - Reliability and Robustness: By uncovering complex logic errors, white box testing contributes to the software's reliability and robustness. It helps ensure that the software behaves predictably under various conditions.
    - Quality Assurance: White box testing is a critical component of quality assurance. It verifies that the software functions as intended not only from a user perspective but also at the code level.
    - Comprehensive Testing: It complements black box testing by providing a comprehensive view of the software's quality. This combination of testing approaches results in a more thorough evaluation of the software.