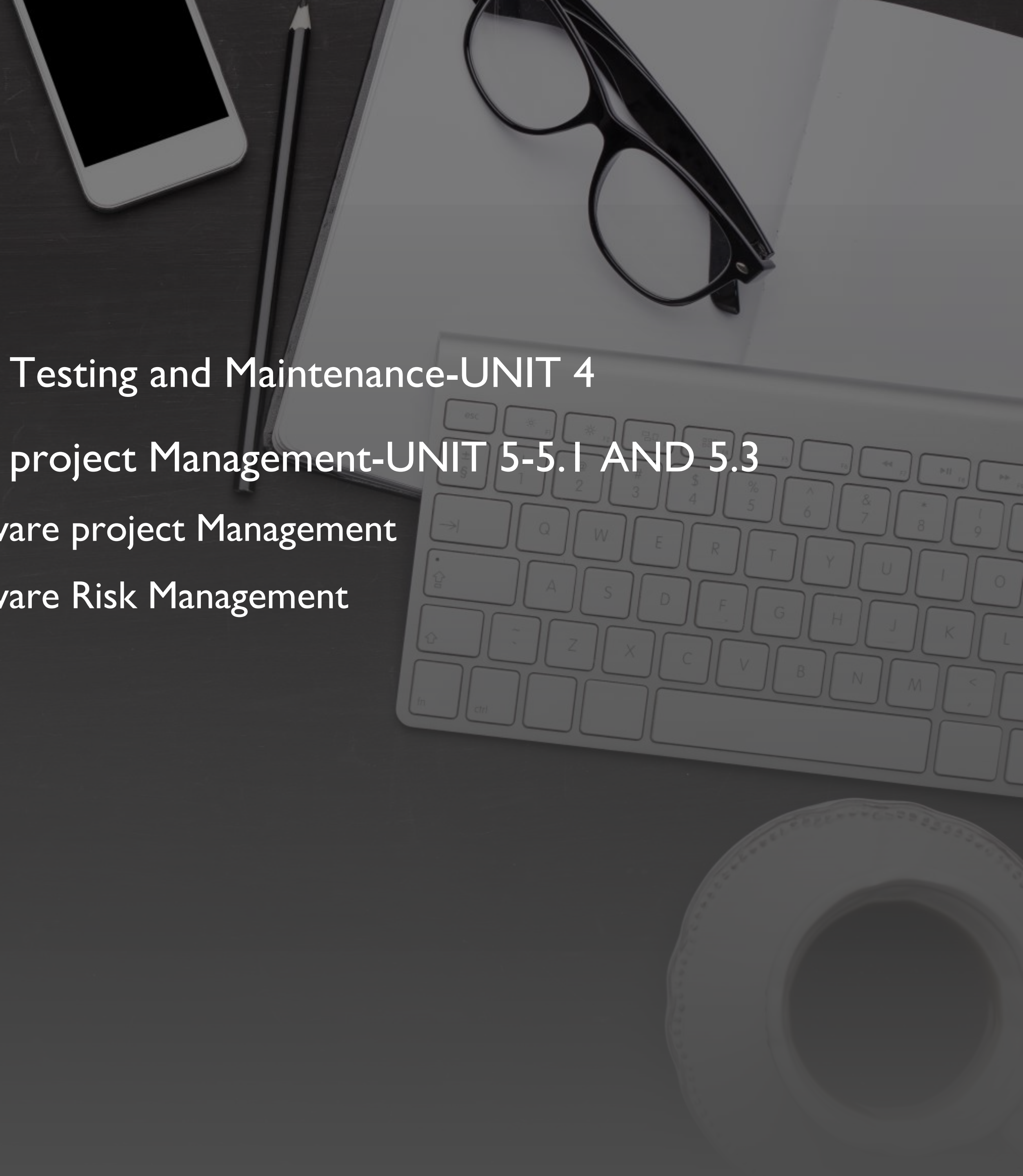
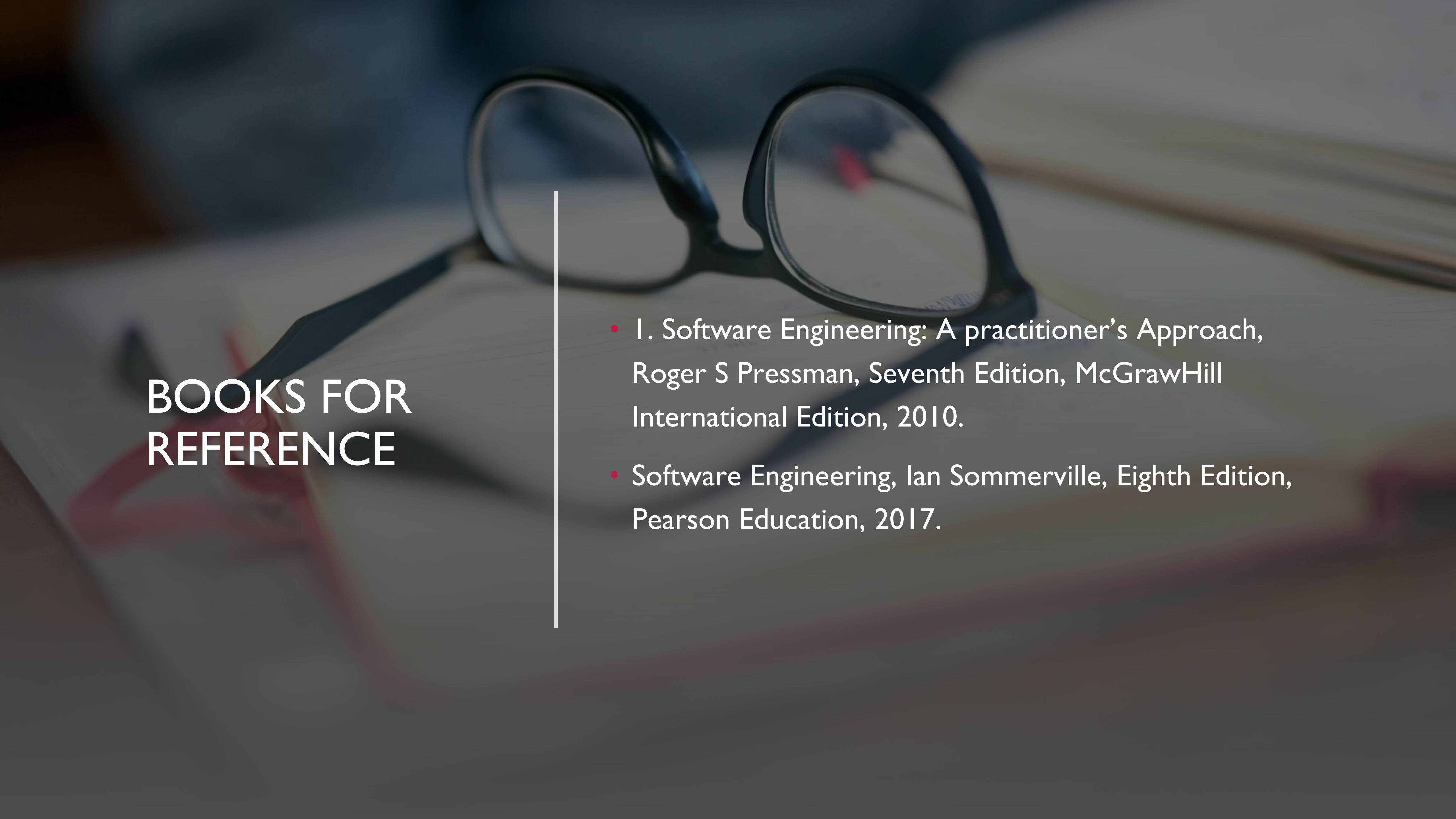
SOFTWARE TESTING



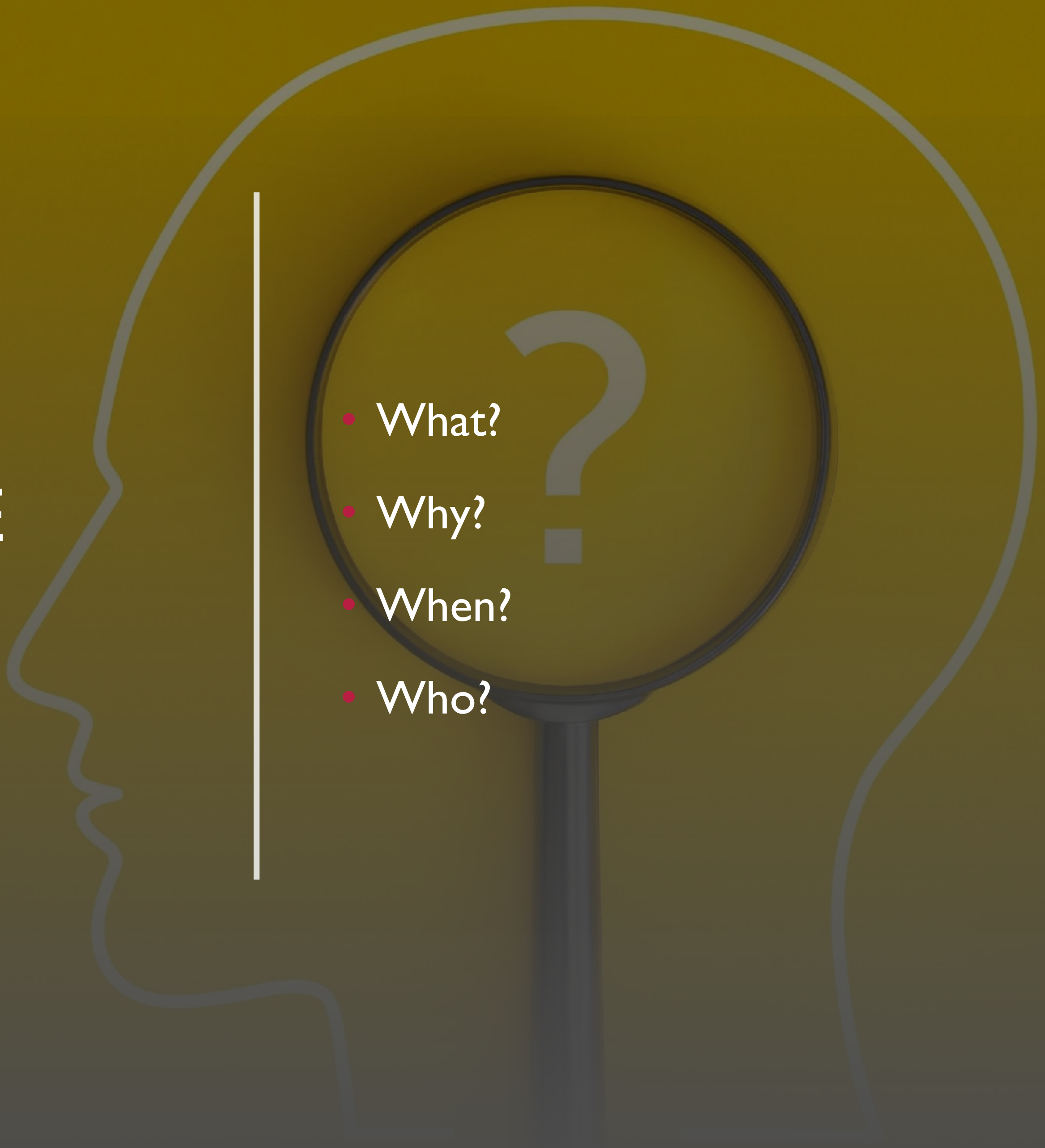
TOPICS

- Software Testing and Maintenance-UNIT 4
- Software project Management-UNIT 5-5.1 AND 5.3
  - Software project Management
  - Software Risk Management



BOOKS FOR REFERENCE

- 1. Software Engineering: A practitioner's Approach, Roger S Pressman, Seventh Edition, McGrawHill International Edition, 2010.
- Software Engineering, Ian Sommerville, Eighth Edition, Pearson Education, 2017.



SOFTWARE TESTING

- What?
- Why?
- When?
- Who?

What is Software Testing?

- To find out the errors in the software
- **How?**

Testing Strategy- But Why?

- Testing begins “in the small” and progresses “to the large.”
- Give the answer to the Who and when

Work Product?

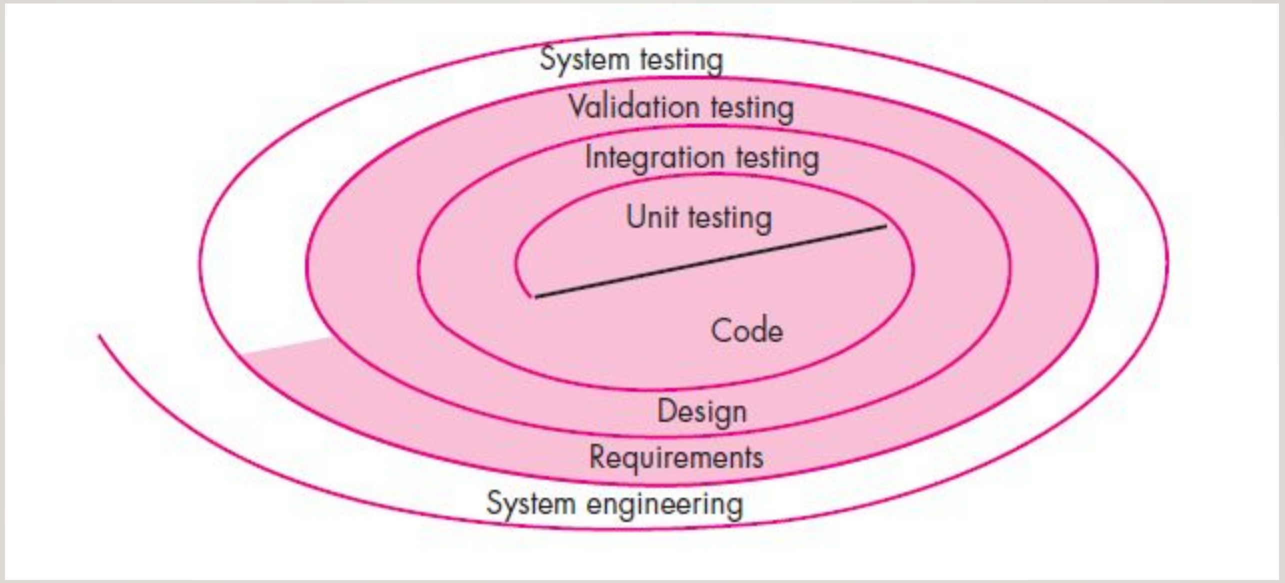
- Test Specification documents

VERIFICATION AND VALIDATION

- Testing is also referred as verification and validation
- **Verification:** “Are we building the product right?”
- **Validation:** “Are we building the right product?”

- Software developer
- Independent Testing Team

TESTING STRATEGY



FOUR STEPS FOR TESTING

Unit Testing- coverage of each path

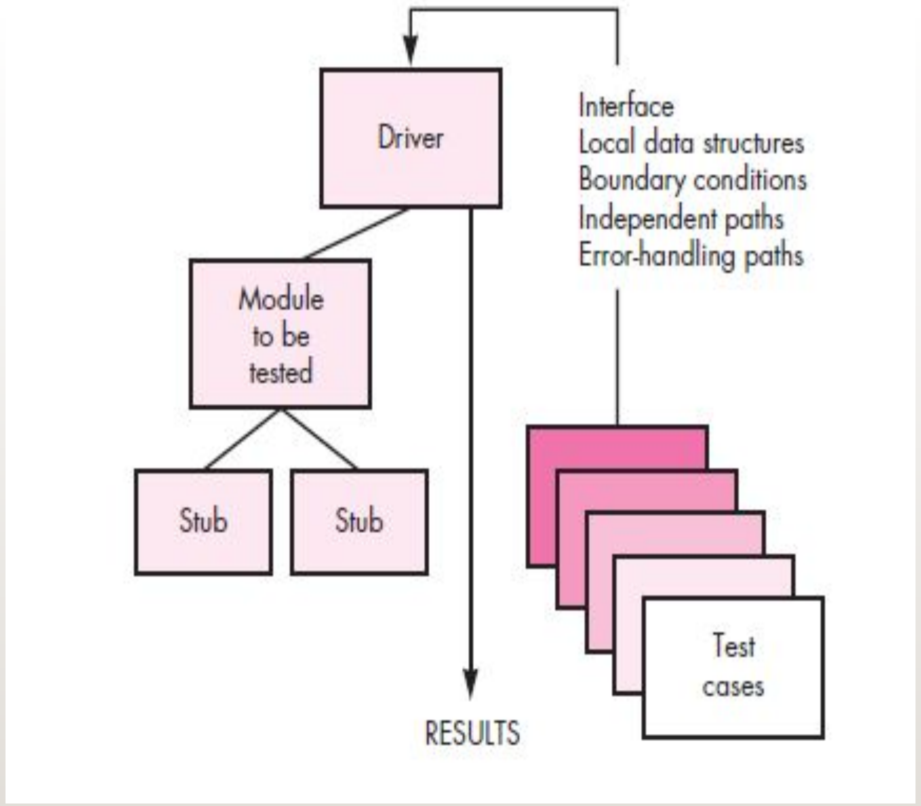
Integration Testing- Test cases for Input/output

Validation Testing-Requireme nts

System Testing- system performance

UNIT TESTING CATEGORIES

Interface	• test data flow across the component
Local Data Structures	• Analysis of Local data structure and impact on Global Variable
Boundary Conditions	• test the boundary Values
Independent Paths	• Identify due to erroneous computations, incorrect comparisons, or improper control flow.
Error-Handling Paths	• To test the error handling paths



UNIT TESTING PROCEDURE

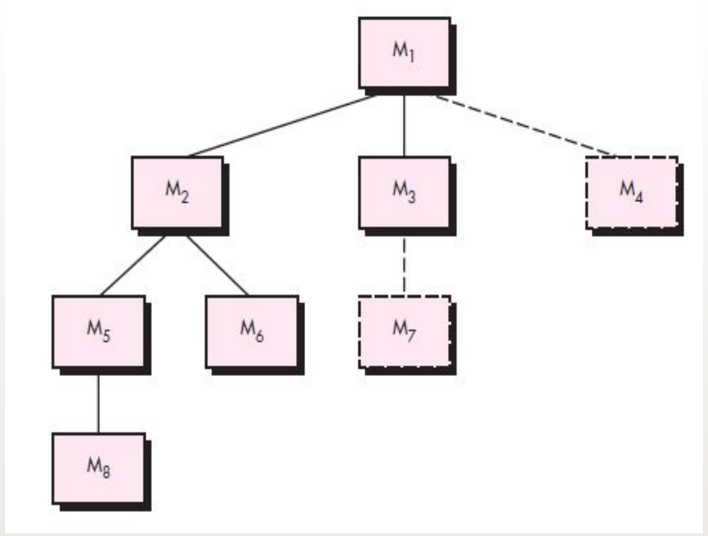
- Stub- Invoked by the component to be tested
- Drivers- “main program” that accepts test case data and invoked the component to be tested

INTEGRATION TESTING

- Need of Integration Testing?
  - -Interfacing
- systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing.
- How?
  - Big Bang Approach
  - Incremental approach-
    - Top Down Approach
    - Bottom Up Approach

TOP-DOWN APPROACH

- either a depth-first or breadth-first manner
  - Steps to follow
    - main control module is used as a test driver and stubs are substituted for all components directly subordinate
    - subordinate stubs are replaced one at a time with actual components
    - Test is conducted
    - On completion, another stub is replaced
    - Regression testing



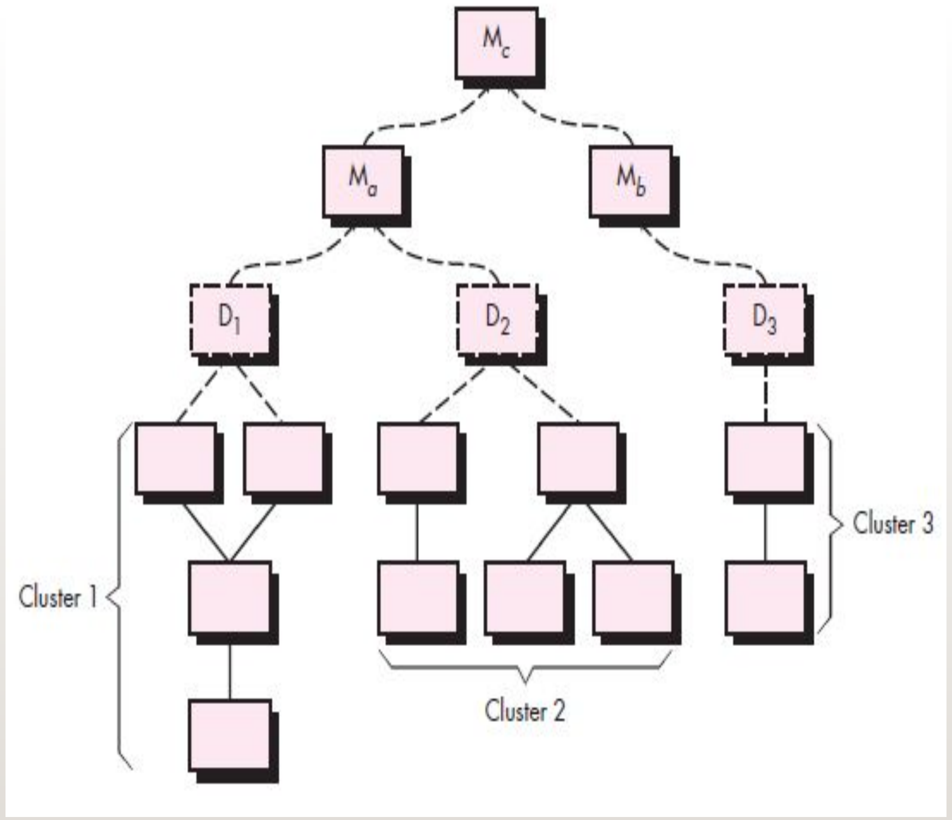
LIMITATION WITH TOP-DOWN APPROACH

Stubs are used in place of subordinates of main module

Solutions

- delay many tests until stubs are replaced with actual modules,
- develop stubs that perform limited functions
- integrate the software from the bottom of the hierarchy upward.

BOTTOM-UP INTEGRATION



Need for stubs is eliminated

Procedure

- Low-level components are combined into clusters
- A driver is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward

https://www.testingmind.com/what-is-the-difference-between-smoke-and-regression-testing/#:~:text=Smoke%20Testing%20Is%20Used%20To,Have%20Impacted%20The%20Existing%20Functionality.&text=It%20Is%20Employed%20By%20Both,Used%20By%20Software%20Testers%20Alone.
Reg vs smoke

REGRESSION TESTING

Re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

The Regression test suit includes these categories:

- A representative sample of tests that will exercise all software functions.
- Additional tests that focus on software functions that can be affected by the change.
- Tests that focus on the software components that have been changed.

SMOKE TESTING

A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.

tests is designed to expose errors that will keep the build from properly performing its function.

The build is integrated with other builds, and the entire product is smoke tested daily.

VALIDATION TESTING

The function or performance characteristic conforms to specification and is accepted deviation from specification is uncovered and a deficiency list is created.

Configuration Review

to ensure that all elements of the software configuration have been properly developed, are cataloged, and have the necessary detail.

Alpha Testing

The alpha test is conducted at the developer's site by a representative group of end users.

Beta Testing

The beta test is conducted at one or more end-user sites

Acceptance Testing

Customer performs a series of specific tests

SYSTEM TESTING

RECOVERY TESTING

SECURITY TESTING

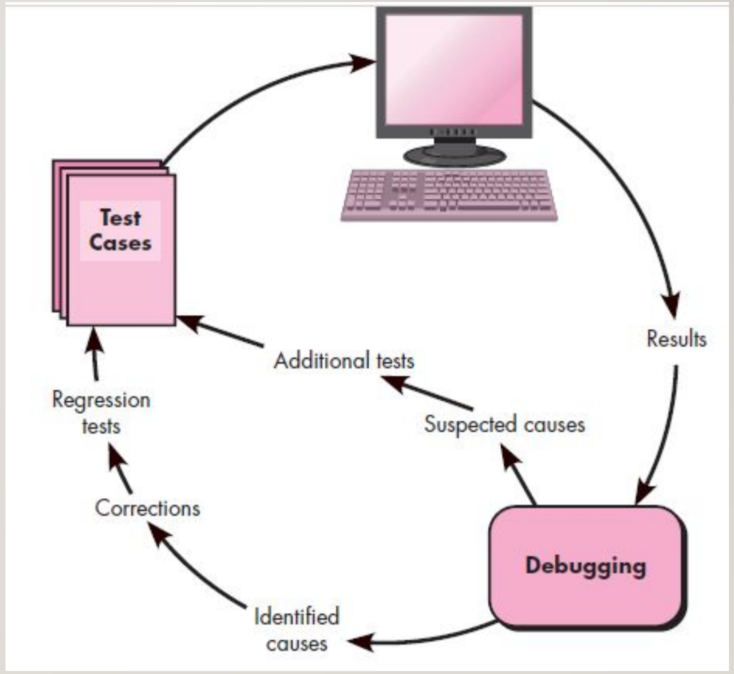
STRESS TESTING

PERFORMANCE TESTING

DEPLOYMENT TESTING

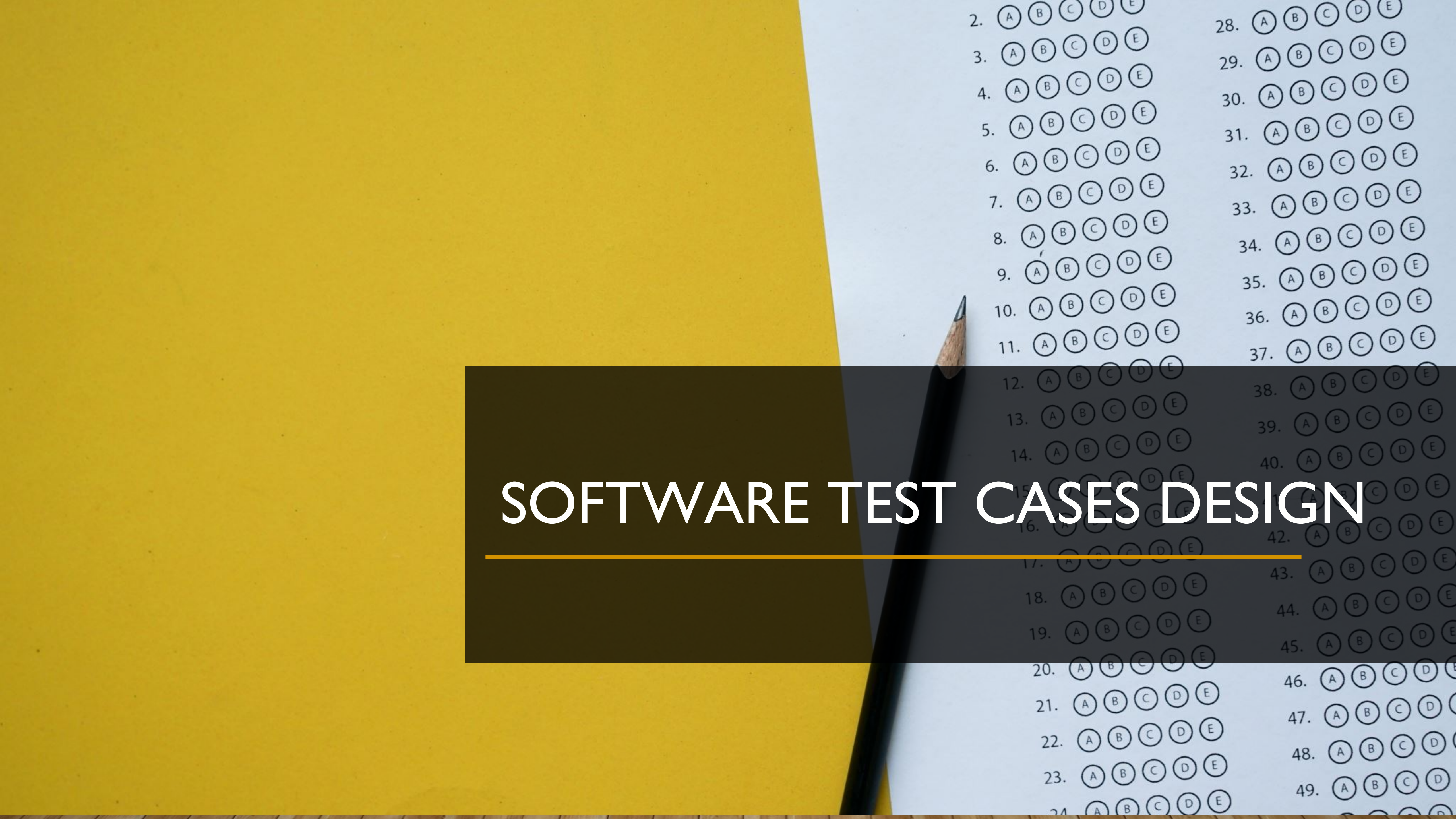
DEBUGGING PROCESS

- When a test case uncovers an error, debugging is the process that results in the removal of the error.
- The debugging process will usually have one of two outcomes:
  - the cause will be found and corrected
  - the cause will not be found.



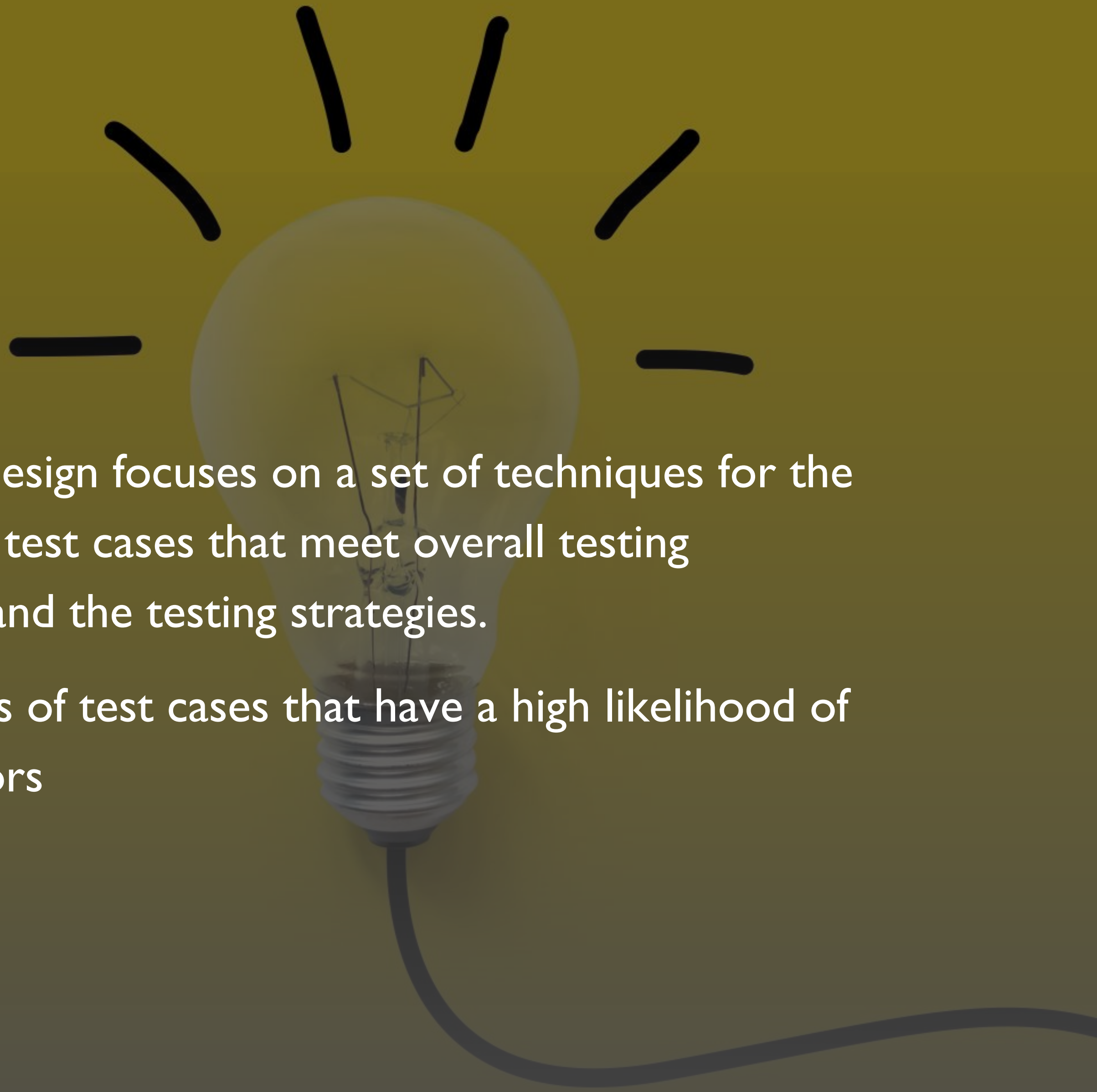
DEBUGGING STRATEGIES

Brute Force	manually searching through stack-traces, memory-dumps, log files, and so on, for traces of the error. Extra output statements, in addition to break points, are often
Backtracking	backtracks through the execution path, looking for the cause.
Cause Elimination	Hypotheses constructed as to why the bug has occurred. The code can either be directly examined for the bug, or data to test the hypothesis can be constructed.
Automated Debugging	Automated Debugging tool Available
Bisect	Previous versions of the software are examined until a version which does not have the error is located.



SOFTWARE TEST CASES DESIGN

SOFTWARE TEST CASES DESIGN



- Test-case design focuses on a set of techniques for the creation of test cases that meet overall testing objectives and the testing strategies.
- Goal- series of test cases that have a high likelihood of finding errors

TESTABILITY

- Operability. “The better it works, the more efficiently it can be tested.”
- Observability. “What you see is what you test.”
- Controllability. “The better we can control the software, the more the testing can be automated and optimized.”
- Decomposability. “quickly isolate problems and perform smarter retesting.”
- Simplicity. “The less there is to test, the more quickly we can test it.
  - functional
  - structural
  - code
- Stability. “The fewer the changes, the fewer the disruptions to testing’
- Understandability. “The more information we have, the smarter we will test.”

INTERNAL AND EXTERNAL VIEWS OF TESTING

- White box testing

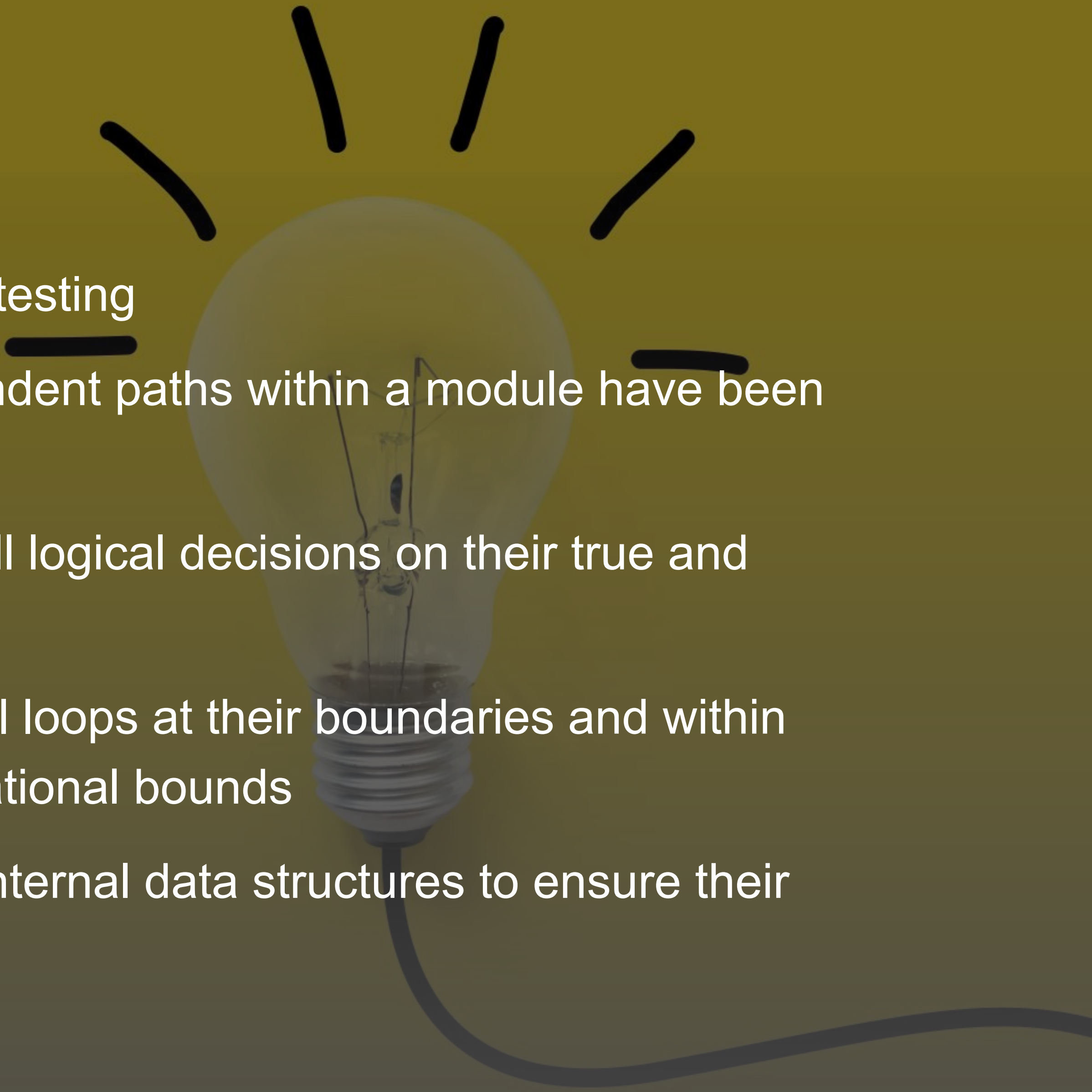
  examines some fundamental aspect of a system with little regard for the internal logical structure of the software.

- Black box testing

  Logical paths through the software and collaborations between components are tested

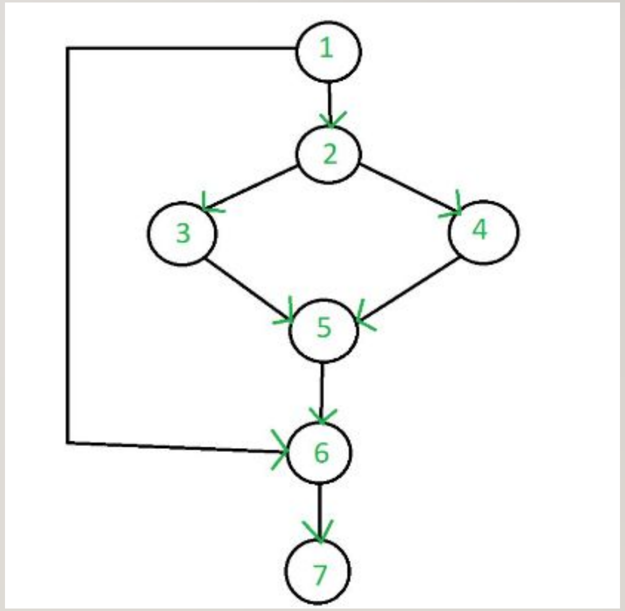
WHITE BOX TESTING

- glass-box testing
- all independent paths within a module have been exercised
- exercise all logical decisions on their true and false sides
- Execute all loops at their boundaries and within their operational bounds
- Exercise internal data structures to ensure their validity

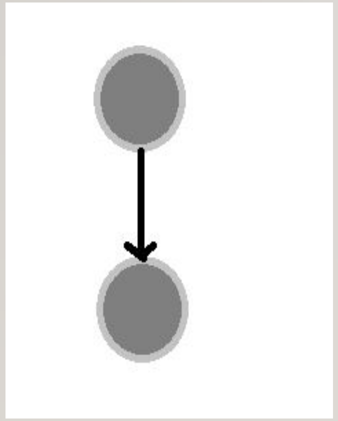
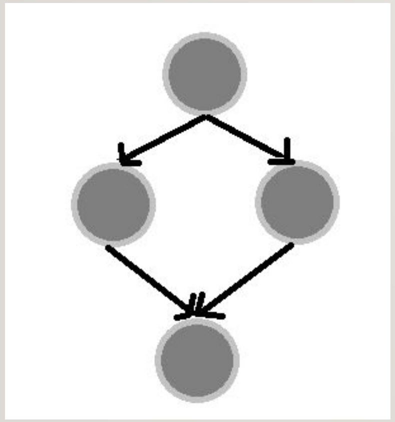
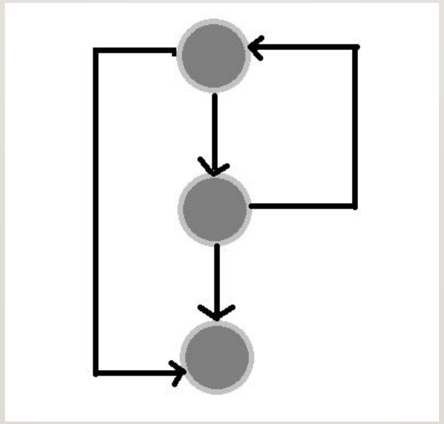
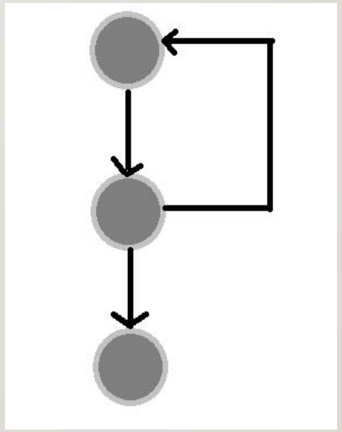
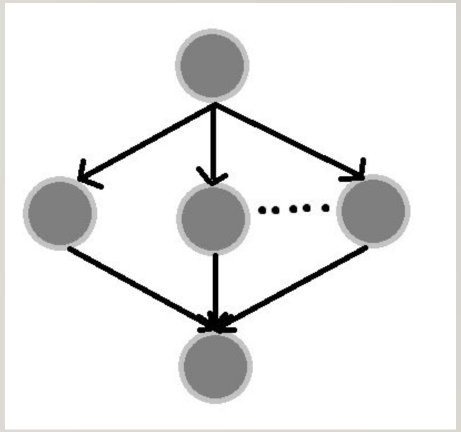


BASIS PATH TESTING

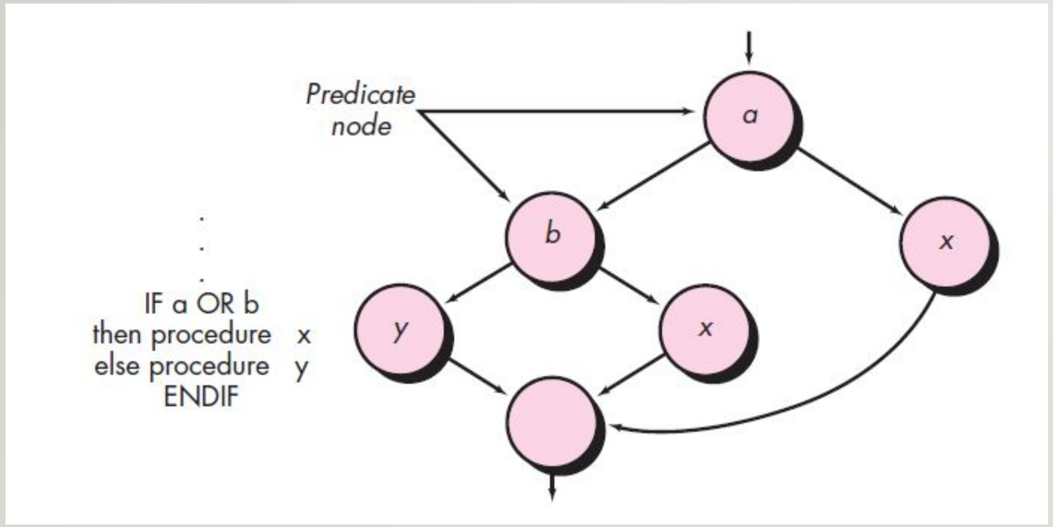
- enables the test-case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.
- Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.
- **Flow Graph Notation**



CONTROL STRUCTURES

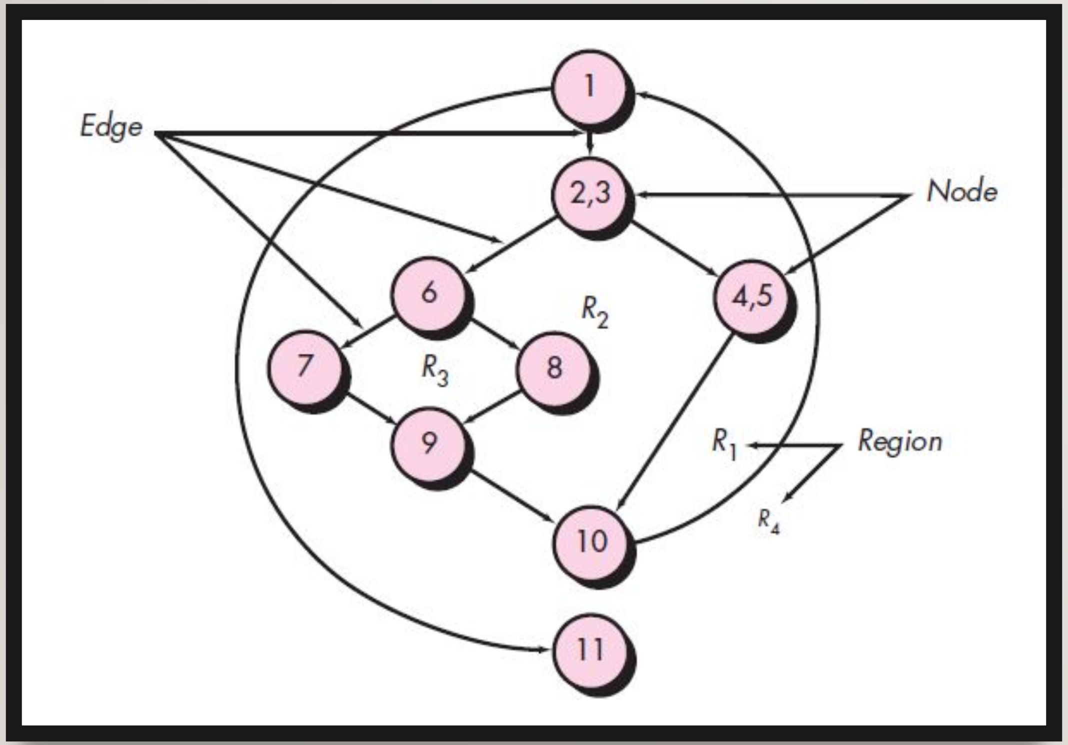






COMPOUND CONDITION STATEMENT



- Predicate Node
- Region

INDEPENDENT PROGRAM PATHS



- An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.

Path 1: 1-11

Path 2: 1-2-3-4-5-10-1-11

Path 3: 1-2-3-6-8-9-10-1-11

Path 4: 1-2-3-6-7-9-10-1-11

1-2-3-4-5-10-1-2-3-6-8-9-10-1-11
?????

How do you know how many paths to look for?

CYCLOMATIC COMPLEXITY

- software metric that provides a quantitative measure of the logical complexity of a program.
- provides an upper bound for the number of tests

WAYS TO CALCULATE THE CYLCOMATIC COMPLEXITY

- The number of regions of the flow graph corresponds to the cyclomatic complexity.
- Cyclomatic complexity $V(G)$ for a flow graph G is defined as

$$V(G) = E - N + 2$$

where E is the number of flow graph edges and N is the number of flow graph nodes.

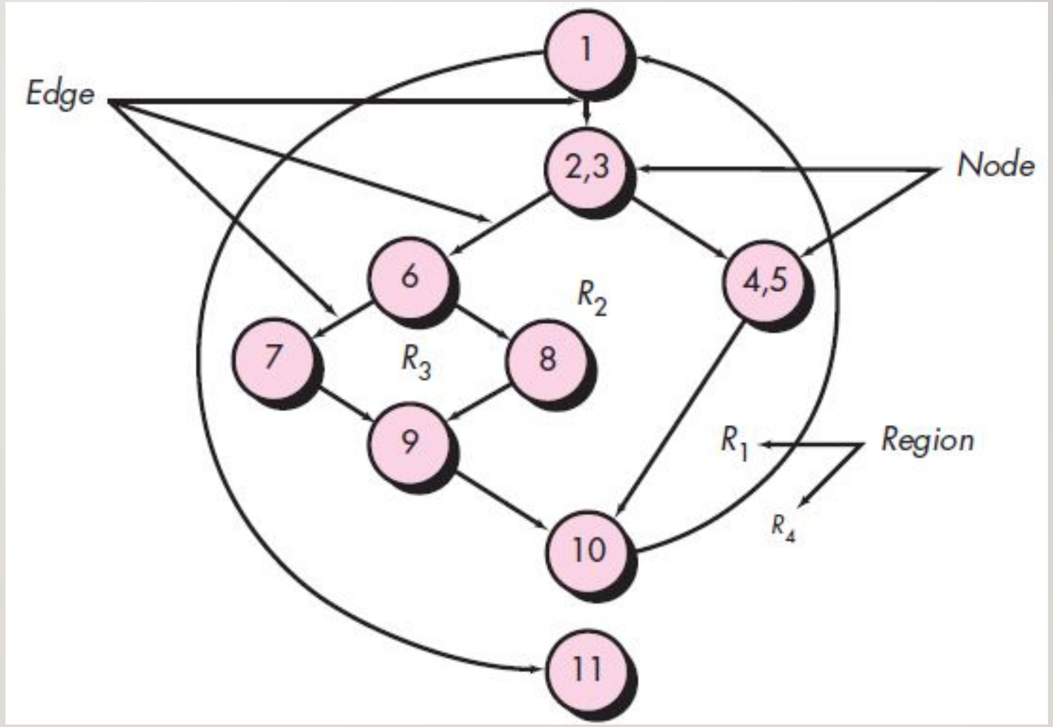
- Cyclomatic complexity $V(G)$ for a flow graph G is also defined as

$$V(G) = P + 1$$

where P is the number of predicate nodes contained in the flow graph G.

EXAMPLE

- The flow graph has four regions.
- V(G) = 11 edges - 9 nodes + 2 = 4.
- V(G) = 3 predicate nodes + 1 = 4.


1
Edge
2,3
Node
6
4,5
R_2
7
R_3
8
9
R_1
Region
R_4
10
11