

Testing Profession

The testing profession is not about just creating test related documents.

- It is understanding the application functionality and coding
- Testers are the ones who are the certifying authority for the software itself
- Testers destruct the software to give better solutions

Who performs testing?

- Programmers and testers

What should a good tester possess?

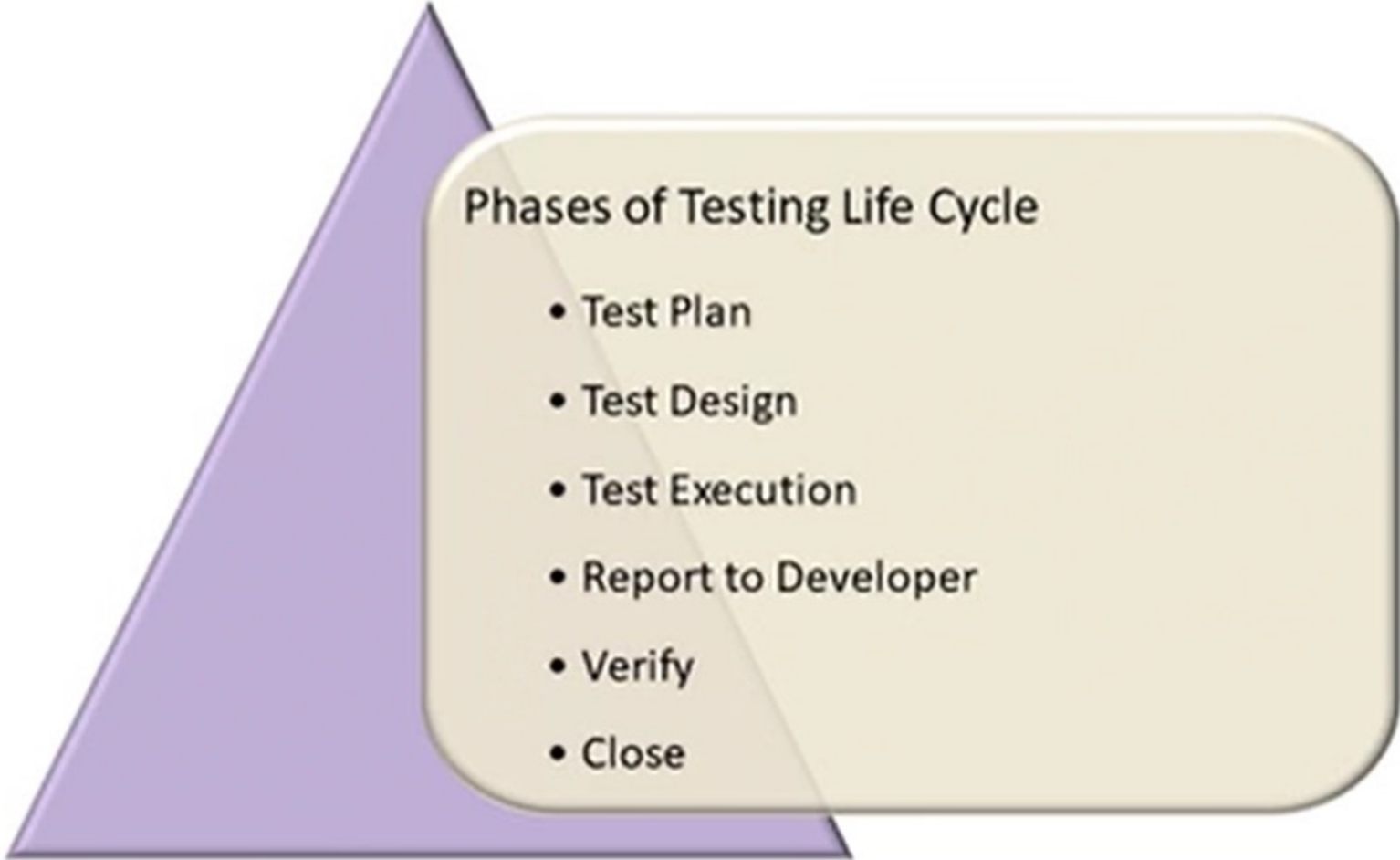
- Creative thinking
- Questioning skills
- Never give up attitude

Who will test a Software ?
What makes one a good Tester?



Testing Life Cycle

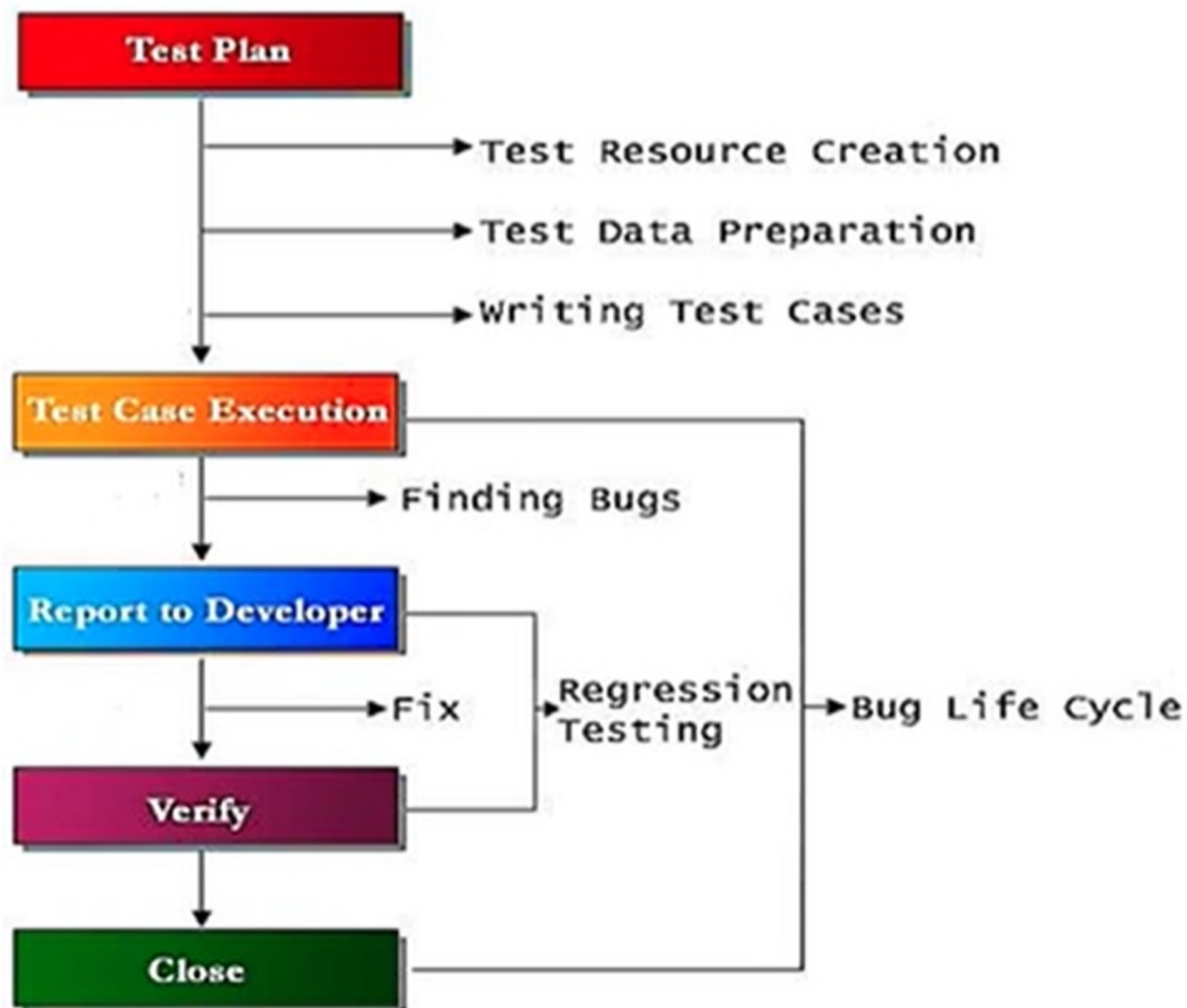
- Software testing life cycle identifies what test activities to carry out and when (what is the best time) to accomplish those test activities



Phases of Testing Life Cycle

- Test Plan
- Test Design
- Test Execution
- Report to Developer
- Verify
- Close

Testing Life Cycle



Methodology of Testing

Manual Testing

Automated Testing

- Example
 - Java – JUNIT
 - Dot Net – N UNIT

Bugzilla

- Defect Tracking Tool

What are the different methods of Testing?



Levels of Testing

Unit Testing

Integration
Testing

System
Testing

Acceptance
Testing

What are the different
Levels of Testing?



Unit Testing

Unit Testing is used to check whether a particular module is implementing its specification

The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.

Who performs testing?

- normally performed by software developers themselves or their peers
- In rare cases it may also be performed by independent software testers



Integration Testing

Integration Testing is a level of the software testing process where individual units are combined and tested as a group.

The purpose of Integration Testing is to expose faults in the interaction between integrated units.

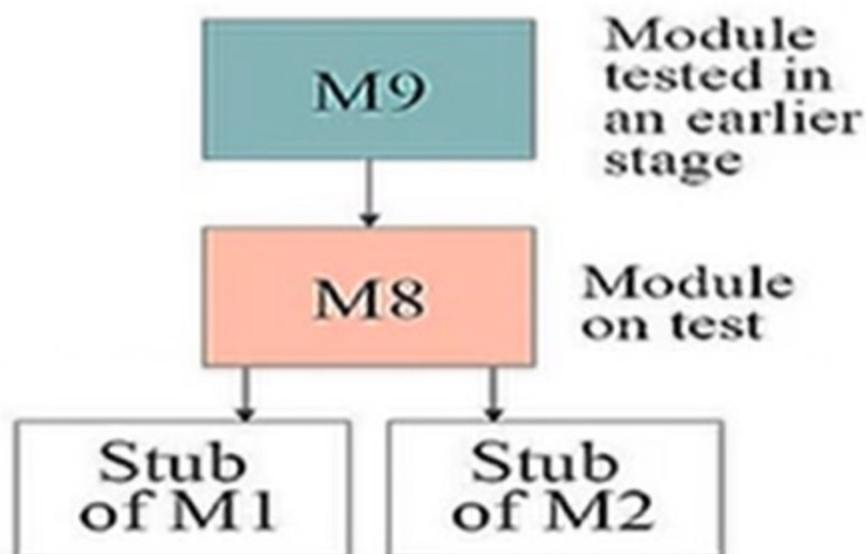
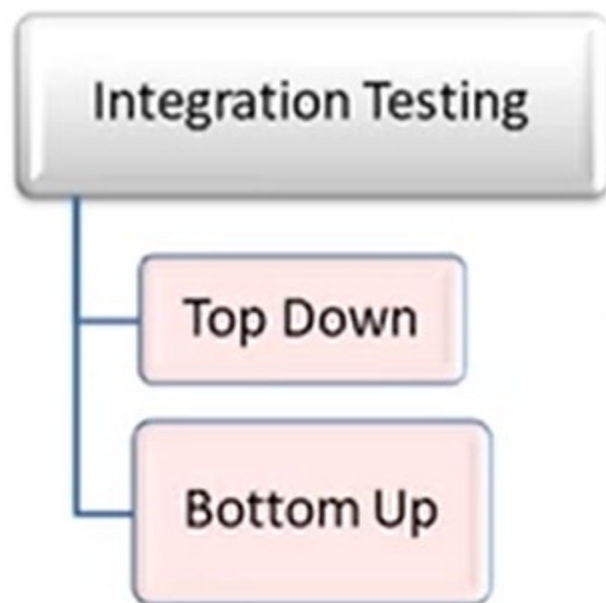
Who performs this testing?

- Either Developers themselves or independent Testers

Integration Testing Approaches

Top Down

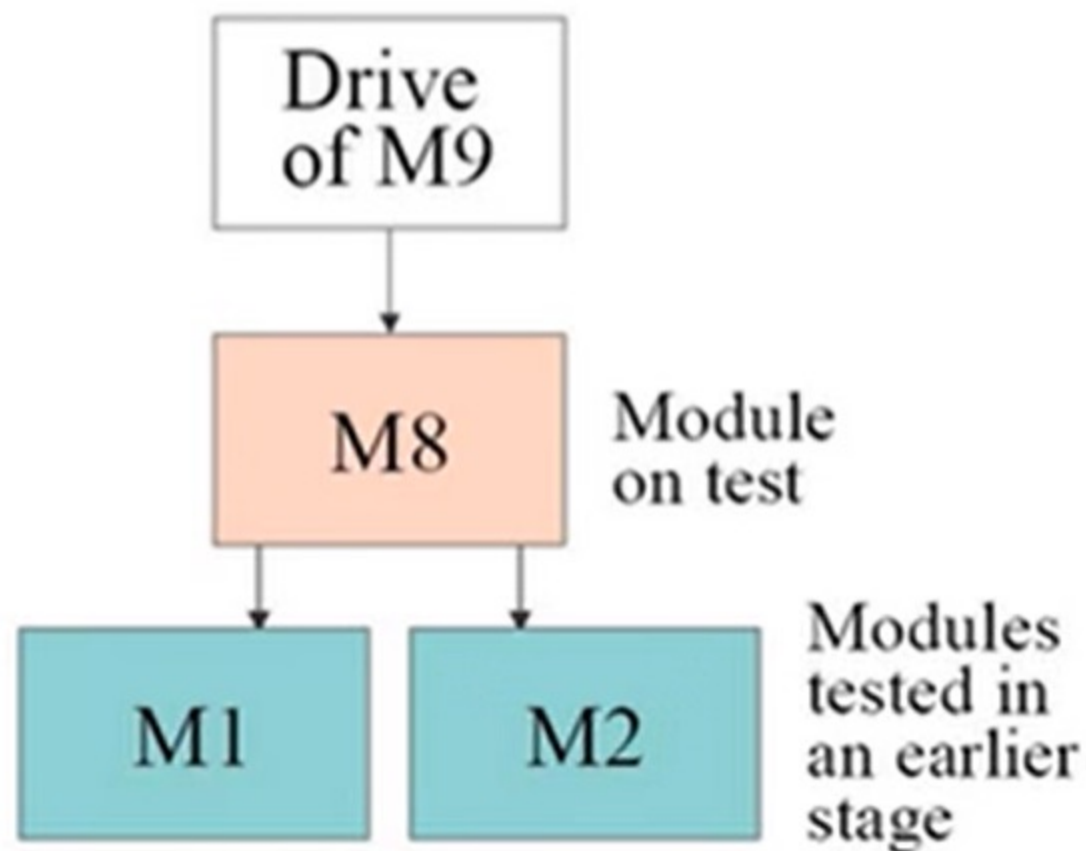
- Top level units are tested first and lower level units are tested step by step after that
- Test Stubs are needed to simulate lower level units which may not be available during the initial phases



Integration Testing Approaches

Bottom Up

- Bottom level units are tested first and upper level units step by step after that
- Test Drivers are needed to simulate higher level units which may not be available during the initial phases



System Testing

System testing finds disparities between implementation and specification

Performed by:
Testers

System testing involves

Functional testing - Test the implementation of the business needs

Performance testing - It will test all the non functional requirements of the system specified in the specification

Performance Testing - Types

Stress tests

- evaluates the system when stressed to its limits

Regression tests

- required when the system being tested is replacing an existing system

Usability test

- testing characteristics related to user friendliness

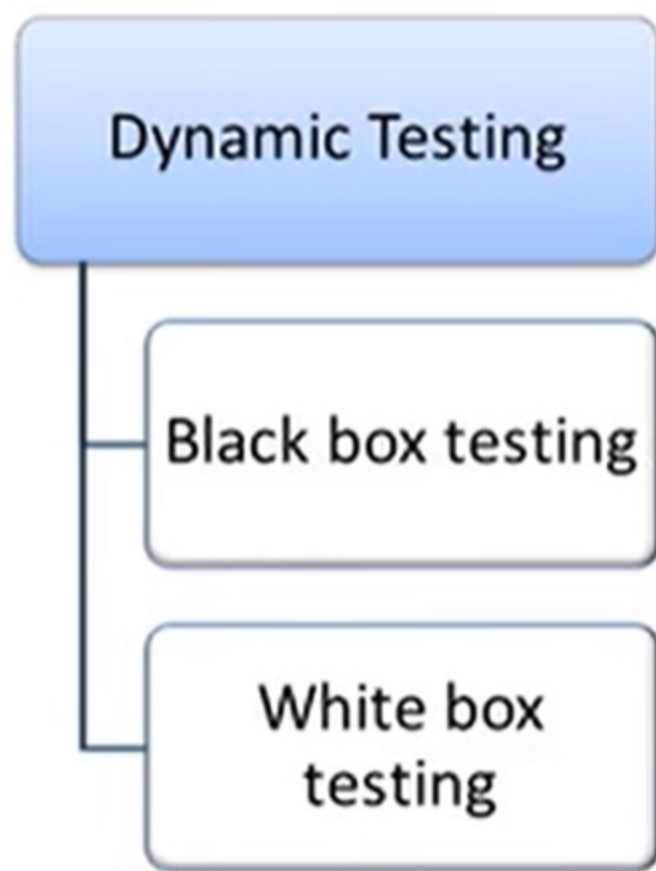
User Acceptance Testing

Done by the client to test whether the system meets the specified requirements

Types

- **Alpha testing** - Done by the client in developer's environment
- **Beta testing** - Done by the client in real world environment

Types of Testing




Static Testing

Static testing is the testing of the software work products manually to find errors


Work product means all the documents related to software as well as the code

Execution of the Code is not performed. Sanity of the code is checked.


Members of Static Testing

A light beige arrow-shaped box pointing to the right, containing the text 'Author'.

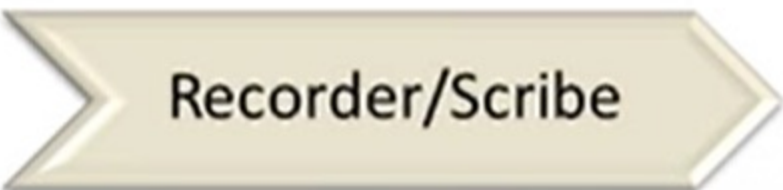
Author

A light beige arrow-shaped box pointing to the right, containing the text 'Moderator'.


Moderator

A light beige arrow-shaped box pointing to the right, containing the text 'Reader'.

Reader

A light beige arrow-shaped box pointing to the right, containing the text 'Recorder/Scribe'.

Recorder/Scribe

A light beige arrow-shaped box pointing to the right, containing the text 'Inspector'.

Inspector

Techniques in Static Testing

- REVIEW
 - It is a process in which the product is re-examined or re-evaluated for possible corrections
- WALK THROUGH
 - Mostly done on the code that is developed
 - The author gives sample test data. The test data examined with the code and intermediate results are recorded
- INSPECTION
 - Inspection involves step by step reading of the product, with each step checked against a predefined list of criteria(historical common errors, standards)

Review

- Review is a static testing technique.
- During review the artifacts are manually examined and the findings are recorded.
- Artifacts for which reviews are performed are
 - ❖ SRS
 - ❖ Design specification
 - ❖ Source code
 - ❖ Test plans
 - ❖ Test specification
 - ❖ Test cases
 - ❖ Test scripts
 - ❖ User guides
 - ❖ web pages

Review Advantages

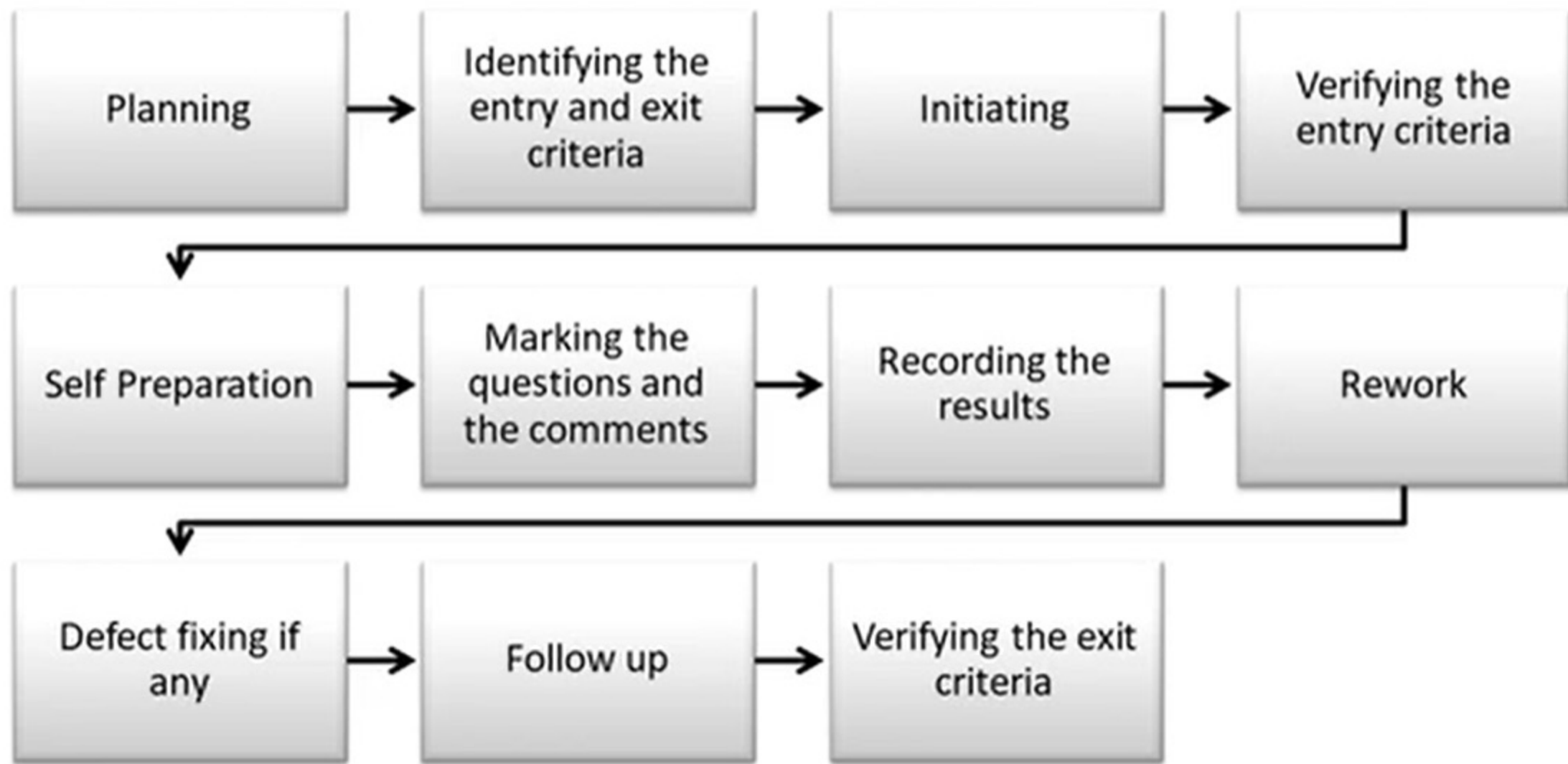
Early defect detection and correction

Fewer defects

Helps in identifying omissions

Saves testing cost and time

Review Process



Types of Review

Informal Review

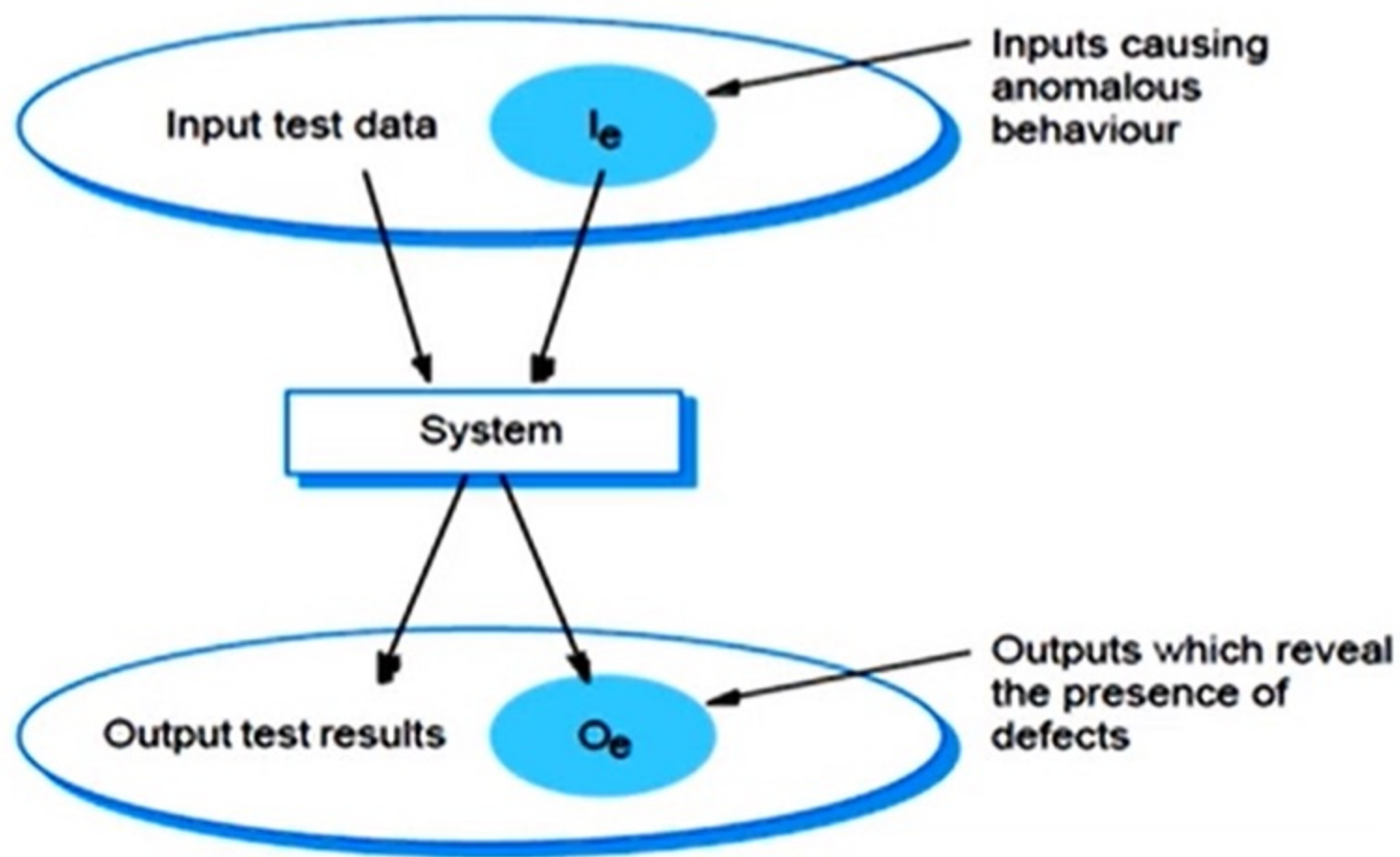
Walkthrough

Technical Review

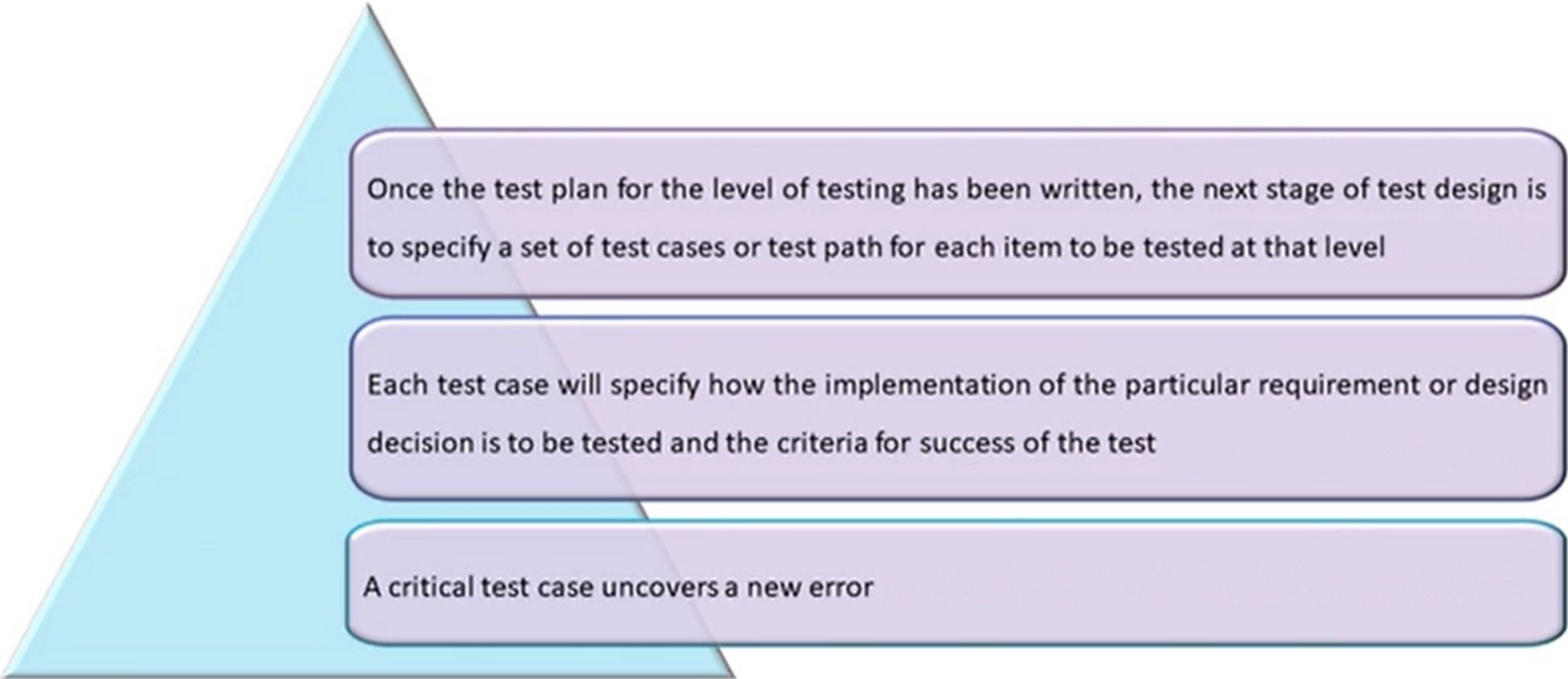
Inspection

Dynamic Testing – Black Box Testing

It is testing that ignores the internal mechanism of a system or a component and focuses solely on the outputs generated in response to selected inputs and execution conditions.



Test Case

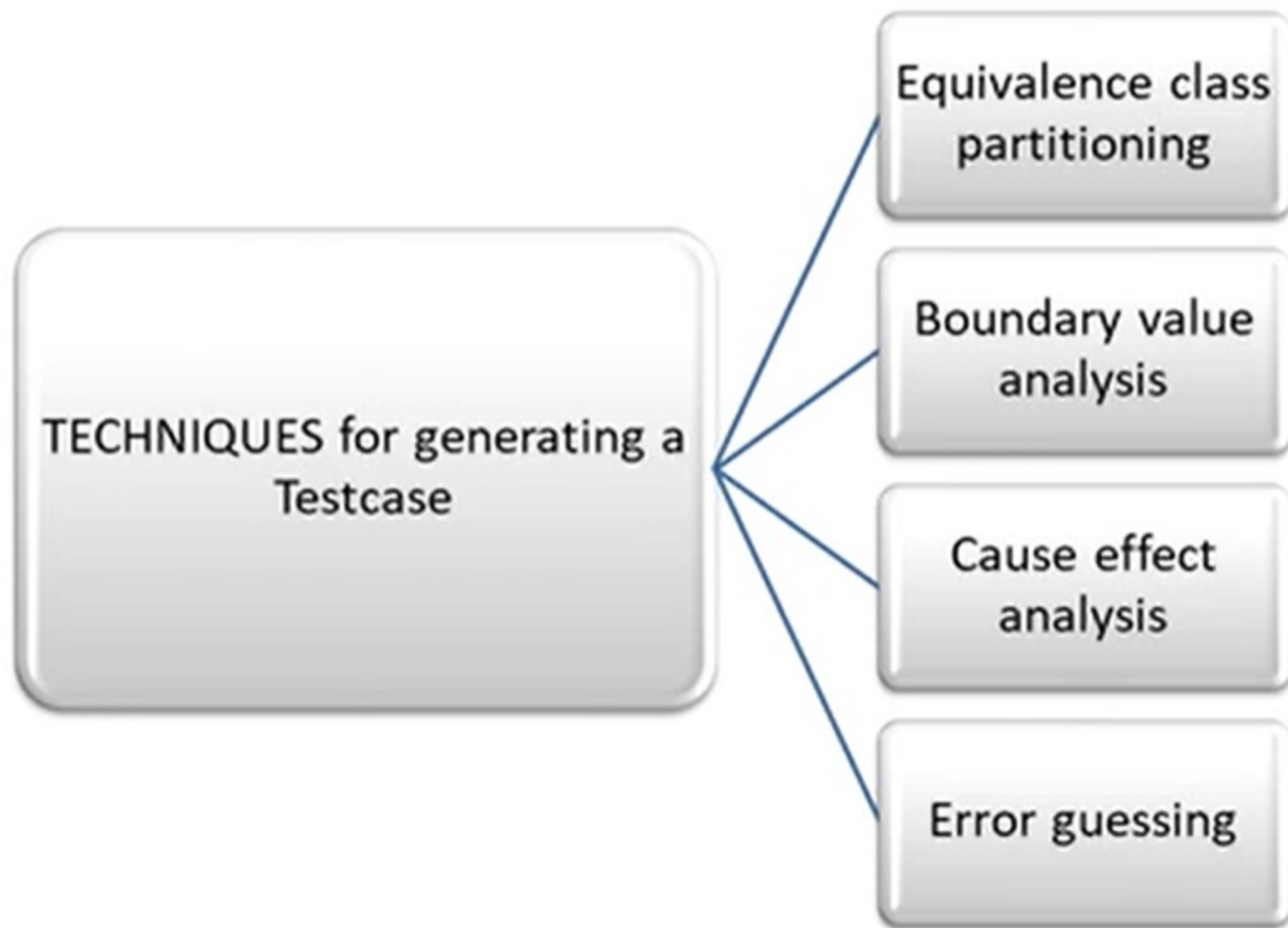


Once the test plan for the level of testing has been written, the next stage of test design is to specify a set of test cases or test path for each item to be tested at that level

Each test case will specify how the implementation of the particular requirement or design decision is to be tested and the criteria for success of the test

A critical test case uncovers a new error

Black Box Testing



Equivalence Class Partitioning

Black-box technique divides the input domain into classes of data from which test cases can be derived.

For each of these equivalence classes, the set of data should be treated the same by the module under test and should produce the same answer.

Equivalence class guidelines:

- For a range of input choose three values such that,
 - One value is above the range
 - One value is below the range
 - One value is within the range



Equivalence Class Partitioning

- EXAMPLE SCENERIO

- The component “generate_grading” is passed an exam mark (out of 75) and a coursework (c/w) mark (out of 25), from which it generates a grade for the course in the range 'A' to 'D'. The grade is calculated from the overall mark, which is calculated as the sum of the exam and c/w marks, as follows:

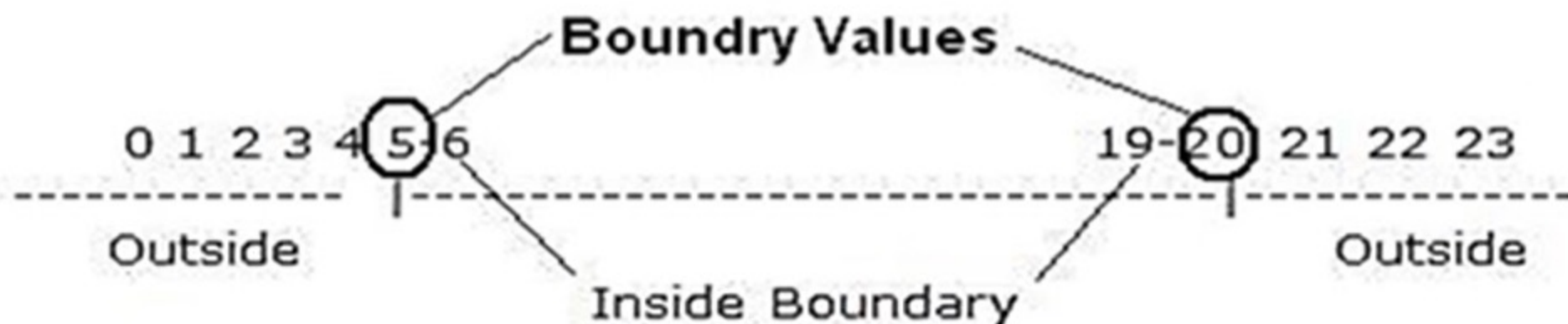
- greater than or equal to 70 - 'A'
- greater than or equal to 50, but less than 70 - 'B'
- greater than or equal to 30, but less than 50 - 'C'
- less than 30 - 'D'

- When a mark is outside its expected range, then a fault message ('FM') is generated. All inputs are passed as integers.




Boundary Value Analysis

- Black-box technique focuses on the boundaries of the input domain rather than its center
- Boundary value analysis guidelines:
 - If input condition specifies a range bounded by values a and b, test cases should include a and b, values just below a and just above b



Cause Effect Analysis

Three concentric semi-circles on the left side of the slide, colored in increasing shades of blue from the outermost to the innermost.

The cause effect analysis helps in identifying the causes and their effects (and identify how they are linked) associated with a particular problem or situation.

A cause is an input condition or an equivalence class of input conditions. An effect is an output condition or a system transformation.

It is suitable for applications in which combinations of input conditions are few.

Cause Effect Analysis

EXAMPLE:

- A module for calculating increment, that decides how much increment percent will be assigned to every employee. The decision based on the experience and the designation
 - For exp (valid input range is 1 to 50)
 - Between 1 – 3 then inc% 10
 - Between 4 – 10 then inc% 20
 - Greater than 10 then inc% 30
 - For designation
 - Developer then inc% 30
 - Tester then inc% 40

Error Guessing

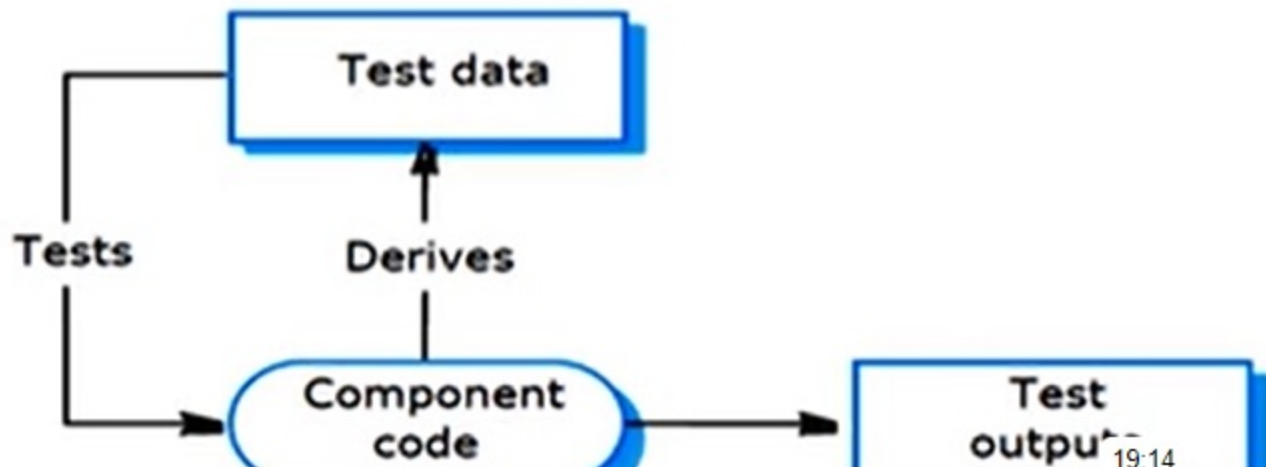
- It is an ad hoc approach guided by intuition and experience
- Example:
 - Consider a program is written into a file. The following test cases can be derived
 - Having an empty file
 - Not at all having a file
 - Having a file with read permission

Dynamic Testing – White Box Testing

It deals with the internal logic and structure of the code

Derivation of test cases according to the program structure

It is also called as glass box testing, structural testing, clear box testing



Basis Path Testing

- Basis Path Testing is white box testing method
- Design test cases to cover the following in the code written
 - ❖ every statement(statement coverage)
 - ❖ every predicate (condition) in the code(branch coverage)
 - ❖ loops (loop coverage)

Basis Path Testing

Derive a logical complexity measure of procedural design

- Break the module into blocks delimited by statements that affect the control flow (eg condition, method call, loop)
- Mark out these as nodes in a control flow graph
- Draw connectors (arcs) with arrow heads to mark the flow of logic
- Identify the number of regions (Cyclomatic Number) which is equivalent to the McCabe's number

McCabe's Number (Cyclomatic complexity)

- It defines the number of independent paths
- Provides the number of tests that must be conducted to ensure that all the statements are executed at least once.

Basis Path Testing

Complexity of a flow graph 'G', $v(G)$, is computed in one of three ways

- $V(G) = \text{No. of regions of } G + 1$
- $V(G) = E - N + 2$ (E: No. of edges & N: No. of nodes)
- $V(G) = P + 1$ (P: No. of predicate nodes in G or No. of conditions in the code)

Define a basis set of execution paths

Determine independent paths

Eliminate infeasible paths

Derive test case to exercise (cover) the basis set

Basis Path testing

```
• int main() {  
  - int a,b;  
  - scanf("%d%d",&a,&b);  
  - if(a > b)  
  -   printf("\na is greater");  
  - else  
  -   printf("\n b is greater");  
  - }
```

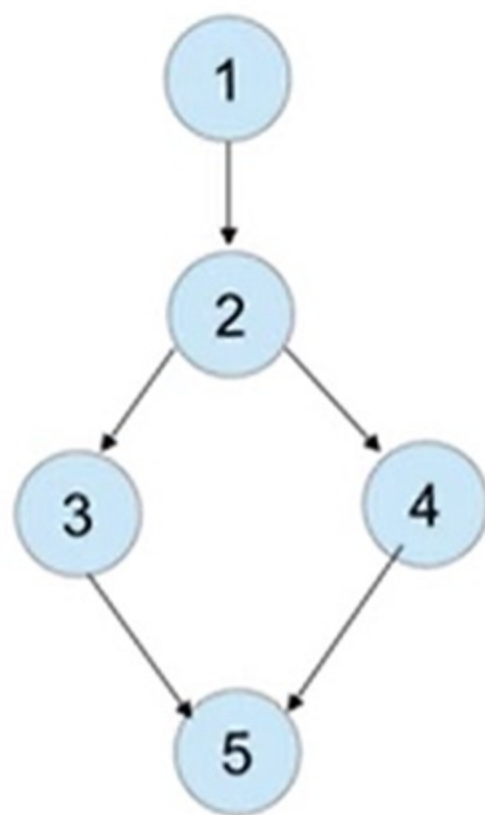
1

2

3

4

5



Mc cab's No

$P+1 = 2$

$E-N+2 = 5-5+2 = 2$

$R+1 = 2$

Individual Path

testcase

1-2-3-5

a=10 , b=5

1-2-4-5

a=5 , b=10

Basis Path Testing

```
• int main() {  
  - int n,i;  
  - scanf("%d",&n);  
  - for(i=1; i<=n; i++){  
  -   printf("\n%d",i);  
  - }  
• }
```

Diagram illustrating the code structure with nodes and edges:

- Node 1: Start of the main function.
- Node 2: Declaration of variables `int n,i;`.
- Node 3: Statement `scanf("%d",&n);`.
- Node 4: Statement `for(i=1; i<=n; i++){`.
- Node 5: Statement `printf("\n%d",i);`.
- Node 6: End of the function.

McCabe's no

$$P+1 = 2$$

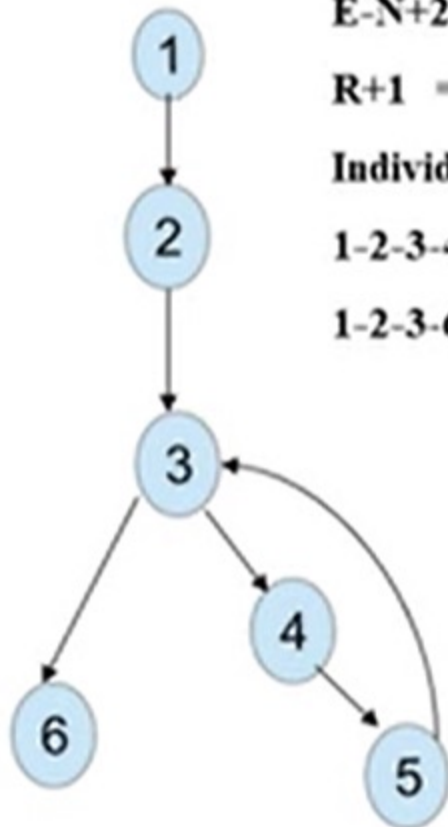
$$E-N+2 = 6-6+2 = 2$$

$$R+1 = 2$$

Individual Path

1-2-3-4-5-3-6 $n=2$

1-2-3-6 $n=0$



Black Box vs. White Box Testing

BLACK BOX TESTING

- Test cases are designed depending on the functionality
- Identifies hidden functionality
- Techniques
 - Equivalence class partitioning
 - Boundary value analysis
 - Cause effect analysis and graphing
 - Decision table

WHITE BOX TESTING

- Test cases are designed depending on the logic of the code
- Identifies unreachable code and checks for code coverage
- Techniques
 - Basis path testing

Static Testing Vs Dynamic Testing


STATIC TESTING

- manual examination (reviews) of the code or other project documentation
- Testing methodologies
 - Review
 - Inspection
 - Walkthrough
- Bugs discovered at this point are less expensive to fix

DYNAMIC TESTING

- the examination of the physical response from the system to variables that are not constant and change with time
- Testing methodologies
 - Black box testing
 - White box testing
- Depends on the type of bug identified.

When to Stop Testing?



Some of the common factors to stop Testing

- Testing budget of the project
- Resources available and their skills
- Project deadline and test completion deadline

Debugging

- Debugging is the process of finding the source of already identified bugs and fixing it.
- Performed by Developers.
- Steps in debugging
 - Find the defect in the code
 - Identify the root cause of the problem
 - Identify the exact place in the code that is the cause of the problem
 - Fix the defect
 - Recheck to ensure that the defect fixed is working as expected



Fundamentals of Testing

Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements, or to identify differences between expected and actual results.

Testing can be used to show the presence of defects, but never their absence!

