Software Testing and Quality Assurance

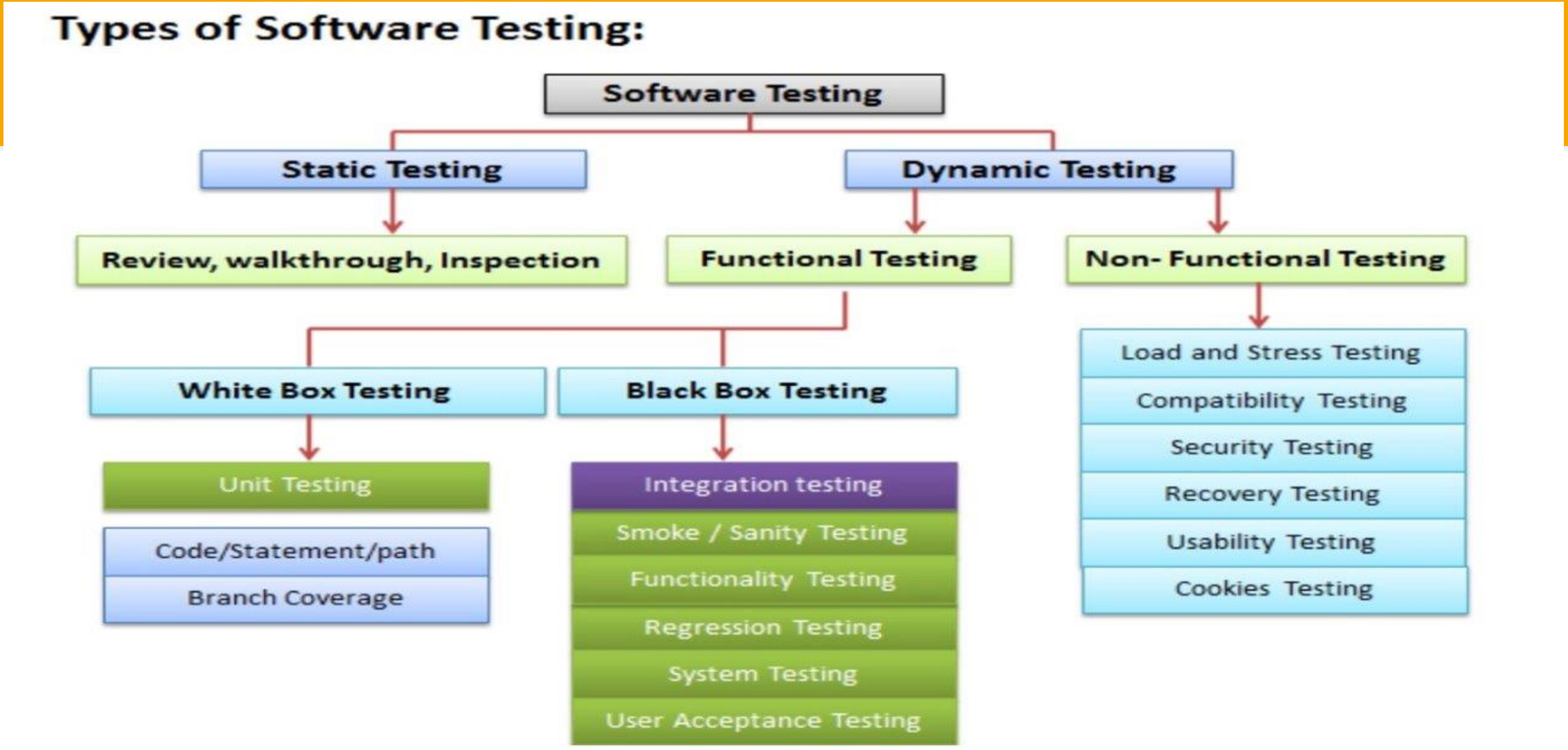
Module-1

By Prof. Pallavi Mahajan

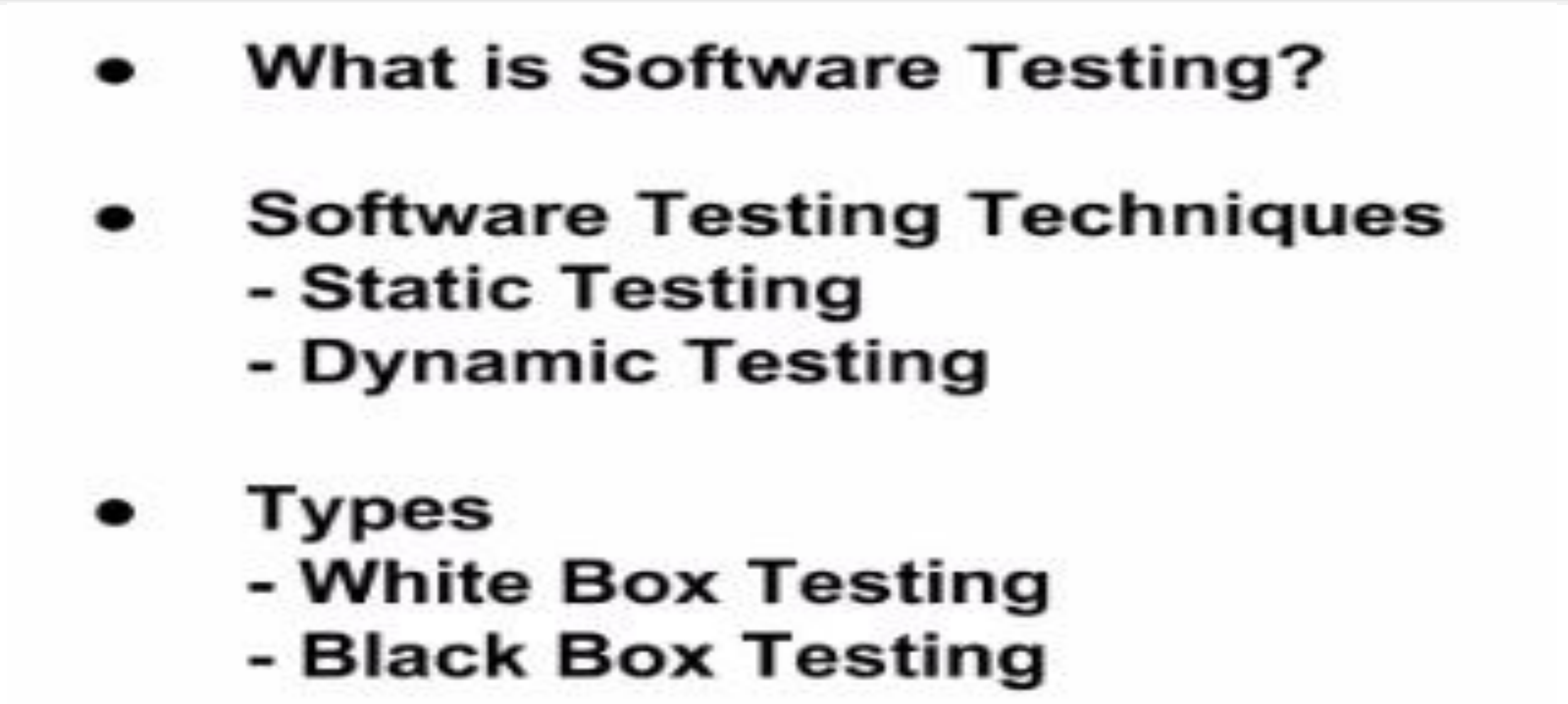
Contains

Introduction to Testing techniques

Testing Types

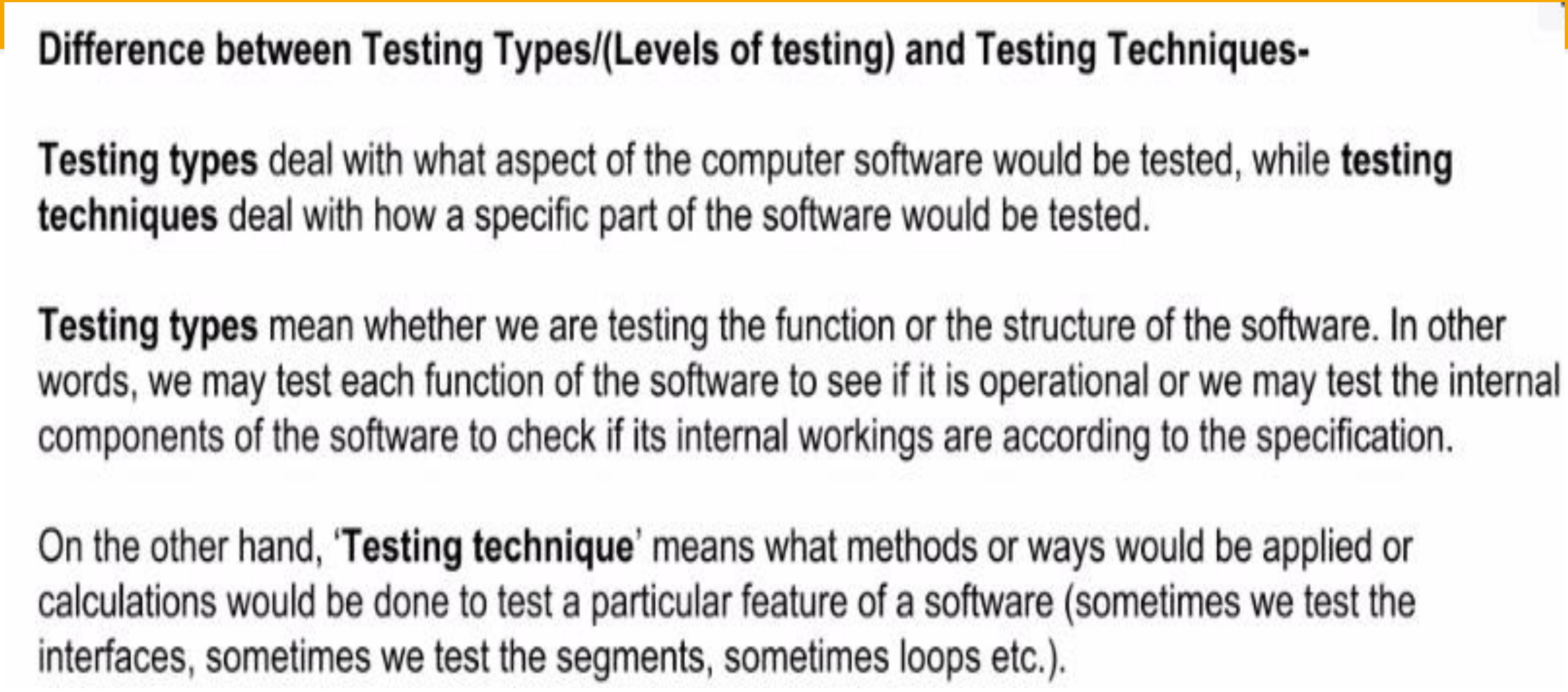


Testing techniques



- What is Software Testing?
- Software Testing Techniques
  - Static Testing
  - Dynamic Testing
- Types
  - White Box Testing
  - Black Box Testing

Testing techniques



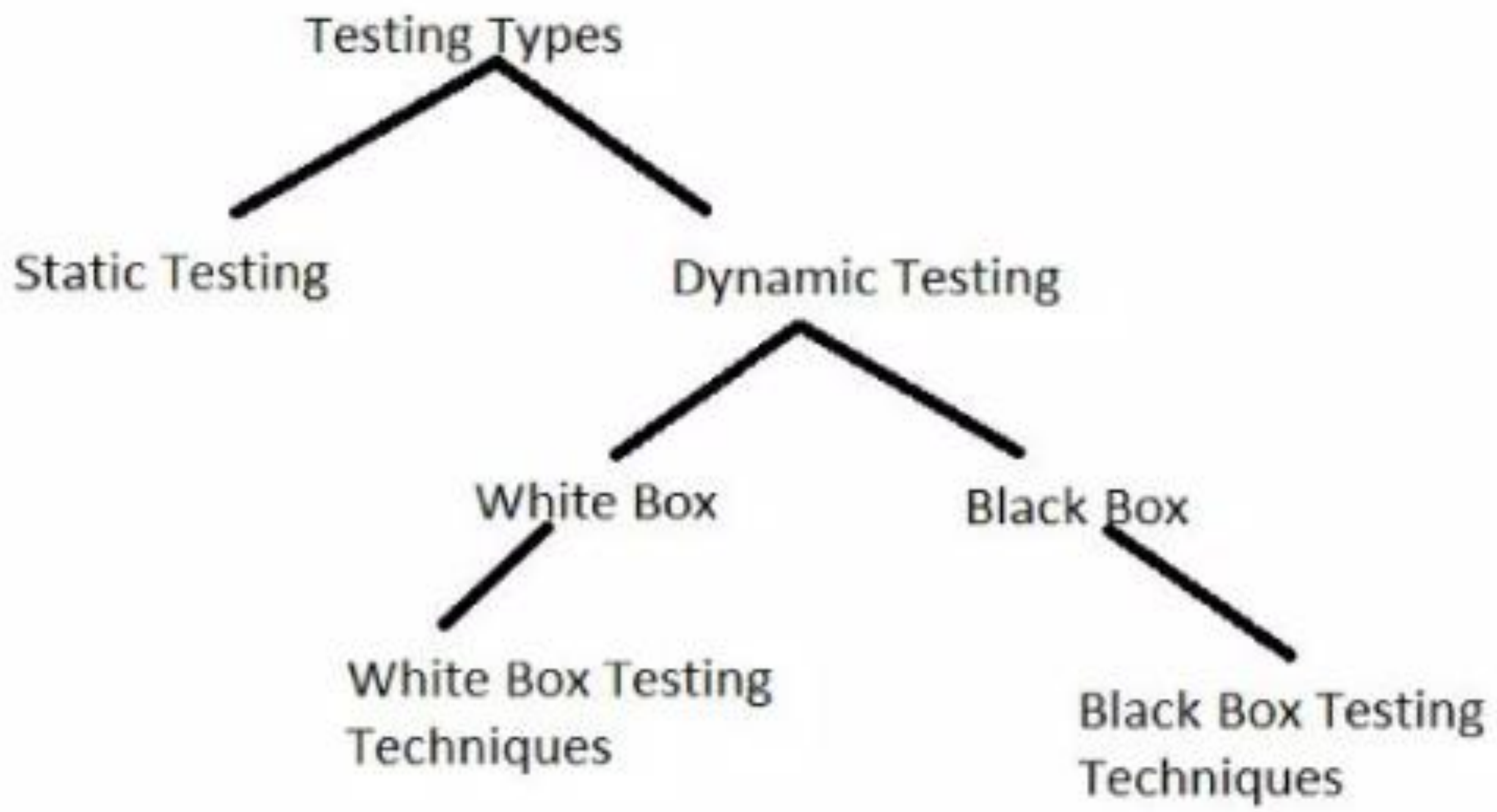
Difference between Testing Types/(Levels of testing) and Testing Techniques-

Testing types deal with what aspect of the computer software would be tested, while **testing techniques** deal with how a specific part of the software would be tested.

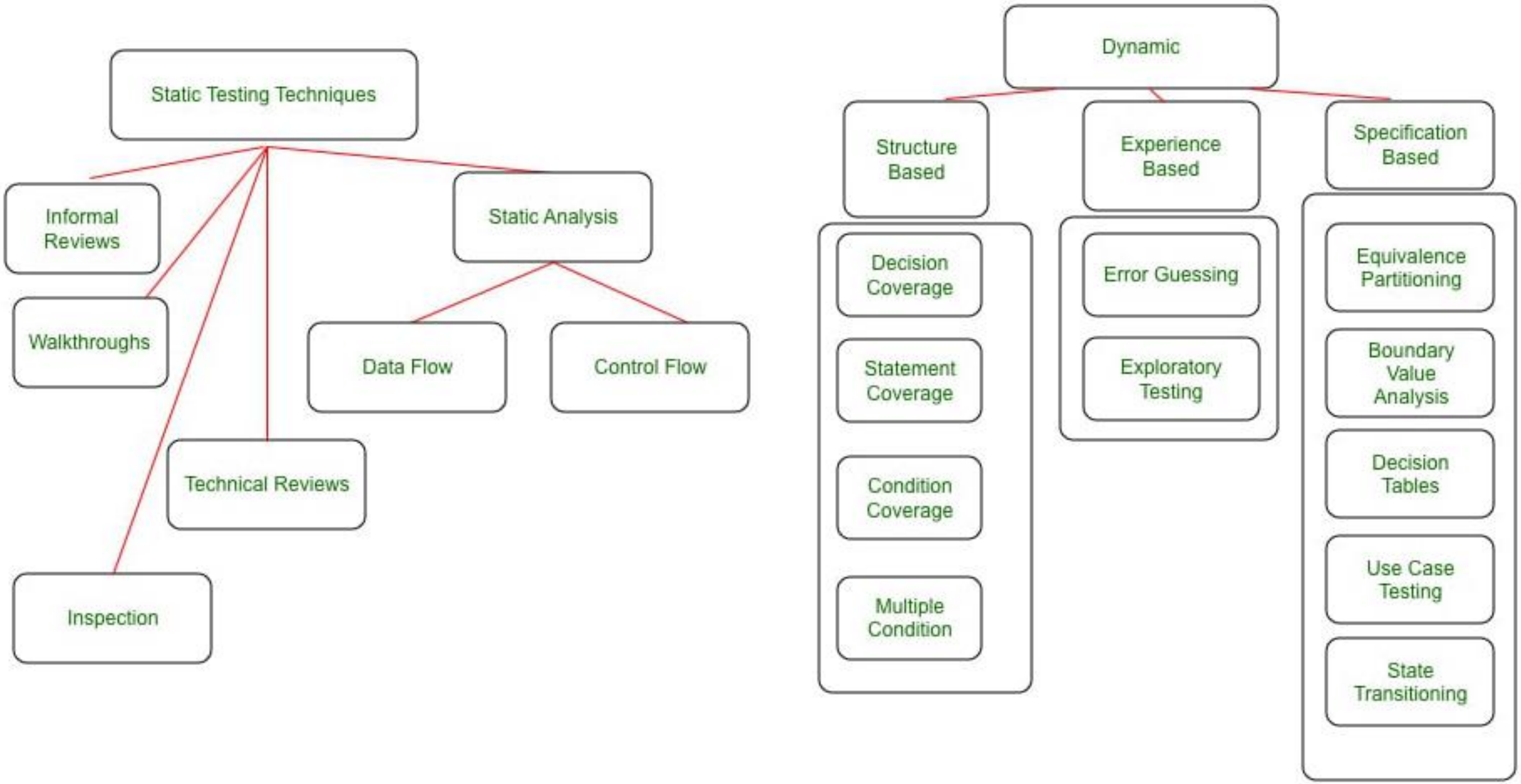
Testing types mean whether we are testing the function or the structure of the software. In other words, we may test each function of the software to see if it is operational or we may test the internal components of the software to check if its internal workings are according to the specification.

On the other hand, '**Testing technique**' means what methods or ways would be applied or calculations would be done to test a particular feature of a software (sometimes we test the interfaces, sometimes we test the segments, sometimes loops etc.).

Classification



Testing techniques



Static Testing

In Static Testing, the main aim is to check whether the work being done is going as per the set standards or not. It does not involve the execution of the software.

Static testing is generally in the form of :

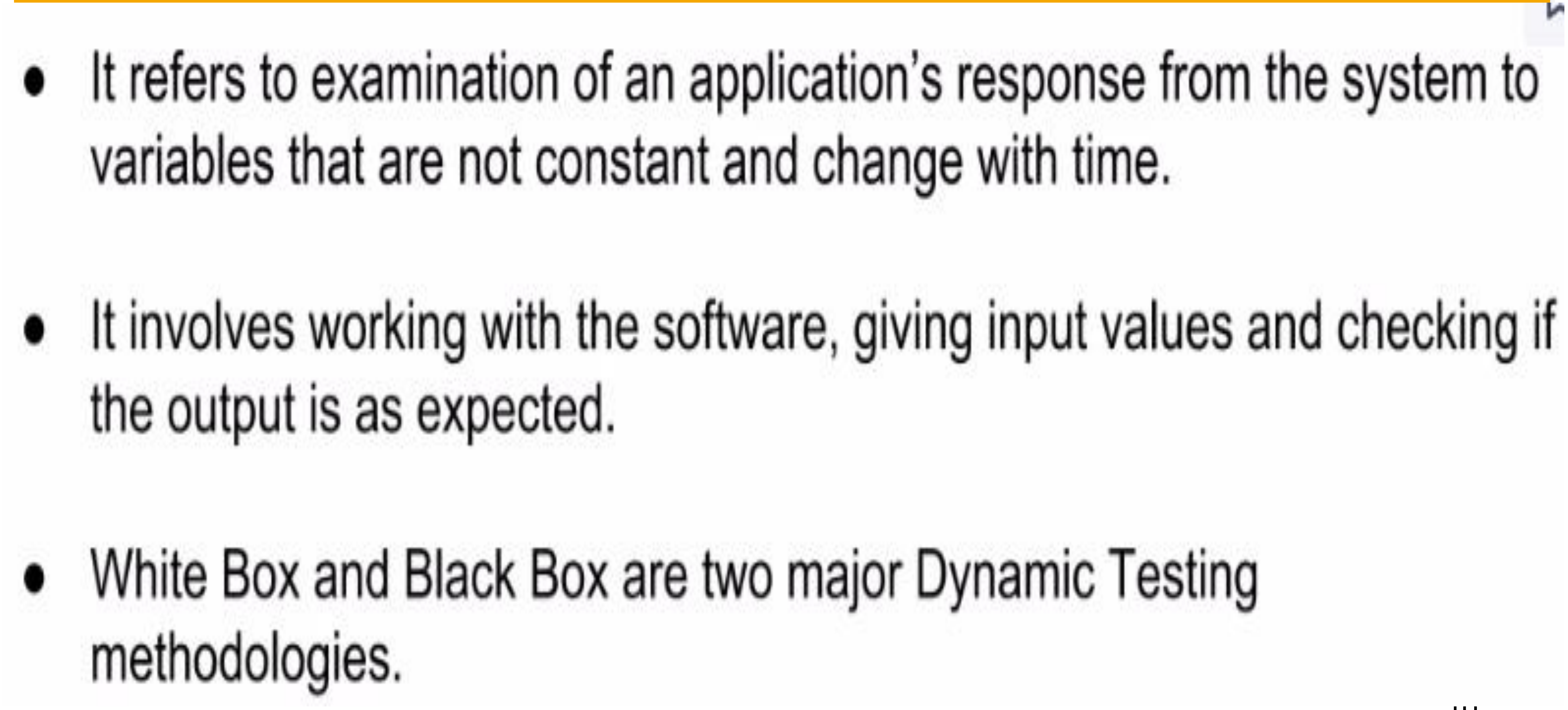
- Reviews
- Walkthroughs
- Inspection

For

→ Source code
→ Documentation
→ Design Document
→ Requirements Specification
→ Project Plan and other project related documents

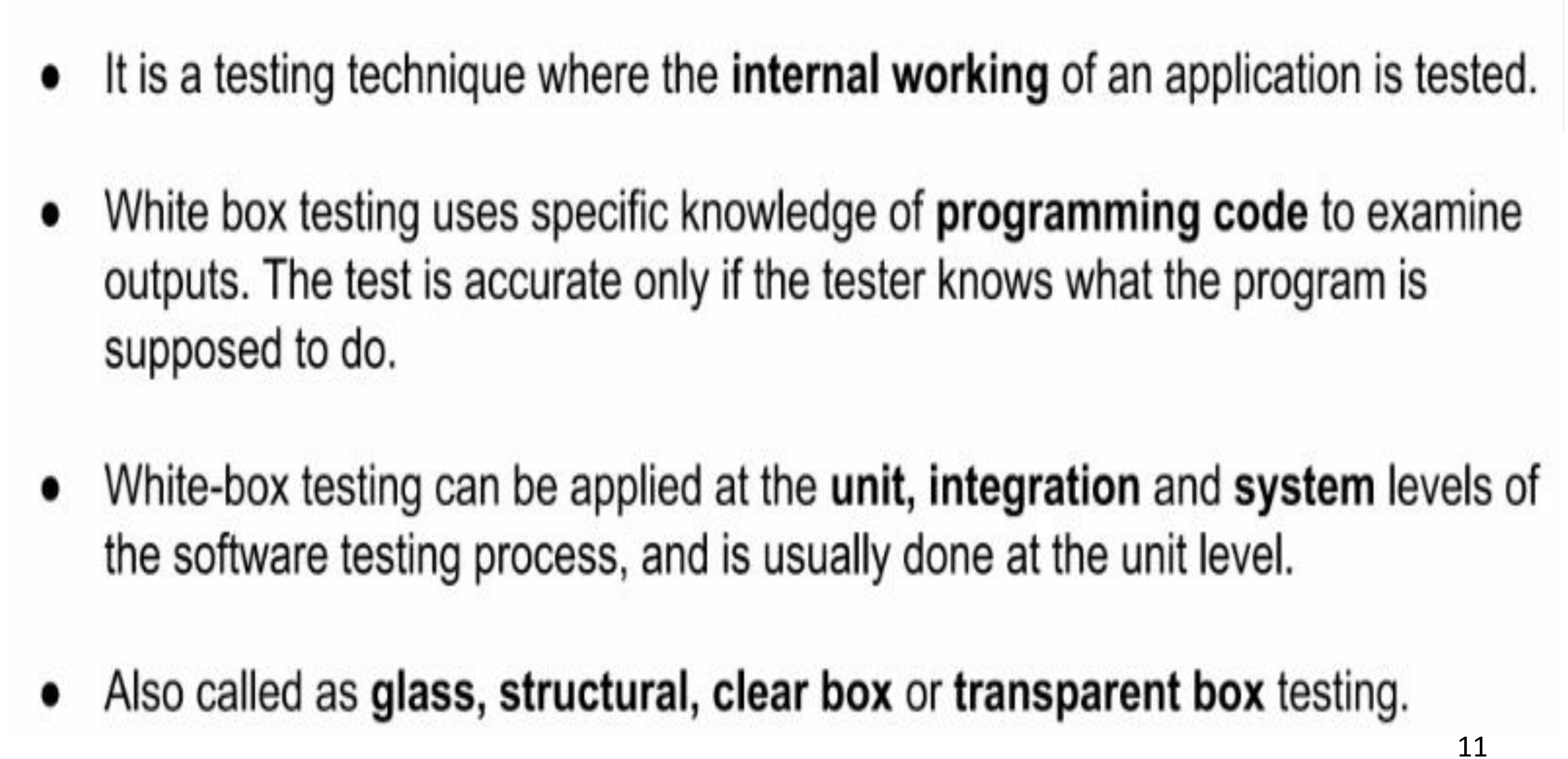
The goal of static testing is to prevent defects as early as possible.

Dynamic Testing



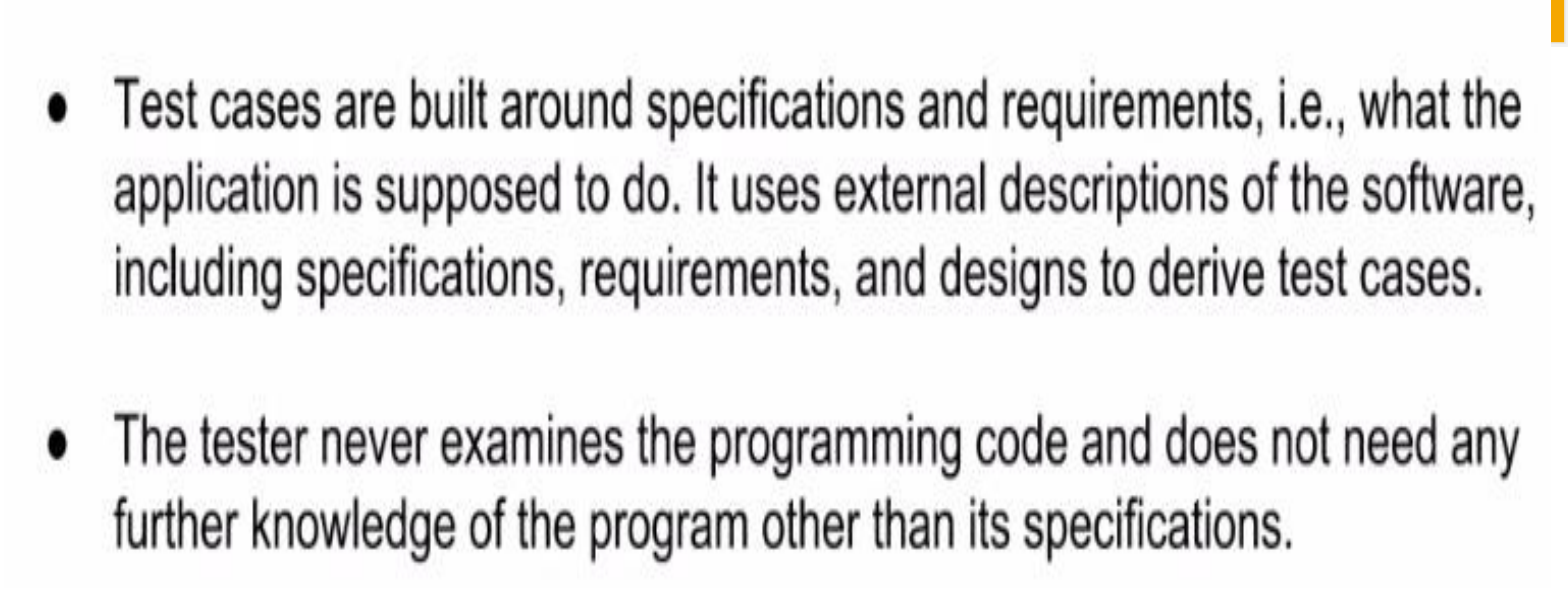
- It refers to examination of an application’s response from the system to variables that are not constant and change with time.
- It involves working with the software, giving input values and checking if the output is as expected.
- White Box and Black Box are two major Dynamic Testing methodologies.

White Box Testing



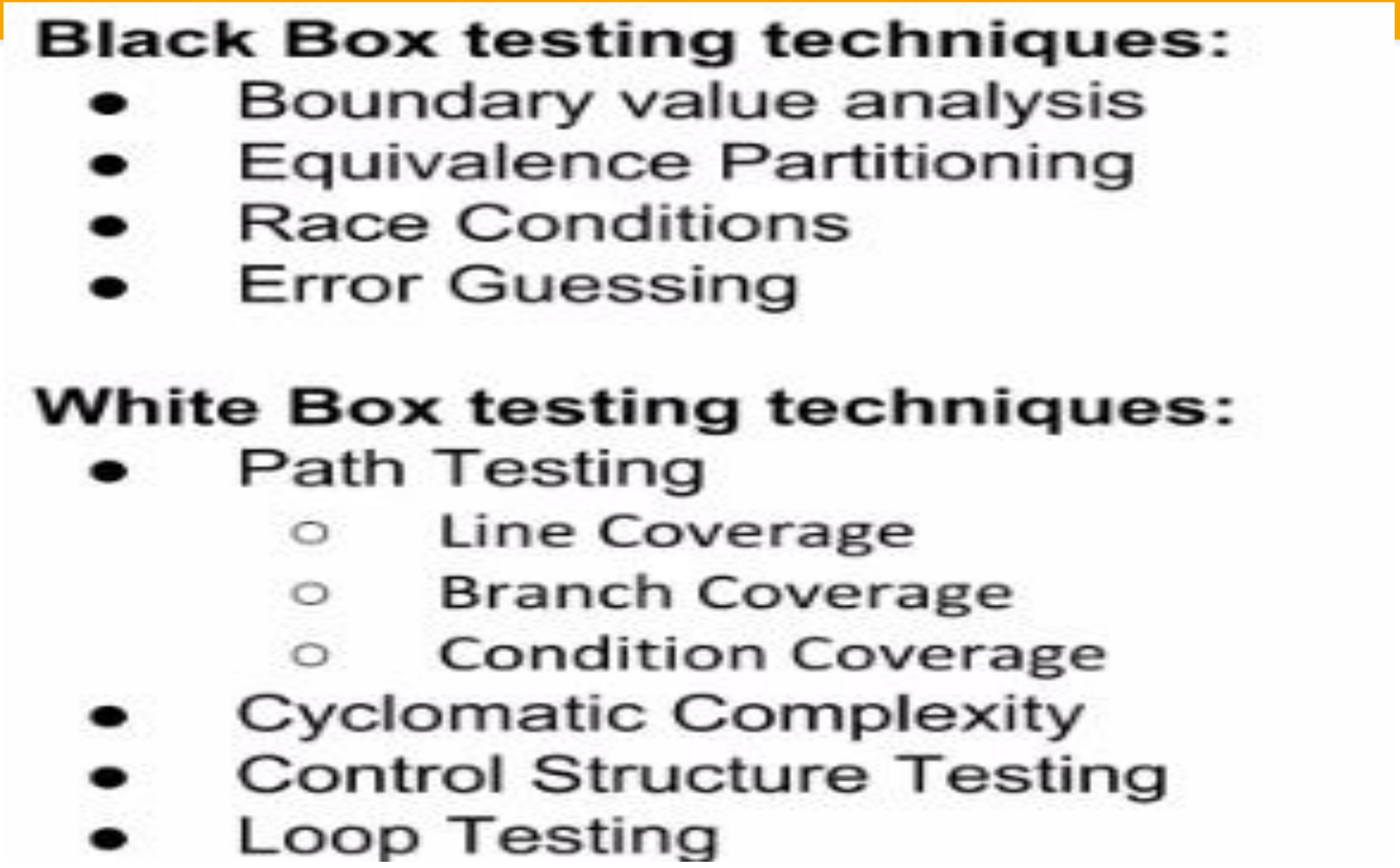
- It is a testing technique where the **internal working** of an application is tested.
- White box testing uses specific knowledge of **programming code** to examine outputs. The test is accurate only if the tester knows what the program is supposed to do.
- White-box testing can be applied at the **unit, integration** and **system** levels of the software testing process, and is usually done at the unit level.
- Also called as **glass, structural, clear box** or **transparent box** testing.

Black Box Testing



- Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases.
- The tester never examines the programming code and does not need any further knowledge of the program other than its specifications.

Black Box and White Box Testing Techniques



Black Box testing techniques:

- Boundary value analysis
- Equivalence Partitioning
- Race Conditions
- Error Guessing

White Box testing techniques:

- Path Testing
  - Line Coverage
  - Branch Coverage
  - Condition Coverage
- Cyclomatic Complexity
- Control Structure Testing
- Loop Testing

Introduction to Testing strategies

Strategic Approach

- To perform effective testing, you should conduct effective technical reviews. By doing this, many errors will be eliminated before testing commences.
- Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
- Different testing techniques are appropriate for different software engineering approaches and at different points in time.
- Testing is conducted by the developer of the software and (for large projects) an independent test group.
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

Who Tests the Software?

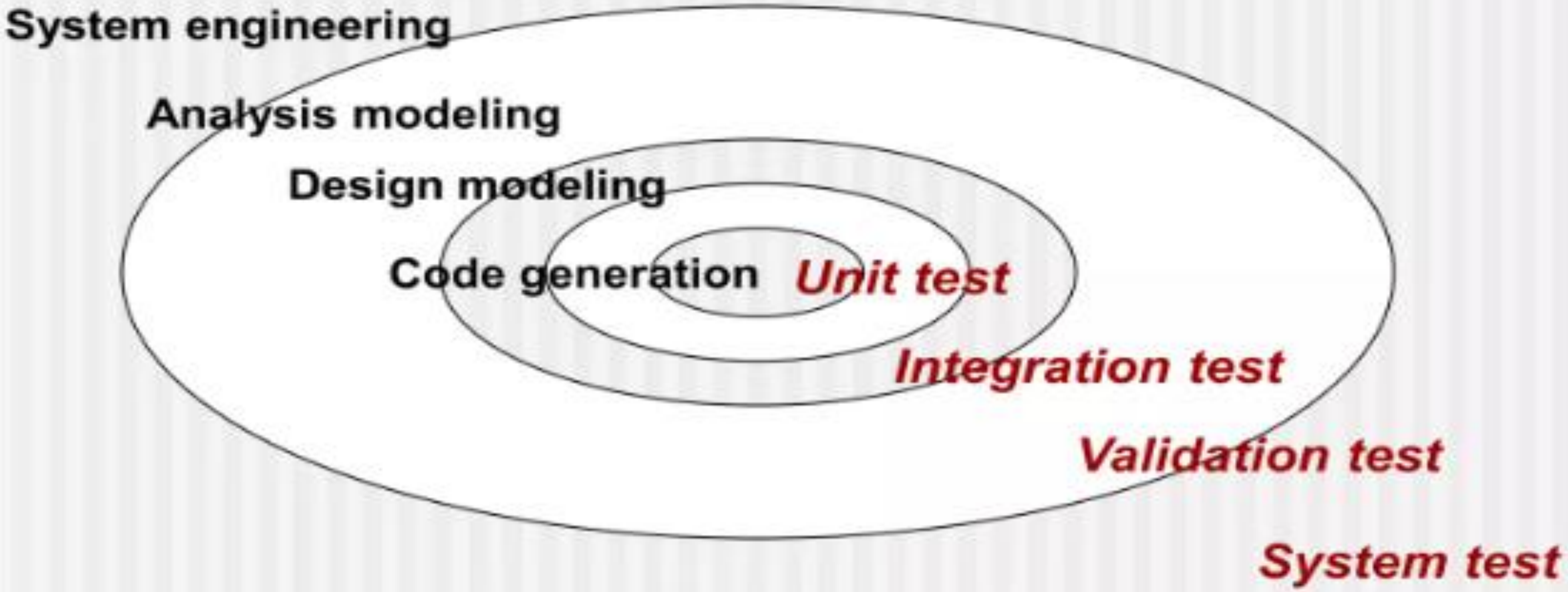
developer

Understands the system
but, will test "gently"
and, is driven by "delivery"

independent tester

Must learn about the system,
but, will attempt to break it
and, is driven by quality

Testing Strategy



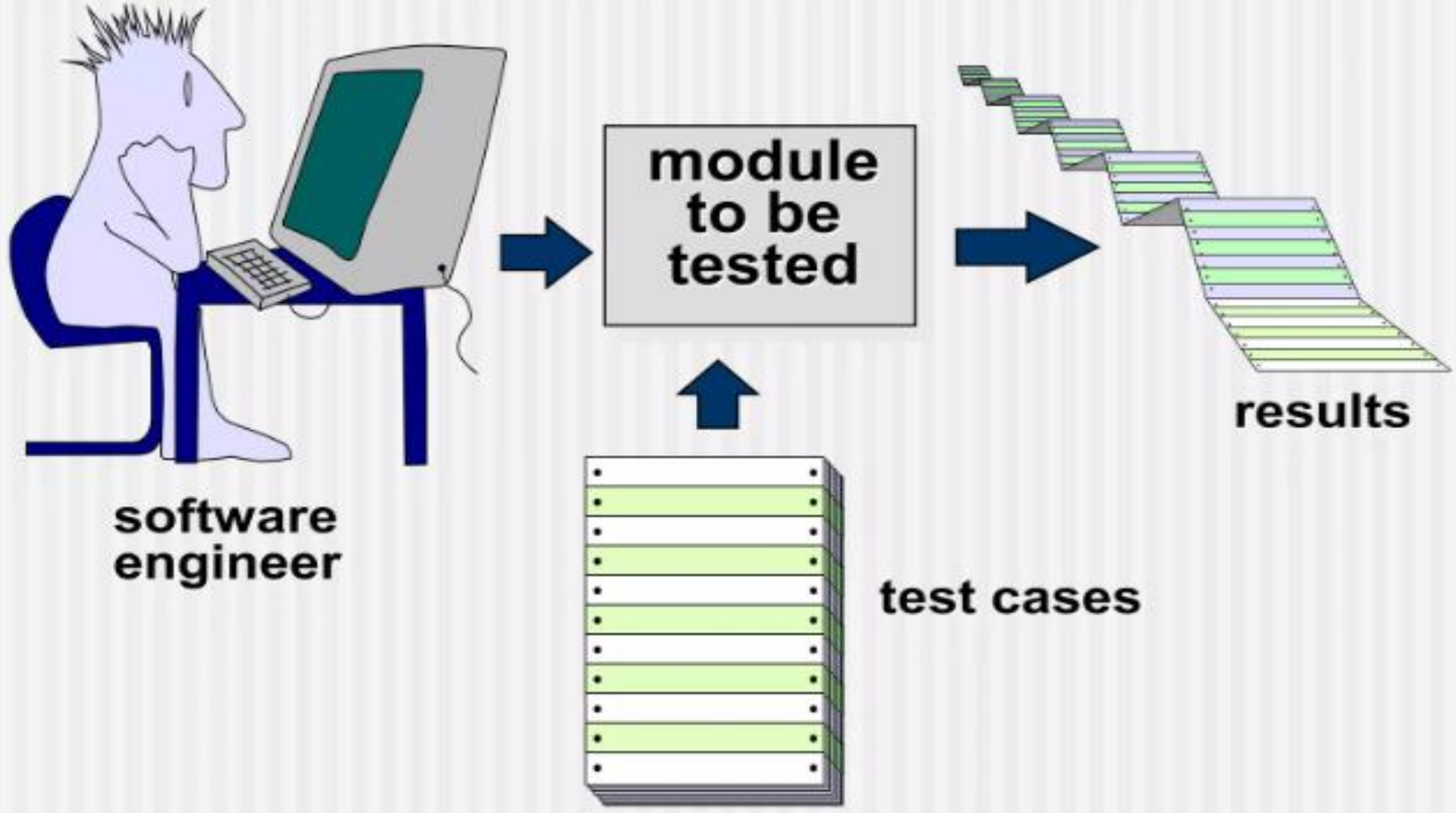
Testing Strategy

- We begin by ‘testing-in-the-small’ and move toward ‘testing-in-the-large’
- For conventional software
  - The module (component) is our initial focus
  - Integration of modules follows
- For OO software
  - our focus when “testing in the small” changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

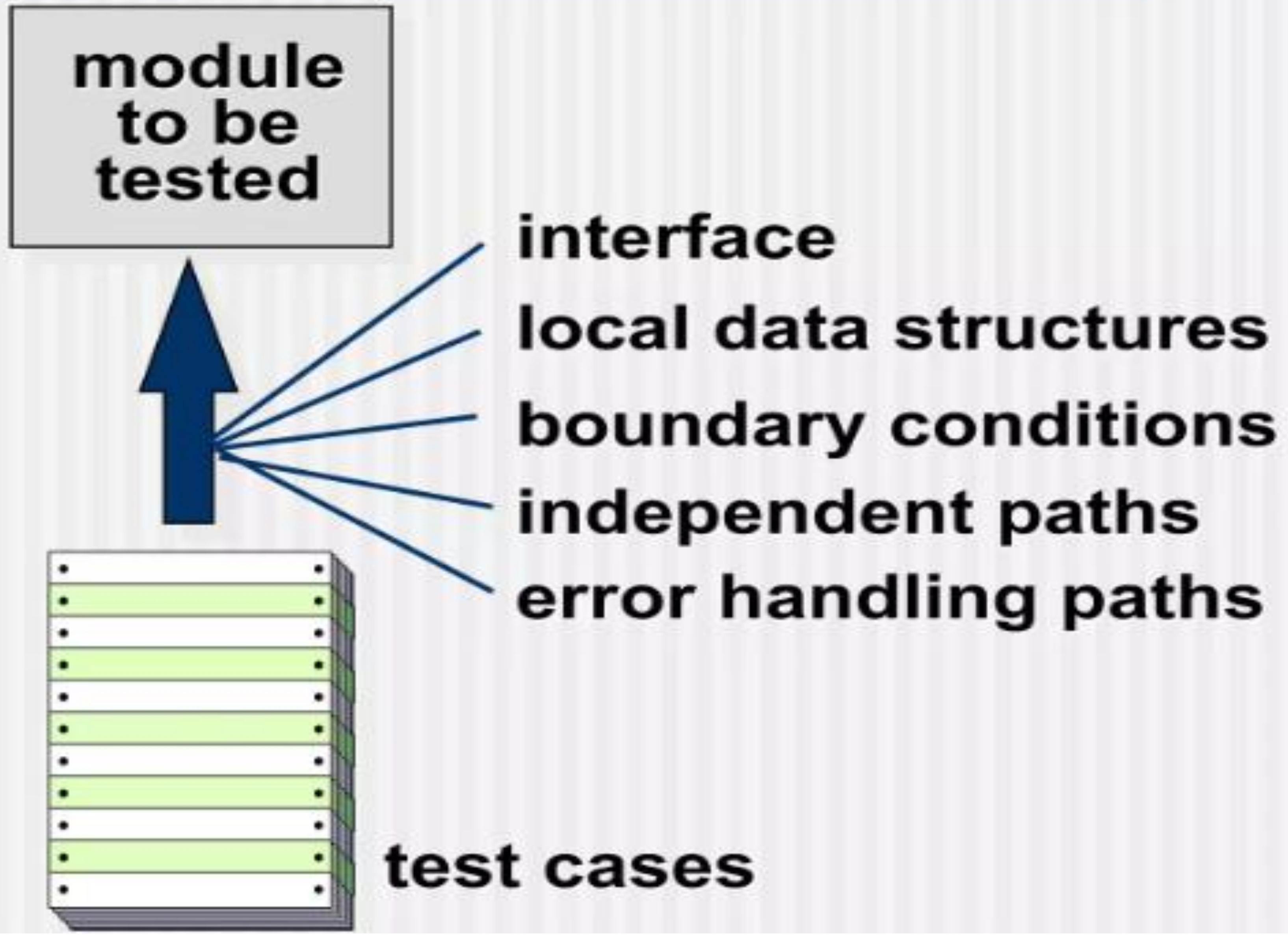
Strategic Issues

- Specify product requirements in a quantifiable manner long before testing commences.
- State testing objectives explicitly.
- Understand the users of the software and develop a profile for each user category.
- Develop a testing plan that emphasizes “rapid cycle testing.”
- Build “robust” software that is designed to test itself
- Use effective technical reviews as a filter prior to testing
- Conduct technical reviews to assess the test strategy and test cases themselves.
- Develop a continuous improvement approach for the testing process.

Unit Testing



Unit Testing

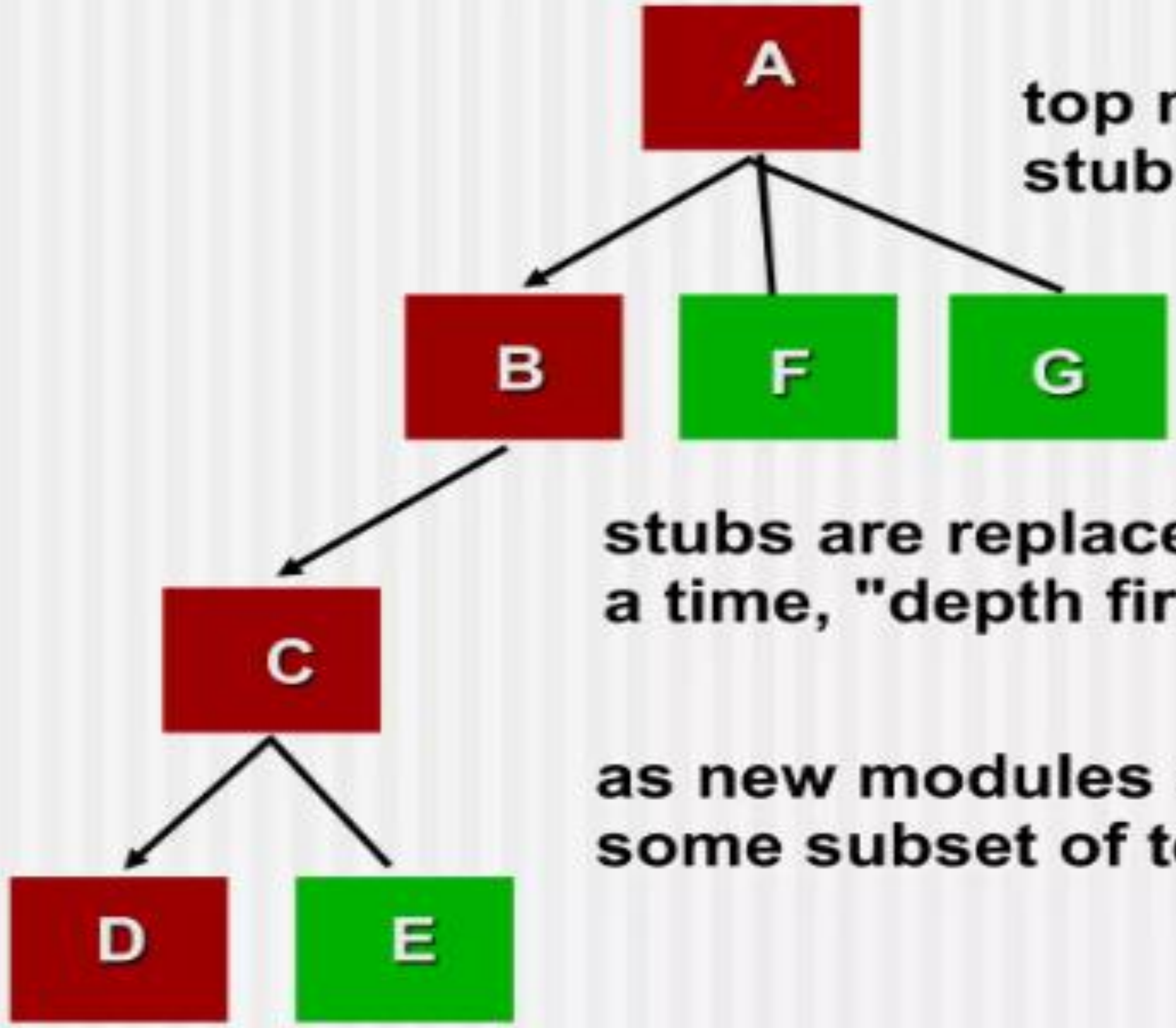


Integration Testing Strategies

Options:

- **the “big bang” approach**
- **an incremental construction strategy**

Top Down Integration

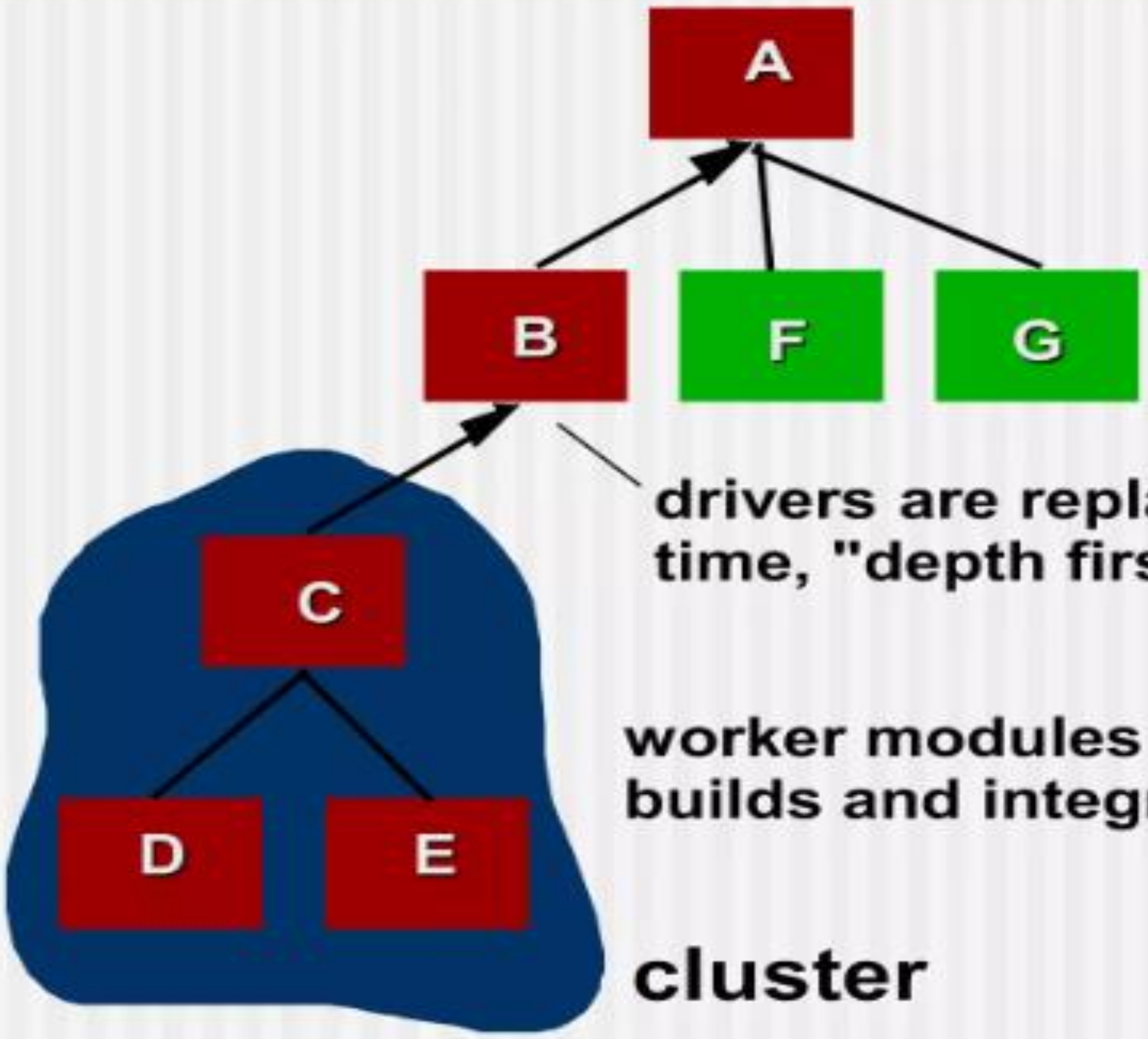


top module is tested with stubs

stubs are replaced one at a time, "depth first"

as new modules are integrated, some subset of tests is re-run

Bottom-Up Integration



drivers are replaced one at a time, "depth first"

worker modules are grouped into builds and integrated

cluster

OO Testing Strategy

- class testing is the equivalent of unit testing
  - operations within the class are tested
  - the state behavior of the class is examined
- integration applied three different strategies
  - thread-based testing—integrates the set of classes required to respond to one input or event
  - use-based testing—integrates the set of classes required to respond to one use case
  - cluster testing—integrates the set of classes required to demonstrate one collaboration

WebApp Testing - I

- The content model for the WebApp is reviewed to uncover errors.
- The interface model is reviewed to ensure that all use cases can be accommodated.
- The design model for the WebApp is reviewed to uncover navigation errors.
- The user interface is tested to uncover errors in presentation and/or navigation mechanics.
- Each functional component is unit tested.

WebApp Testing - II

- Navigation throughout the architecture is tested.
- The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration.
- Security tests are conducted in an attempt to exploit vulnerabilities in the WebApp or within its environment.
- Performance tests are conducted.
- The WebApp is tested by a controlled and monitored population of end-users. The results of their interaction with the system are evaluated for content and navigation errors, usability concerns, compatibility concerns, and WebApp reliability and performance.

High Order Testing

- Validation testing
  - Focus is on software requirements
- System testing
  - Focus is on system integration
- Alpha/Beta testing
  - Focus is on customer usage
- Recovery testing
  - forces the software to fail in a variety of ways and verifies that recovery is properly performed
- Security testing
  - verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration
- Stress testing
  - executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
- Performance Testing
  - test the run-time performance of software within the context of an integrated system

Test Planning and Design

Test Planning and Design

- **Test Planning**: Preparation for test execution, including defining scope, resources, effort, schedule, and budget.

- **Test Design**: Critical phase to identify system features, objectives, and design test cases based on system requirements.

Test Planning Overview

❖ **Purpose of Test Planning**: Organize for test execution.

➢ Framework, scope, resource details, effort, schedule, budget.

➢ Scope includes managerial aspects, not detailed techniques.

Test Planning Overview

❖ Critical Phase of Software Testing

- Identify system features to be tested.
- Define objectives of test cases.
- Design test cases and test steps.

Test-Driven Development (TDD)

What is TDD?

- Design and implement test cases before writing production code.

- Key practice in agile methodologies like XP (Extreme Programming).

Test-Driven Development (TDD)

Agile Characteristics:

- **Incremental development.**
- **Unit and acceptance tests conducted by programmers and customers.**
- **Frequent regression testing.**
- **Writing test code before production code.**

Monitoring and Measuring Test Execution

Monitoring and Measuring Test Execution

Importance of Monitoring:

- Measure testing progress and system quality.
- Trigger corrective and preventive actions based on metrics.

Metrics Types:

- **Test Execution Metrics:** Measures the process of executing test cases.
- **Defect Monitoring Metrics:** Measures defects found during execution.

Test Metrics and Evaluation

Key Metrics to Track:

- **Effectiveness of Testing**: Measures the effectiveness of test techniques (e.g., number of defects detected).
- **Productivity**: Test cases designed and executed per day.
- **Quality**: Defects detected per week of testing.

Test Case Effectiveness Metric

Objective:

- Measure defect revealing ability of test cases.
- Analyze test cases that failed to reveal defects (Test Case Escapes - TCE).
- Improve test design process based on TCE.

Test Effort Effectiveness Metric

Evaluation of Testing Effectiveness:

- Measure defects found by customers that were missed by testing.
- Helps identify missed areas in testing before release.

Test Tools and Automation

Test Tools and Automation

Labor-Intensive Nature of Software Testing:

- Manual generation and execution of test cases are time-consuming.

Importance of Test Automation:

- Increases productivity, regression coverage, and reduces testing duration.
- Improves test case effectiveness and software maintenance.

Benefits of Test Automation

Key Benefits:

- Increased productivity of testers.
- Better regression test coverage.
- Reduced testing time and cost.
- Consistent test execution results and simplified debugging.

Automation Considerations

Challenges with Test Automation:

- High initial setup time and resources.
- Needs maintenance with each modification in the system.
- Cannot replace manual testing for all cases (usability, robustness, etc.).

Test Automation Prerequisites

Conditions for Successful Automation:

- Well-defined test cases to automate.
- Availability of test tools and infrastructure.
- Experienced professionals in automation.
- Budget for tool procurement.

Test Team Organization and Management

Test Team Organization and Management

- **Distributed Testing:**
  - Testing occurs at different stages: unit, integration, system, and acceptance.
- **Team Structure:**
  - Unit testing handled by programmers.
  - System integration testing handled by integration test engineers.
  - System-level testing performed by a dedicated system test group.

User Acceptance Testing (UAT)

Role of User Acceptance Testing:

- Performed by a temporary team composed of integration engineers, system test engineers, and customer support engineers.
- Ensures that the system will pass the final acceptance test by the client.

Test Engineer Management

Hiring and Retaining Test Engineers:

- Importance of hiring skilled test engineers through interviews.
- Treat test engineers as professionals and integral to the team.
- Recognize their efforts at the same level as development teams.

Thank You..