



WHITE BOX and BLACK BOX SW TESTING



Black Box and White Box

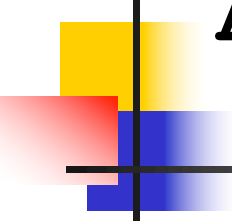
- WBT and BBT is not an alternative, rather it is *a complementary approach* that is likely to uncover a different class of errors than other methods



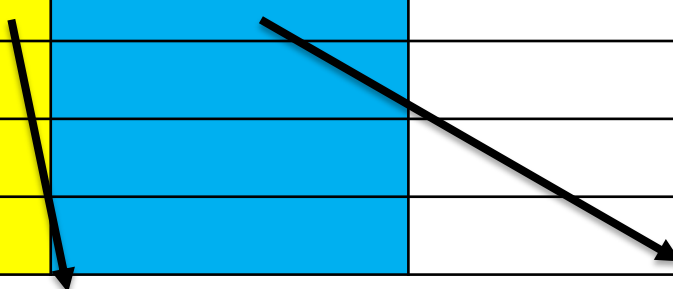
White Box vs Black Box

- White box testing (also called structural testing and glass box testing) is testing that takes into account the internal mechanism of a system or component.
- Black box testing is testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.

ANATOMY TEST CASE



Id.	Test Case	Expected Result	Actual Result	Note



IP atau V(g)

Diisi data untuk eksekusi alur yang dilewati



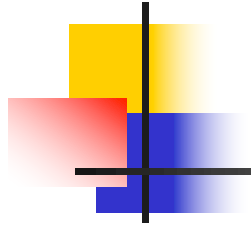
Contoh Poor Test Case

Test ID	Description	Expected Results	Actual Results
1	Player 1 rolls dice and moves.	Player 1 moves on board.	
2	Player 2 rolls dice and moves.	Player 2 moves on board.	

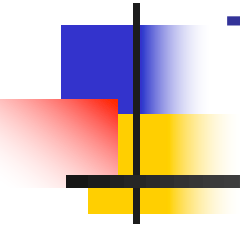


Contoh Good Test Case

Test ID	Description	Expected Results	Actual Results
3	Precondition: Game is in test mode, SimpleGameBoard is loaded, and game begins. Number of players: 2 Money for player 1: \$1200 Money for player 2: \$1200 Player 1 dice roll: 3	Player 1 is located at Blue 3.	
4	Precondition: Test case 3 has successfully completed Player 2 dice roll: 3	Player 1 is located on Blue 3. Player 2 is located on Blue 3.	



PERFORMING WHITE BOX TESTING

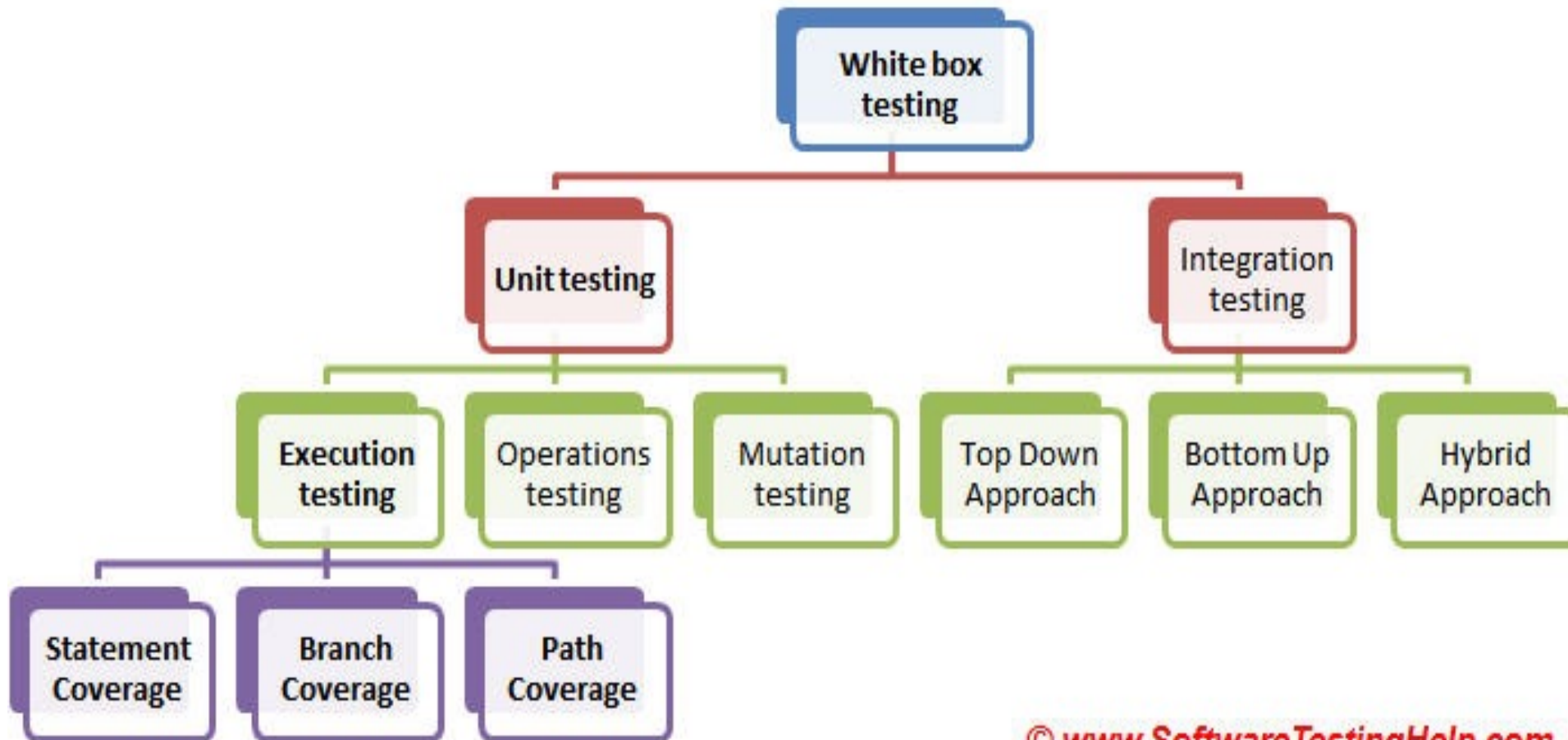




White-Box testing

- Sometime called *Glass-box testing, Clear Box Testing, Structural Testing*
- *A test case design method* that uses the control program structure to derive test case
- Knowledge of the code is used to identify additional test cases
- *Execution-based testing* that uses the code's inner structure and logical properties

Types of White Box Testing



© www.SoftwareTestingHelp.com

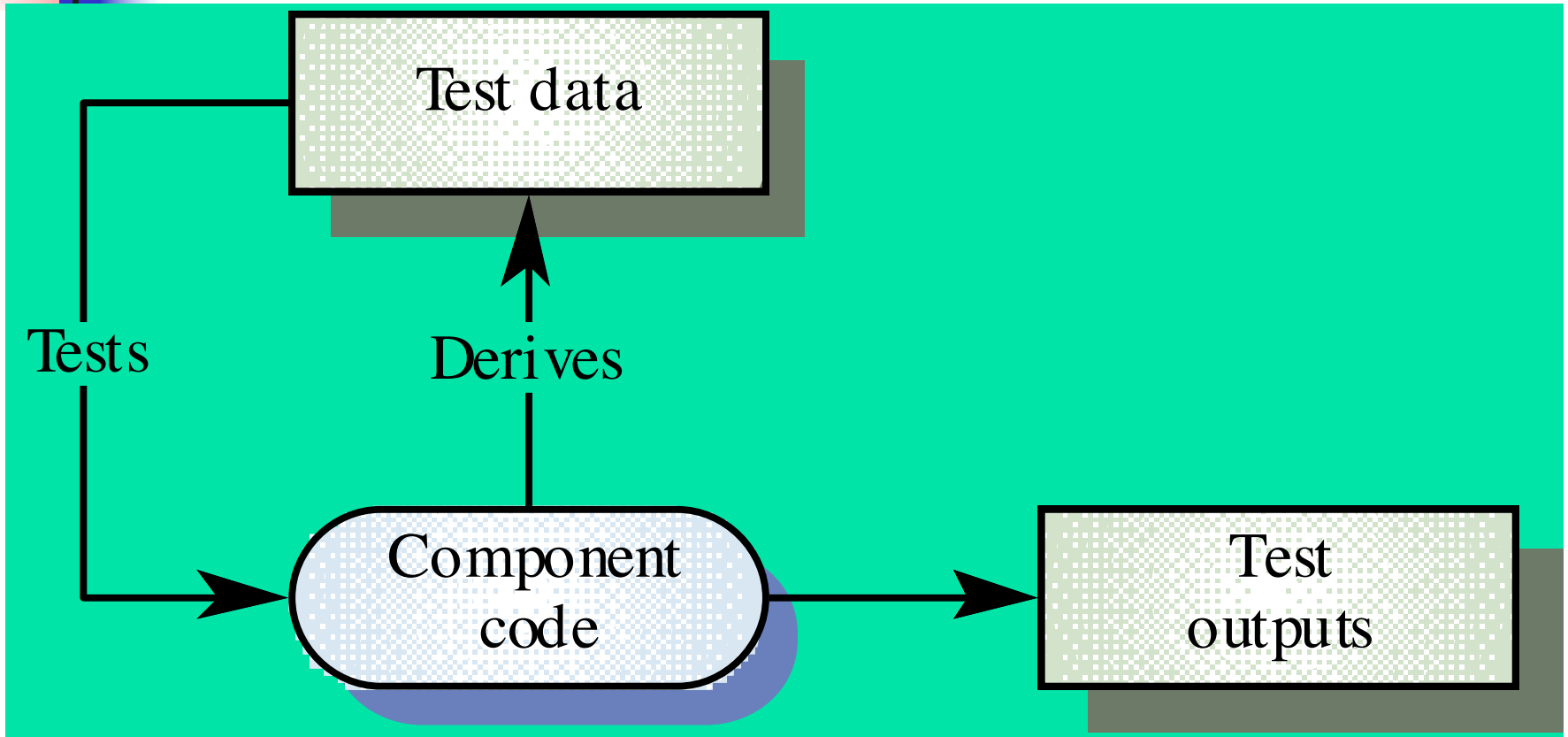


White-Box testing

Objective WBT is to exercise all program
Logic Coverage

- Every instruction / statement
- Every branch / decision

White-box testing





Types of Logic Coverage

- **Statement:** each statement executed at least once / **node coverage**
- **Branch:** each branch traversed (and every entry point taken) at least once / **edge coverage**
- **Condition:** each condition True at least once and False at least once



Types of Logic Coverage (cont'd)

- **Branch/Condition:** both Branch and Condition coverage achieved
- **Compound Condition:** all *combinations* of condition values at every branch statement covered (and every entry point taken)
- **Path:** all program paths traversed at least once



Execution Path

- An execution path is a set of nodes and directed edges in a control flow graph that connects (in a directed fashion) the start node to a terminal node.
- Two execution paths are said to be independent if they do not include the same set of nodes and edges.



Basis Set Execution Path

- A basis set of execution paths for a control flow graph is an independent maximum set of paths in which all nodes and edges of the graph are included at least once.



Basis Path Testing

- One of **white box testing technique**
- The objective of basis path testing is to derive a logical complexity measure of procedural design and defining a basis set of execution paths
- **Test case derived to exercise** the basis set to execute every statement in the program at least one time during testing



Steps Deriving Test Cases

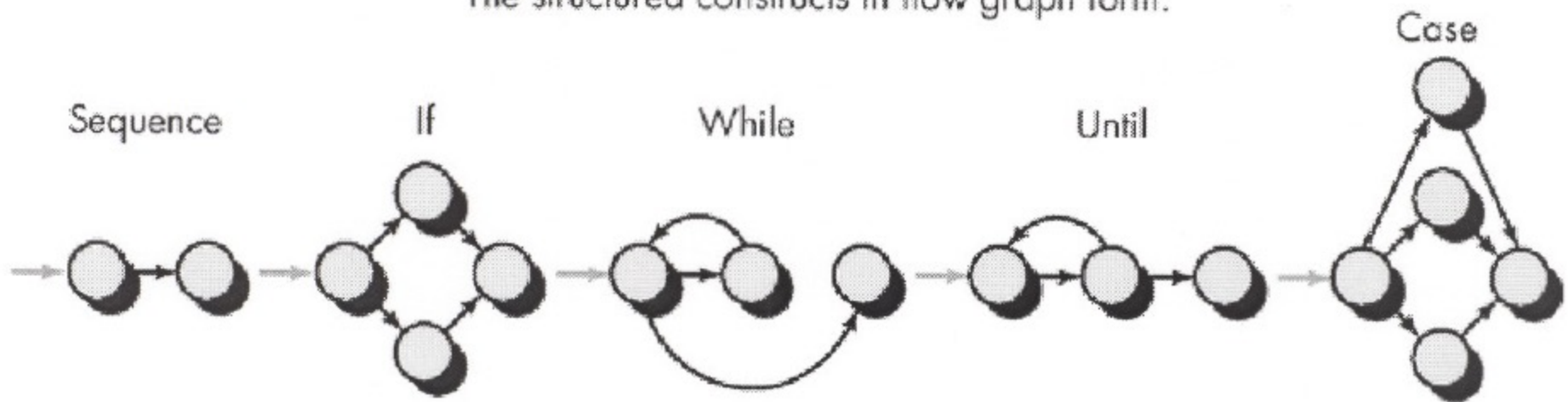
1. Map the design or code to the corresponding **control flow graph**
2. Compute the **cyclomatic complexity** of the resultant flow graph
3. Identify **a basis set of independent paths**
4. Prepare **test cases** that will force execution of each path in the basis set



Basis Path testing

- Uses **control flow graph** or **program graph** that shows nodes representing program decisions and vertex representing the flow of control
- Statements with conditions are therefore nodes in the flow graph
 - Node is any point in the program where the control either joins, or forks or both
 - Nodes are joined by edges or links

The structured constructs in flow graph form:



Where each circle represents one or more nonbranching PDL or source code statements

Flow graph notation

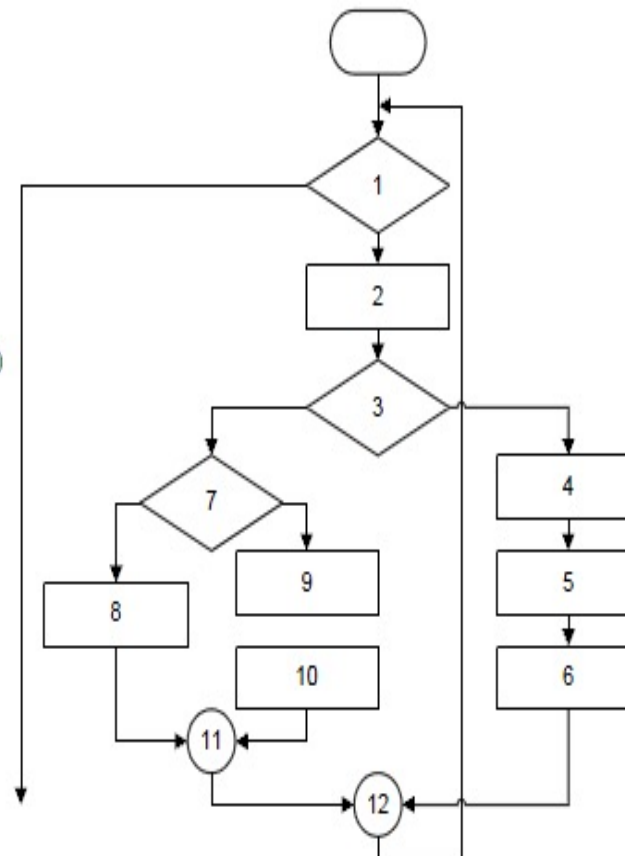
PROGRAM

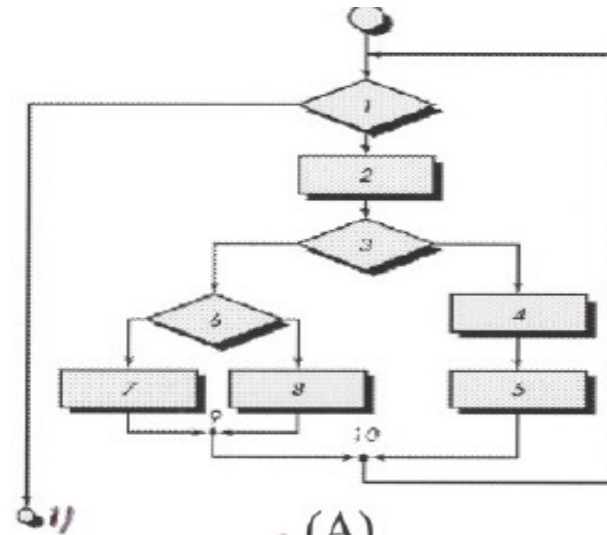
- Procedure Sort

```

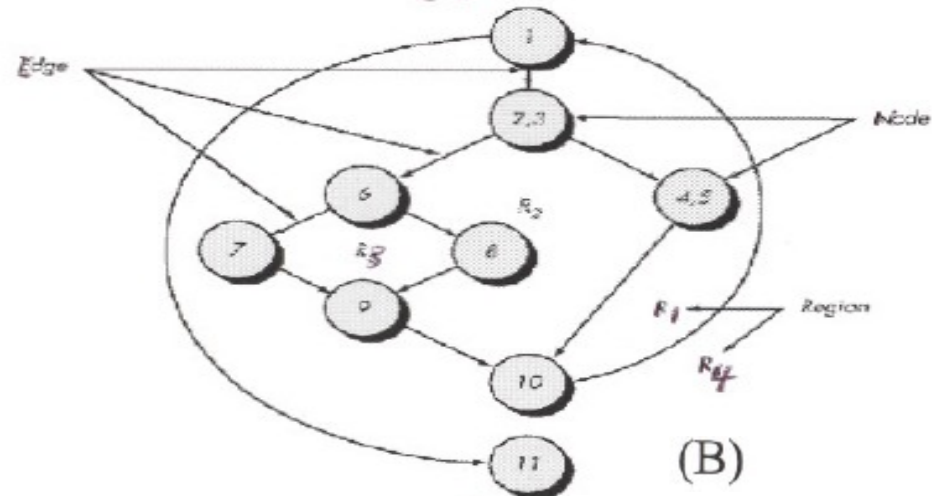
Procedure Sort
1. do while not eof
2.   Read Record
3.   if record field 1 = 0
4.     then process record
5.       store in buffer;
6.       increment counter
7.   else if record field 2 = 0
8.     then reset counter
9.     else process record
10.      store in file
11.   endif
12. endif
13. enddo
    
```

- Flow Chart





(A)



(B)

Flowchart, (A) and flow graph (B)



Cyclomatic Complexity : $V(G)$

- Metric that provides a quantitative measure of logical complexity of a program
- Define the number of independent path in the basis set of a program
- The number of tests cases used to test all control statements equals the cyclomatic complexity



Compute Cyclomatic Complexity

1. $V(G) = R$
2. $V(G) = P + 1$
3. $V(G) = E - N + 2$

Where :

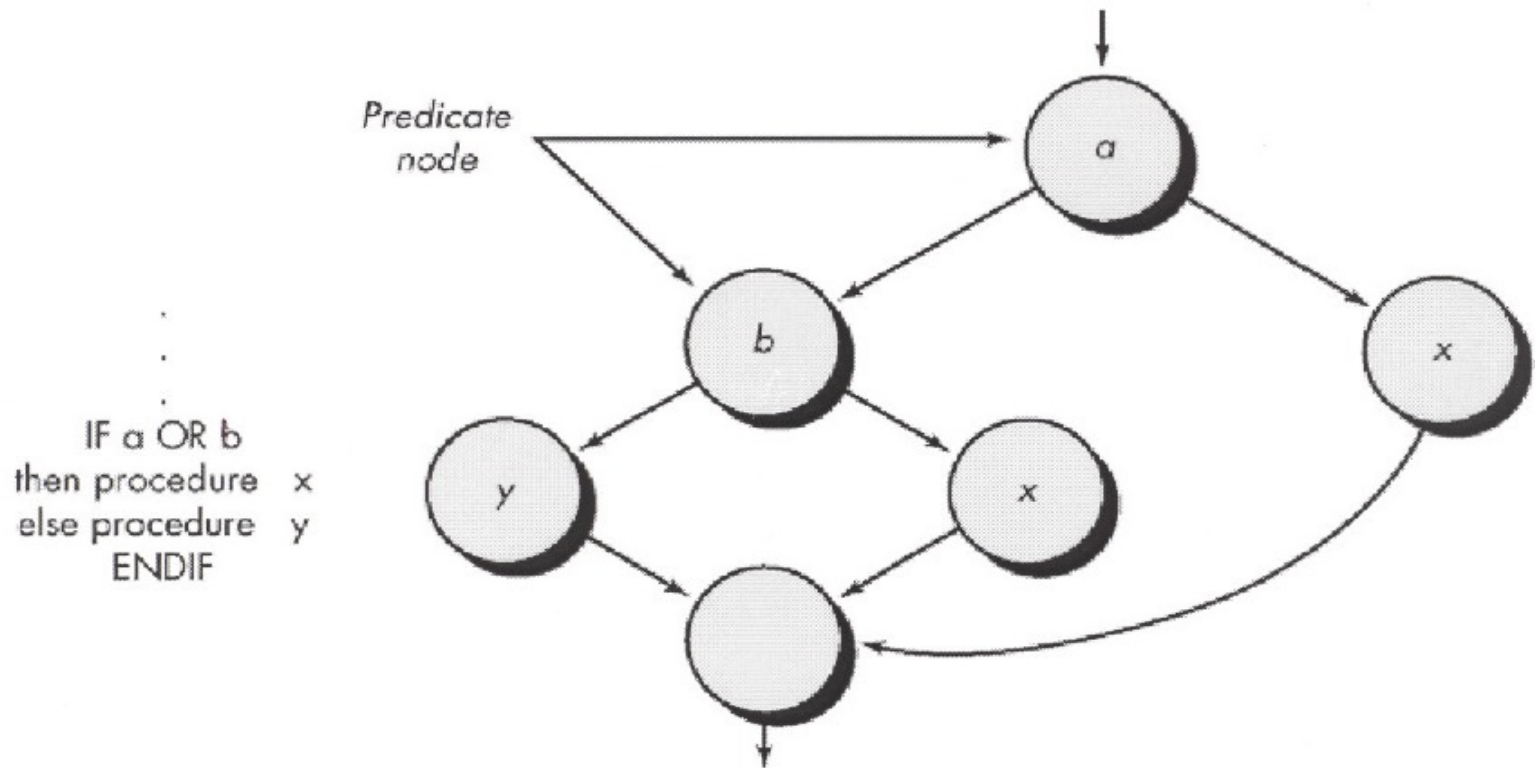
- R is the number of regions of the flow graph
- P is the number of logical expressions (or predicates node)
- E is the number of flow graph edges
- N is the number of flow graph nodes



Independent Path (IP)

Independent Path is any path through the program that introduces at least one **new set processing statement** or a **new condition**

Compound predicate Node

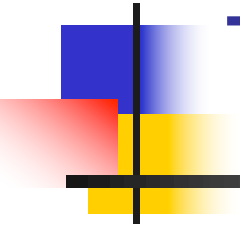




Main Limitations of Basis path testing

- Can not be used to show totally wrong or missing functions
- Interface errors may not be caught
- Database errors may not be caught
- Not all initialization errors are caught by path testing

PERFORMING BLACK BOX TESTING





Black-box testing

- An approach to testing where the program is considered as a 'black-box'
- Also called *Functional Testing* or *Behavioral Testing*
- Focus on determining whether or not a program does what it is supposed to do based on its functional requirements



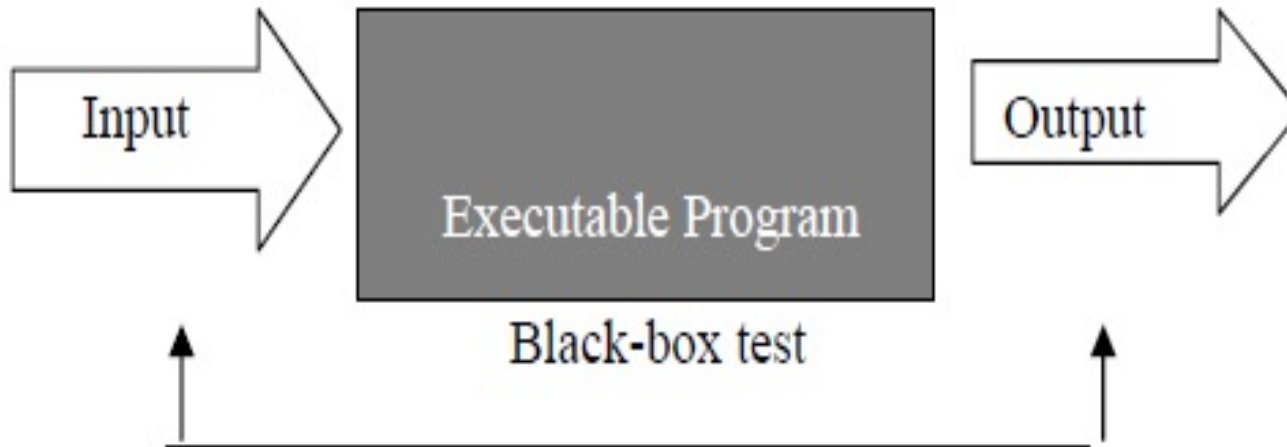
Objective Black Box Testing

Black box testing tends to find different kinds of errors :

- (1) incorrect or missing functionality;
- (2) interface errors;
- (3) errors in data structures used by interfaces;
- (4) behavior or performance errors;
- (5) initialization and termination errors

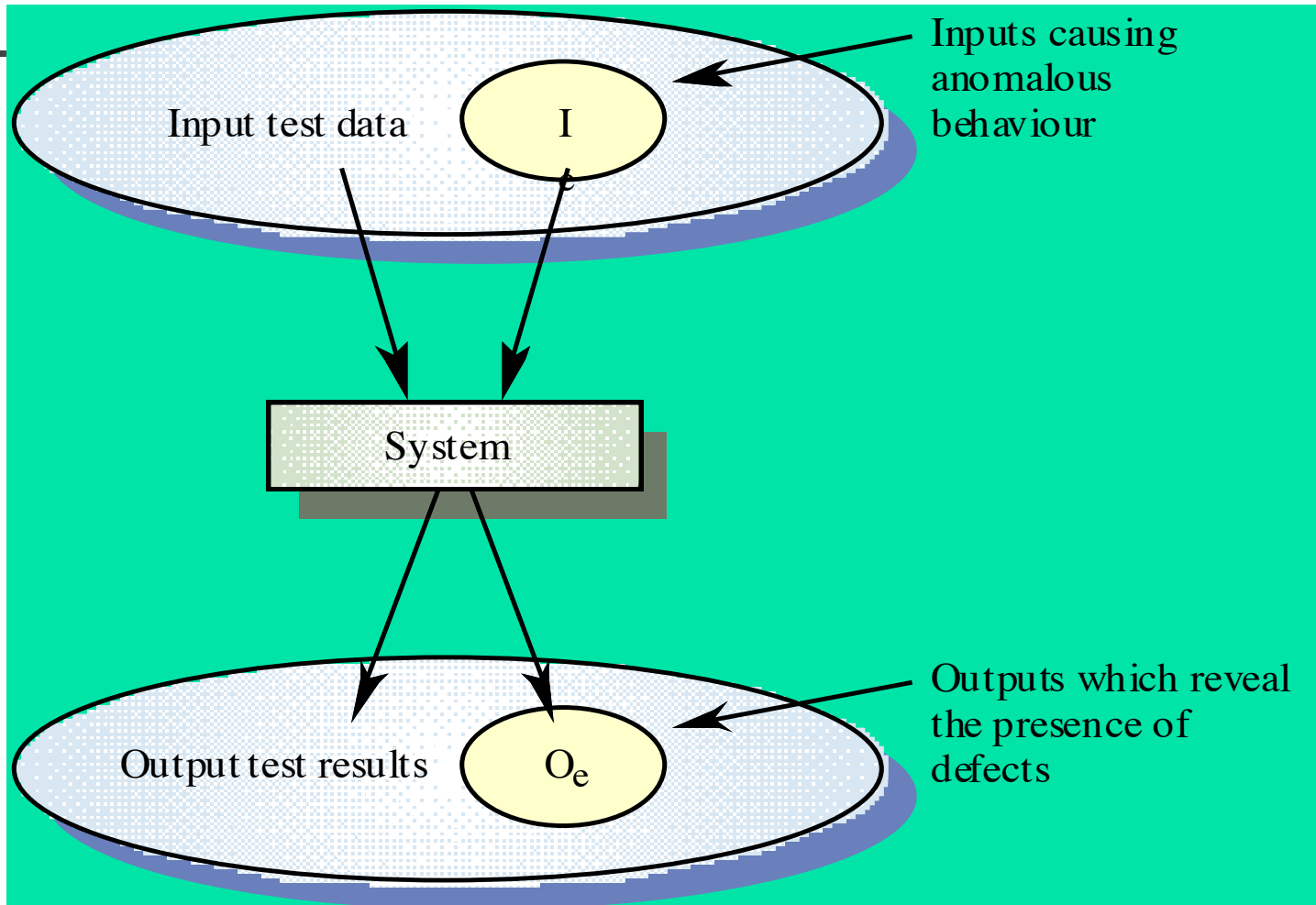
Most organizations have independent testing groups to perform black box testing. These testers are not the developers and are often referred to as *third-party testers*. output should be for a given input into the program

Black-box testing principles



A black-box test takes into account only the input and output of the software without regard to the internal code of the program.

Black Box Testing





Strategy Black Box Testing

- Test of User Requirement / Specification
- Equivalence Partitioning
- Boundary Value Analysis / Limit Testing
- Decision Table
- *Cause Effect Graph*
- *Error Guessing*
- etc



Test Of User Requirement

Requirement: When a user lands on the “Go to Jail” cell, the player goes directly to jail, does not pass go, does not collect \$200. On the next turn, the player must pay \$50 to get out of jail and does not roll the dice or advance. If the player does not have enough money, he or she is out of the game.

There are many things to test in this short requirement above, including:

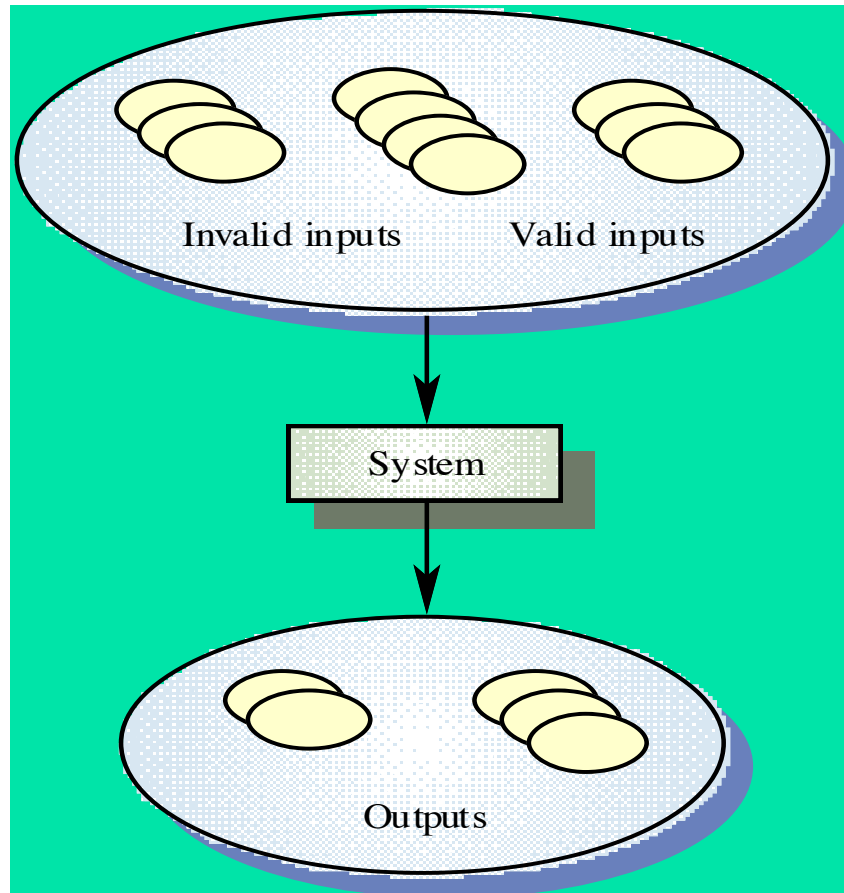
1. Does the player get sent to jail after landing on “Go to Jail”?
2. Does the player receive \$200 if “Go” is between the current space and jail?
3. Is \$50 correctly decremented if the player has more than \$50?
4. Is the player out of the game if he or she has less than \$50?



Equivalence Partitioning

- Divide the input domain into classes of data for which test cases can be generated, based on equivalence classes for input conditions
- An equivalence class can represent a set of valid or invalid states
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member

Equivalence partitioning





General Rules in EQP

Equivalence classes can be defined by :

A. If an input condition specifies a range or a specific value, 1 valid and 2 invalid equivalence class defined

Ex: [1..999]

- Valid Test Case : 1 value in range
- Invalid Test Case : 2 value out of range, < bottom bound, > upper bound



General Rule in EQ

B. If an input condition specifies a boolean or a member of a set, 1 valid and 1 invalid equivalence class defined

Ex : [red, blue, yellow]

- Valid Test Case : 1 value in domain
- Invalid Test Case : 1 value out of domain



Example

Requirement of subprogram to be tested

IS : The subprogram takes an integer input
in the range $[-100, 100]$

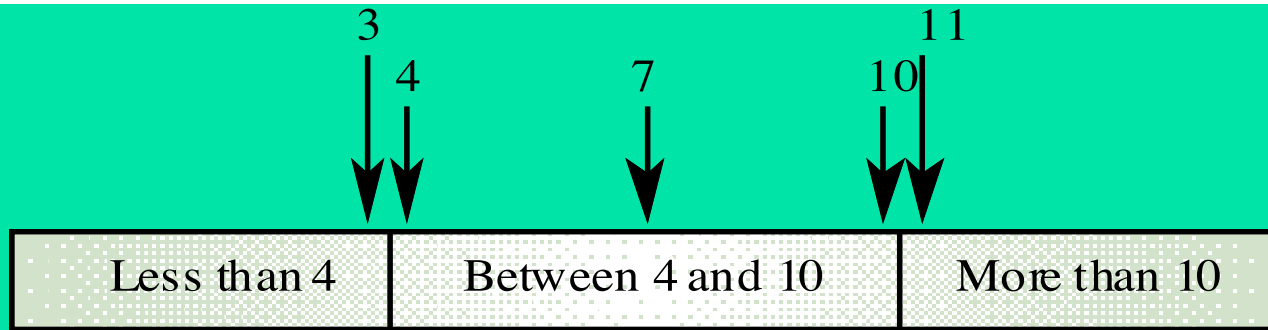
FS : Output is the sign of the input value
(value 0 is considered positive)



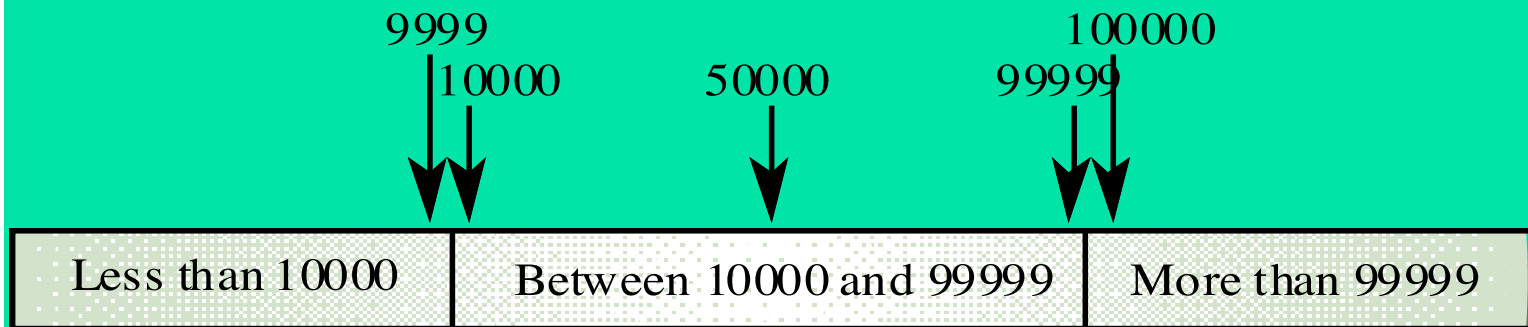
Boundary Value Analysis (BVA)

- A kind of BBT that complements equivalence partitioning
- Focus of the testing is on the boundaries of the input domain
 - If input condition specifies a range bounded by a certain values, say, a and b , then test cases should include
 - The values for a and b
 - The values just above and just below a and $b \rightarrow (a-1), (a+1), (b-1), (b+1)$

Boundary Value Analysis



Number of input values



Input values



Boundary Value Analysis (BVA)

- If an input condition specifies a number of values N , test cases should be exercise
 - the numbers (N) ,
 - the values just above and just below the values : $(N-1)$ & $(N+1)$



Decision Table

- Decision Table are used to record a complex bussiness rules that must be implemented in the program, therefore must be tested

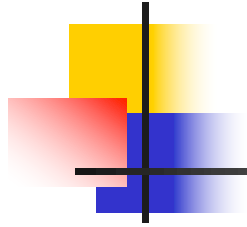


Example Decision Table

If a Player (A) lands on property owned by another player (B), A must pay rent to B. If A does not have enough money to pay B, A is out of the game.

Table 7: Decision table

	Rule 1	Rule 2	Rule 3
Conditions			
A lands on B's property	Yes	Yes	No
A has enough money to pay rent	Yes	No	--
Actions			
A stays in game	Yes	No	Yes



THANK YOU