---

1. White Box Testing 📦

White Box Testing (also known as Clear Box, Glass Box, or Structural Testing) is a software testing technique where the internal structure, design, and coding of the software are tested to verify the flow of input-output and improve design, usability, and security. Unlike Black Box testing, the tester requires knowledge of the code and the internal workings of the system.

Key Aspects:

- **Code Visibility:** The tester has full visibility of the source code. They look at "how" the system processes data, not just "what" it produces.
- **Logic Verification:** It focuses on verifying the internal logic paths, ensuring that all decision points (if-else conditions), loops, and statements are executed and function correctly.
- **Security:** It is often used to find hidden security loopholes and vulnerabilities within the code that might not be apparent from the interface.
- **Levels:** It is primarily performed at the **Unit Testing** and **Integration Testing** levels.

---

2. Performance Testing with Example 🚀

Performance Testing is a non-functional testing technique used to determine how a system performs in terms of responsiveness and stability under a particular workload. It ensures the software is fast, scalable, and stable before it goes live.

Detailed Explanation:

1. **Speed (Response Time):** Determines whether the application responds quickly enough to user inputs.
2. **Scalability:** Checks the maximum user load the software application can handle.
3. **Stability:** Checks if the application remains stable under varying loads.

Types of Performance Testing:

- **Load Testing:** Testing with an expected number of concurrent users.
- **Stress Testing:** Testing beyond normal operational capacity (breaking point).
- **Endurance Testing:** Testing with a significant load over an extended period.

Example:

Imagine an e-commerce website like Amazon during a "Black Friday" sale.

- **Scenario:** The system normally handles 10,000 users. During the sale, traffic is expected to spike to 500,000 users.
- **The Test:** Performance engineers will simulate 500,000 virtual users accessing the site simultaneously.
- **Goal:** To verify if the "Checkout" page loads within 2 seconds (Speed) and if the server crashes (Stability) under this massive load.

---

3. Risk-Based Testing (RBT) ⚠️

Risk-Based Testing (RBT) is a testing strategy that prioritizes the testing of features and functions based on the risk of failure and the impact of that failure. It organizes testing efforts to ensure that the most critical parts of the application are tested first and most thoroughly.

Key Concepts:

1. **Risk Identification:** The team identifies potential risks (e.g., "The payment gateway might fail").
2. **Risk Analysis:** Risks are categorized by **Probability** (how likely is it to fail?) and **Impact** (how bad is it if it fails?).
   - *High Probability + High Impact* = Critical Priority.
3. **Optimization:** In projects with tight deadlines, RBT helps managers decide what to test when there isn't enough time to test everything. You test the "High Risk" areas first.
4. **Benefits:** It improves quality by focusing on critical defects earlier and reduces the cost of failure in production.

---

4. Black Box Testing 🎁

Black Box Testing (also known as Behavioral Testing) is a method where the functionality of the software is tested without knowing the internal code structure, implementation details, or internal paths. The tester interacts with the system's user interface, providing inputs and observing outputs.

Key Aspects:

- **User Perspective:** It simulates how a real user would interact with the system. The tester focuses on "what" the system does, not "how."
- **Input/Output:** The primary focus is on valid and invalid inputs and verifying if the correct

output is generated.
- **Requirement Based:** Test cases are built directly from the requirements specification document (SRS).
- **Levels:** It is applicable to all levels of testing: Unit, Integration, System, and Acceptance.

---

5. Compatibility Testing & Security Testing 🛡️

Compatibility Testing

This ensures that the software application functions as expected across different computing environments. It verifies that the software is compatible with:

- **Browsers:** Chrome, Firefox, Safari, Edge (Cross-browser testing).
- **Operating Systems:** Windows, Linux, macOS, Android, iOS.
- **Hardware:** Different memory sizes, processors, and network speeds.
- **Versions:** Backward compatibility with older versions of the software.

Security Testing

This is a type of non-functional testing intended to uncover vulnerabilities, threats, and risks in a software application and prevent malicious attacks from intruders.

- **Goal:** To identify potential weaknesses in the system that could result in the loss of information or reputation.
- **Key Areas:** Confidentiality (data is safe), Integrity (data is accurate), Authentication (user identity), and Authorization (access control).

---

6. Exploratory Testing 🧭

Exploratory Testing is an approach to software testing that is concisely described as simultaneous learning, test design, and test execution. Unlike scripted testing, there are no pre-defined test cases.

What I think about it:
It is a highly creative and intellectual process. It relies heavily on the skill, intuition, and experience of the tester.

- **Agile Friendly:** It fits perfectly in Agile environments where documentation is lean and cycles are short.
- **Bug Discovery:** It is often more effective at finding "edge case" bugs than scripted testing because testers can follow their curiosity ("What happens if I click this button 10 times quickly?").
- **Ad-hoc vs. Exploratory:** While similar, Exploratory is more structured than Ad-hoc; it often uses "Charters" or missions to guide the session.

---

7. Importance of Regression Testing & Explanation 🔄

Regression Testing is the practice of running a subset of existing test cases to ensure that recent code changes (fixes, enhancements, or configuration changes) have not adversely affected existing features.

Explanation:
Whenever a developer fixes a bug, there is a risk that their fix accidentally broke something else in the system (a "regression"). Regression testing verifies that the "old" code still works with the "new" changes.
Importance:

1. **Safety Net:** It provides confidence that the system is stable after changes.
2. **Continuous Integration:** It is the backbone of DevOps and CI/CD pipelines; automated regression suites run every time code is committed.
3. **Cost Saving:** Catching regressions early prevents critical bugs from reaching production, where they are expensive to fix.

---

8. Explain Any Two Non-Functional Testing ⚙️

Non-functional testing checks the "quality" attributes of the system, rather than specific behaviors.

1. Usability Testing:

- **Definition:** Evaluating a product by testing it with representative users. It measures how easy and user-friendly the software is.
- **Focus:** It checks navigation, layout, text clarity, and how intuitive the interface is. If a user cannot figure out how to use a feature, it is a usability failure, even if the code works perfectly.

2. Reliability Testing:

- **Definition:** Verifying whether the software can perform a failure-free operation for a specified period of time in a specified environment.
- **Focus:** It checks for consistent performance and accurate results. A system is reliable if it produces the same output for the same input every single time without crashing.

---

9. Statement Coverage & Branch Coverage Testing 📊

These are White Box testing metrics used to measure the completeness of test cases.

1. Statement Coverage:

- **Definition:** A metric that calculates the percentage of executable statements in the source code that have been executed at least once during testing.
- **Formula:** (Number of executed statements / Total number of statements) * 100.
- **Goal:** To ensure every line of code is run. However, 100% statement coverage does not guarantee bug-free code (it might miss false conditions).

2. Branch Coverage (Decision Coverage):

- **Definition:** A metric that ensures every possible branch from each decision point (e.g., if-else, case statements) is executed at least once.
- **Goal:** To ensure that both the TRUE and FALSE sides of every decision are tested. It is a stronger metric than statement coverage.

---

10. Explain Any Two Functional Testing 🛠️

Functional testing validates the software system against the functional requirements/specifications.

1. Unit Testing:

- **Definition:** The testing of individual units or components of a software application. A

"unit" is the smallest testable part of any software (e.g., a single function or method).
- **Who:** Usually performed by developers.
- **Goal:** To isolate a section of code and verify its correctness.

2. System Testing:

- **Definition:** Testing the complete and integrated software product to evaluate the system's compliance with specified requirements.
- **Who:** Performed by the independent testing team.
- **Goal:** To verify the end-to-end flow of the application (e.g., login -> add to cart -> payment -> logout) in an environment that mimics production.

---

11. Boundary Value Analysis (BVA) and Equivalence Class Partitioning (ECP) 📏

These are Black Box test design techniques used to reduce the number of test cases while maintaining high coverage.

1. Equivalence Class Partitioning (ECP):

- **Concept:** Divides the input data into different classes (partitions). The theory is that if one value in a class works, all values in that class should work.
- **Example:** An age field accepts 18 to 60.
  - *Valid Partition:* 18 to 60.
  - *Invalid Partition 1:* Less than 18.
  - *Invalid Partition 2:* Greater than 60.
  - *Test Cases:* You only need to test one value from each partition (e.g., 25, 10, 75).

2. Boundary Value Analysis (BVA):

- **Concept:** Bugs are most likely to lurk at the boundaries of input partitions rather than the center. BVA focuses on testing the edges.
- **Example:** Age field accepts 18 to 60.
  - *Boundaries:* 18 and 60.
  - *Test Values:* Min (18), Min-1 (17), Max (60), Max+1 (61).

---

12. Differentiate Between Functional Testing and

Non-functional Testing 🆚

Feature	Functional Testing	Non-Functional Testing
1. Objective	Verifies **what** the system does (features/functions).	Verifies **how** the system performs (behavior/qualities).
2. Requirement Source	Based on Functional Requirements (SRS, Business Rules).	Based on Non-Functional Requirements (Performance, Security specs).
3. Focus	Focuses on user requirements, inputs, and outputs.	Focuses on user expectations like speed, scalability, and reliability.
4. Example Types	Unit, Integration, System, Acceptance testing.	Performance, Load, Stress, Usability, Security testing.
5. Execution Order	Generally performed **before** non-functional testing.	Generally performed **after** functional testing.
6. Defect Type	Finds functional defects (e.g., "Login button doesn't work").	Finds architectural/performance bottlenecks (e.g., "Login takes 10 seconds").
7. Manual vs Auto	Can be easily executed manually.	Often requires automation tools (like JMeter) to simulate load.

---

13. Test Case Design Techniques: Reviews,

Walkthroughs, Inspection 📝

These are **Static Testing** techniques where documents and code are reviewed without executing the code.

i) Informal Reviews:

- **Description:** The most casual method. A document author asks a colleague to look over their work (code or design) and give feedback.
- **Structure:** No formal process, no minutes taken, no checklist.
- **Goal:** Quick, low-cost improvement.

ii) Walkthroughs:

- **Description:** A meeting led by the **author** of the document/code. The author guides the participants through the document to achieve a common understanding and gather feedback.
- **Structure:** Semi-formal. Participants may prepare beforehand.
- **Goal:** Knowledge transfer and finding anomalies.

iii) Inspection:

- **Description:** The most formal and rigorous type of review. It is led by a trained **moderator** (not the author). It involves defined roles (Scribe, Reader, Inspector), entry/exit criteria, and formal logging of defects.
- **Structure:** Highly structured with checklists and preparation rules.
- **Goal:** To find defects efficiently before dynamic testing begins.

---

14. Cookies Testing with Example 🍪

What is Cookies Testing?
Cookies are small files stored on a user's computer by a website to maintain the "state" of the user (e.g., login status, cart contents). Cookies testing verifies that cookies are created, stored, retrieved, and expire correctly.
Example:
Consider an online shopping site.

1. **Test Case 1 (Session Cookie):** A user logs in. A cookie is created. If the user closes the browser and reopens it, are they still logged in (if "Remember Me" was checked)?
2. **Test Case 2 (Security):** Can the cookie data be read by another malicious website? (Testing for Cross-Site Scripting vulnerabilities).

3. **Test Case 3 (Disabling):** If the user disables cookies in their browser settings, does the website handle it gracefully (e.g., show a message) or does it crash?
4. **Test Case 4 (Corruption):** If a tester manually edits the cookie content in the browser, does the system reject the invalid cookie?

---

15. Loop Coverage Testing and Types ➿

Loop Coverage is a White Box testing technique that focuses on the validity of loop constructs (like for, while, do-while). Loops are complex and common sources of errors (e.g., infinite loops, off-by-one errors).

Types of Loops to Test:

1. **Simple Loops:** A single loop with no nesting.
   - *Tests:* Skip loop entirely, pass through once, pass through twice, pass through m times, pass through max times.
2. **Nested Loops:** A loop inside another loop.
   - *Tests:* Start with the innermost loop while holding outer loops at minimum values, then work outward.
3. **Concatenated Loops:** Loops that follow one another.
   - *Tests:* Tested as simple loops if independent. If the second loop relies on the first loop's counter, they are treated like nested loops.
4. **Unstructured Loops:** Loops that jump into or out of the loop body (using goto or break in complex ways). These should be refactored and tested heavily.

---

16. Adhoc Testing with Example 🎲

Adhoc Testing is the least formal type of testing. It is performed without any planning, documentation, or test case design. The tester simply "plays" with the application to try and break it.

Characteristics:

- Also known as "Monkey Testing" or "Random Testing."
- Success depends entirely on the creativity of the tester.
- Difficult to reproduce bugs because steps aren't documented.

Example:

A tester is testing a Calculator app.
Instead of following a plan (e.g., "Test 1+1"), the tester starts randomly mashing buttons: 5 / 0 * 9999 + - - =.

- **Goal:** To see if the application crashes or displays a weird error message like "NaN" or "Infinity" when it shouldn't.

---

17. Differentiate Unit Testing and Integration Testing 🧩

Feature	Unit Testing	Integration Testing
1. Definition	Testing individual modules or components in isolation.	Testing the interface and interaction between combined modules.
2. Scope	Smallest scope (single function/class).	Medium scope (group of modules).
3. Performed By	Typically performed by **Developers**.	Performed by **Developers** or **Testers**.
4. Visibility	White Box (requires code knowledge).	Can be White Box or Black Box.
5. Goal	To check if the individual piece of code is correct.	To check if data flows correctly between modules (Interfaces).
6. Timing	First level of testing performed.	Performed after Unit Testing and before System Testing.
7. Defect Finding	Finds logic errors, calculation errors within a function.	Finds interface errors, data format mismatches between modules.