What is White Box Testing?

White-box testing, also known as structural testing or code-based testing, concentrates on examining the internal structure and logic of the software application.

White Box Testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.

Characteristics of White Box Testing

- **Knowledge of the code**: The tester needs a deep understanding of the code and the system's internal workings.
- **Focus on code structure**: Tests are written to cover specific code paths, branches, conditions, loops, and functions.
- **Code coverage**: The aim is often to achieve a certain level of code coverage, such as statement coverage, branch coverage, or path coverage.
- **Early testing**: White box testing can be applied early in the development process, even before the user interface is complete.

Common Techniques in White Box Testing

- **Statement Coverage**: Ensuring that every line of code is executed at least once during testing.
- **Branch Coverage**: Testing all possible branches of decision points (e.g., if-else statements) in the code.
- **Path Coverage**: Testing all possible execution paths through the program.
- **Condition Coverage**: Ensuring that each condition in a decision statement evaluates to both true and false.
- **Loop Testing**: Validating the correct functioning of loops (e.g., for, while) with different iteration counts, including edge cases like zero or maximum iterations.

Example of White Box Test Case for Appointment Booking:

Component	Test Scenario	Test Case (Code-level)	Expected Outcome
Login Functionality	Code Path Testing	Test each possible code path for login (valid, invalid	Correct path executed: success on valid login, failure

Component	Test Scenario	Test Case (Code-level)	Expected Outcome
		credentials).	on invalid login.
	Condition Testing	Check if conditions for correct password and email validation.	Each condition branch is tested (true/false).
	Loop Testing	Test loop that counts login attempts (e.g., 3 invalid attempts).	The system should block after 3 invalid attempts.
Appointment Booking	Condition Testing	Test conditions for checking time slot availability before booking.	Only available slots should pass the booking condition.
	Code Path Testing	Test all code paths for booking (valid slot, invalid slot, holidays).	All paths executed successfully: bookings succeed/fail appropriately.
	Loop Testing	Test loop that iterates through available time slots.	Loop runs expected number of times, covering all slots.
	Data Flow Testing	Check data flow from appointment booking to database.	Appointment data should flow correctly to the database (no corruption).
Appointment Modification	Code Path Testing	Test all code paths for rescheduling appointments (valid, invalid).	Paths executed: success or error based on input.
	Condition Testing	Test conditions for validating new date and time slot.	Proper condition handling: reject invalid times, allow valid changes.
Appointment Cancellation	Code Path Testing	Test paths for cancelling an appointment (valid appointment, invalid).	Only valid appointments should be cancellable.
	Condition	Test conditions for validating user permissions before	Only users with permissions should be able to cancel

Component	Test Scenario	Test Case (Code-level)	Expected Outcome
	Testing	cancellation.	appointments.
Admin Panel Access	Code Path Testing	Test paths for different user roles (admin vs. regular users).	Correct path based on role: admin has full access, regular users limited.
	Condition Testing	Test if statements that control access to sensitive data (admin).	Only admin can access sensitive data and functions.
Security	Data Flow Testing	Test flow of sensitive data during appointment creation and updates.	Data is encrypted, follows proper security protocols.
	Code Path Testing	Test paths for handling invalid/malicious inputs (SQL injection, XSS).	Malicious inputs are blocked, and no unauthorized access occurs.