# WHITE-BOX TESTING:



- Knowing internal working of a product, test can be conducted to ensure that internal operations are performed according to specifications.
- It is close examination of Procedural detail
- Logical Paths are tested
- Collaboration b/w components are tested by providing test cases that exercise specific set of conditions.

- what to do?
  - → Identify logical Paths
  - → Develop test cases
  - → Evaluate Results.

As it is not possible to test every Logical path, a limited no. of important paths can be selected.

- Also k/s glass-box testing
- white BOX-testing is an essential stage as testing of module is done at initial stages. [test design & code]

Test cases derived using White Box Testing

1.) ~~Can~~ Guarantee that all independent Paths within module have been executed at-least once

2.) Exercise all logical decisions on their true & false side

3.) Execute all loops within the operational boundaries

4.) Check data structure to ensure the validity.

## Basic Path Testing:

white box testing technique used for designing test cases, intend to examine all possible paths of execution at least once.

Creating and executing test cases for all possible paths and 100% logical coverage.

what is logical coverage?

eg ⇒

```
scanf("%d", &x);
scanf("%d", &y);
while (x != y)
{ if(x>y)
      x = x-y};
   else
      y = y-x;
}
printf("x=", x);
printf("y=", y);
```

~~[illegible]~~ (

test case 1 : $x = y = n$, where n is any number

test case 2 : $x = n$, $y = n'$, where n & n' are different

test case 3 : $x > y$

test case 4 : $x < y$.

The test cases must be executed for every outcome (while & if) both.

Guidelines:

1.) Path testing is based on control structure of program for which flow graph is prepared

2.) It require complete knowledge of program structure

3.) It is used by developer as he knows the module

4.) choose enough paths in a program such that maximum logic coverage is achieved

Control flow Graph:

graphical representation of control structure of a program.

Notations:

1.) Node: represent one or more procedure statements. Nodes are numbered or labeled.
circle → ○

2.) Edges: Represent flow of control in a program. An edge must terminate at node.

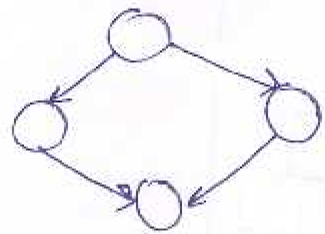
3.) Decision Node: A node with more than one arrow leaving it.

4.) Junction Node: A node with more than one arrow entering it

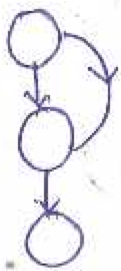
(5) **Regions**: Area bounded by edges and nodes. area outside graph is also considered a region.

**Notations:**

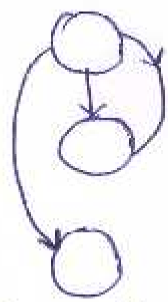
a) Sequence.



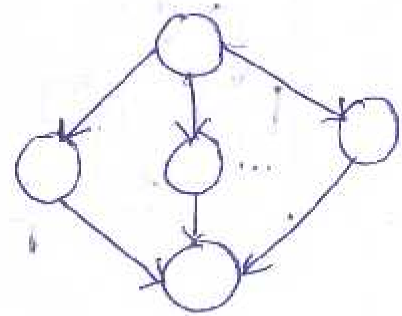
b) if-Then-Else.



c) Do-while



(d) While-Do

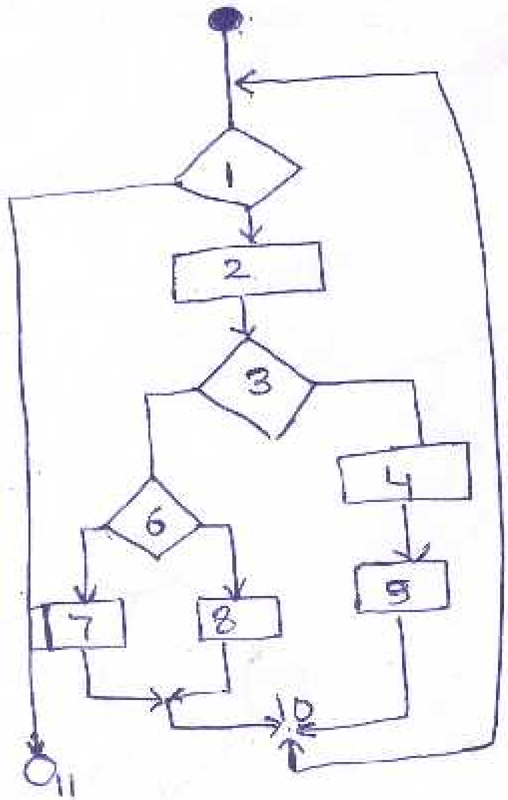


e.) Switch Case

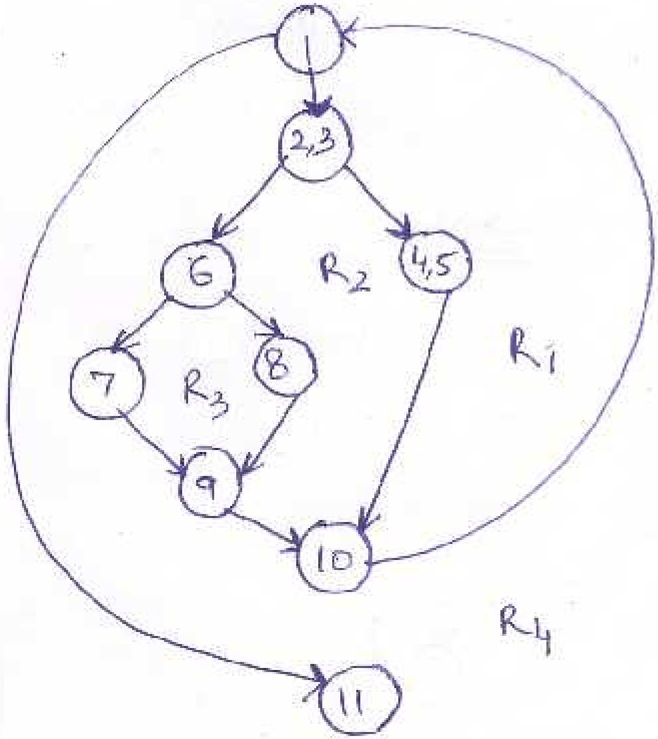
Flowchart is used to depict program control structure. → maps flowchart into a corresponding flow graph.

Circle → flow graph node, represents one or more procedural statements

sequence of process boxes and a decision diamond can map into a single node




flow-chart




flow graph.

Ex-1

```
main ()
{
  int number, index;
1.) printf ("enter a number");
2.) scanf ("%d, & number);
3.) index = 2;
4.) while (index <= number-1)
5.  {
6.    if (number % index == 0)
7.    {
8.      printf ("not a prime no");
9.      break;
10.   }
11.   index++;
12. }
13. if (index == number)
14.     Printf ("Prime no.");
15  } // end main.
```

n = 3

2 <= 3-1

2 <= 2

{

(3 <= 2) (false)

3 = 3

Print Prime

---

n = 4

2 <= 4-1

2 <= 3

{

True ✓ $\boxed{4\%2 == 0}$

Print no is Prime

---

n = 2

2 <= 1-1

false $\boxed{2 <= 0}$

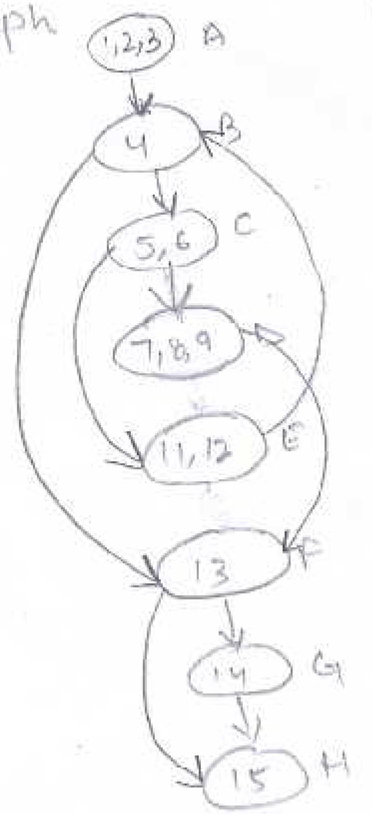
---

n = 1

output not displayed

(1) Draw DD graph

DD graph.



(2) Cyclomatic Complexity.

$$V(G) = e - n + 2$$
$$= 10 - 8 + 2$$
$$= 4$$

(3) Independent Paths.

1.) ABFH

2.) ABFGH

3.) ABC EBFGH

4.) ABC DFH.

Test case designed from list of independent Path

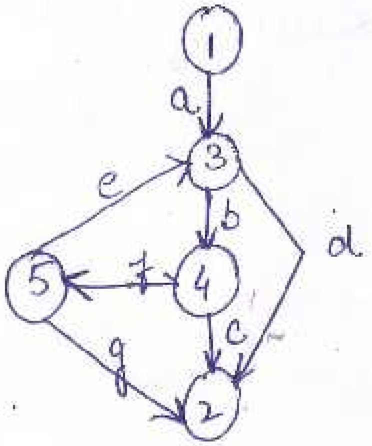
| Test case | I/p no. | Expected Result | Paths covered |
|---|---|---|---|
| 1 | 1 | No o/p displayed | A-B-F-H |
| 2 | 2 | Prime no. | A-B-F-G-H |
| 3 | 4 | Not a prime no | A-B-C-D-F-H |
| 4 | 3 | Prime no. | A-B-C-E-B-F-G-H |

# Graph - Matrices:

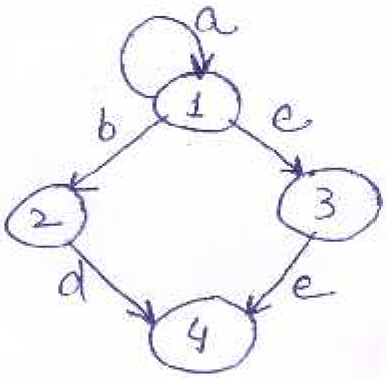


- The procedure for deriving flow graph.
- Determine set of basis paths.
- Assists in basis path testing.
- It is a square matrix (rows and columns)
- Each row and column corresponds to an identified node.

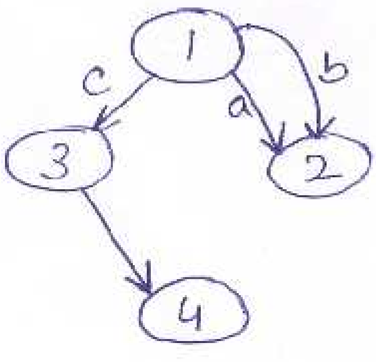
## Graph - Matrix




| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | | | a | | |
| 2 | | | | | |
| 3 | | d | | b | |
| 4 | | c | | | f |
| 5 | | g | e | | |




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | a | b | c | |
| 2 | | | | d |
| 3 | | | | e |
| 4 | | | | |




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | a+b | c | |
| 2 | | | | |
| 3 | | | | d |
| 4 | | | | |

-> Tabular representation of flow graph.
-> link weight to each matrix entry.

Link weight properties:

-> The probability that a link (edge) will be executed.

-> The processing time expended during traversal of a link.

-> The memory required during traversal of a link

-> Resources required.

# Control Structure Testing:

→ condition testing:

- The test cases method exercise the logical conditions.
- Simple condition – Boolean variable (0,1) or a relational expression. (¬) NOT.

$E_1$ < relational – operator > $E_2$

↓ < <, ≤, =, ≠, >, ≥ >

arithmetic expressions

Boolean operator – AND, OR, NOT

- If condition is incorrect, then atleast one component of condition is incorrect.

eg: (A||B) && C

A – T | B – F | C – T → True

A – F | B – T | C – F → False

A – F | B – F | C → not evaluated.

Loop testing: (focus on validity of loop constructs)

-> It can fix loop repetition issues

-> loop testing can reveal performance bottle necks

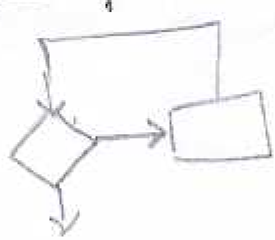
-> Uninialized variables can be determined

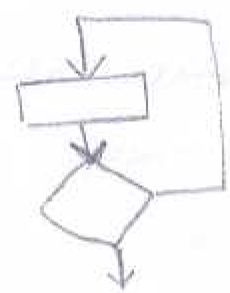
-> Identify loop initialization problems

checked at three different levels:

-> when loop is entered
-> During its execution
-> when loop is left.

Simple loop:

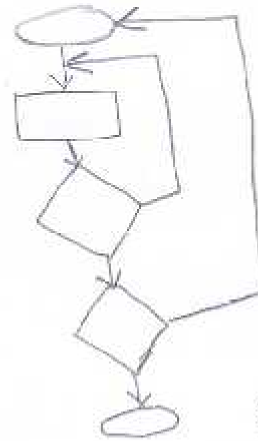
a) 

b) 

-> check whether loop control variable is -ve.
-> write one test case that execute statement inside loop.
-> check whether you can bypass loop or not

. Nested loop: (difficult to test)

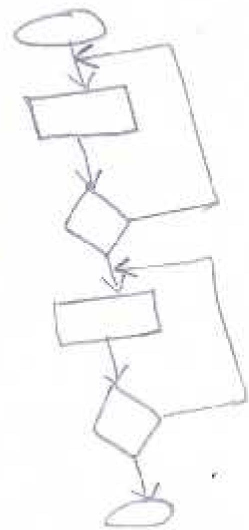
- when two or more loop are embedded.
- Strategy:
  - Start with inner most loop.
  - Conduct simple loop tests
  - Continue this outward
  - Continue until loops have been tested



fig

Concatenated loops.

- either loops can be independent (simple) or concatenated ie loop1 (value) is used in loop 2.



Unstructured loops.

-> Such loops are impractical to test

-> Such loops must be redesigned to reflect structured programming. ie converted into simple or concatenated loops.

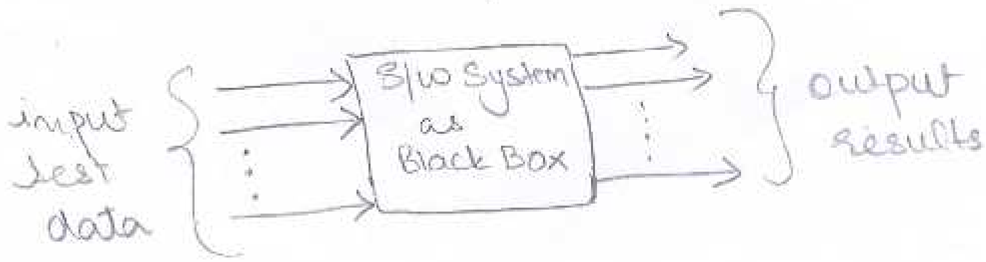
# Black-Box Testing



→ Also k/s behavioral testing
→ focus on the functional requirement
→ fully exercise the functional requirements for a program.

**Find errors like:**

1.) Incorrect / missing functions
2.) Interface errors
3.) Errors in Data structure
4.) Performance Errors
5.) Initialization and termination errors
6.) Test modules independently




Types:

① Equivalence Partitioning
② Boundary Value Analysis.

# I. Equivalence Class Testing

- In this input domain is divided into different equivalence data classes.
- Method typically reduce the total no. of test cases to finite set of test tables test cases., but still covering max. requirements.

Ex: 1 A program reads 3 no.s A, B, C range [1, 50] and prints largest. Design test cases for this program using Equivalence class technique.

Inputs

$I_1$ = ABC : $1 \leq A \leq 50$

$I_2$ = ABC : $1 \leq B \leq 50$

$I_3$ = ABC : $1 \leq C \leq 50$

$I_4$ = ABC : $A < 1$

$I_5$ = ABC : $A > 50$

$I_6$ = ABC : $B < 1$

$I_7$ = ABC : $B > 50$

$I_8$ = ABC : $C < 1$

$I_9$ = ABC : $C > 50$

~~$I_{10}$ = ABC~~

| ID | A | B | C | Expected Result | Test Case covered |
|---|---|---|---|---|---|
| 1 | 13 | 25 | 36 | C is greatest | $I_1, I_2, I_3$ |
| 2 | 0 | 13 | 45 | Invalid | $I_4$ |
| 3 | 51 | 34 | 17 | " | $I_5$ |
| 4 | 29 | 0 | 18 | " | $I_6$ |
| 5 | 36 | 53 | 32 | " | $I_7$ |
| 6 | 27 | 42 | 0 | " | $I_8$ |
| 7 | 33 | 21 | 51 | " | $I_9$ |

[illegible]

Ex: 2 A program determines next date in calendar. input < dd mm yyyy

range:

$1 \leq mm \leq 12$

$1 \leq dd \leq 31$

$1900 \leq yyyy \leq 2020$.

<u>Output</u>: → nextdate or invalid Date.

$I_1 = \langle m, d, y \rangle$ : $1 \leq m \leq 12$

$I_2 = \langle m, d, y \rangle$ : $1 \leq d \leq 31$

$I_3 = \langle m, d, y \rangle$ : $1900 \leq y \leq 2025$.

$I_4 = \langle m, d, y \rangle$ : $m < 1$

$I_5 = \langle m, d, y \rangle$ : $m > 12$

$I_6 = \langle m, d, y \rangle$ : $d < 1$

$I_7 = \langle m, d, y \rangle$ : $d > 31$

$I_8 = \langle m, d, y \rangle$ : $y < 1900$

$I_9 = \langle m, d, y \rangle$ : $y > 2025$.

<u>Testcases</u>:

| ID | mm | dd | yyyy | Expected res. | Test Case |
|---|---|---|---|---|---|
| 1 | 5 | 20 | 1996 | 21-5-1996 | $I_1, I_2, I_3$ |
| 2 | 0 | 13 | 2000 | Invalid | $I_4$ |
| 3 | 13 | 13 | 1950 | " | $I_5$ |
| 4 | 12 | 0 | 2007 | " | $I_6$ |
| 5 | 6 | 32 | 1956 | " | $I_7$ |
| 6 | 11 | 15 | 1899 | " | $I_8$ |
| 7 | 10 | 19 | 2026 | " | $I_9$. |

(BVA) - BOUNDARY VALUE ANALYSIS

10

① Boundary value checking (BVC)

eg (a) → variable range. (1 - 100)

1 — min → min+ (2)

nominal (50)

100 — max → max- (99)

Test cases = $4n+1$ where, n is no. of variable

$= 4 \times 1 + 1$

$= 5$ ✓

eg when 2 variable (a, b), where a, [1 to 100]
b [1 to 100]

Test cases - $4n+1$

$= 4 \times 2 + 1$

$= 9$. ✓

| a | b |
|---|---|
| 50 | 1 |
| 50 | 2 |
| 50 | 99 |
| 50 | 100 |
| 1 | 50 |
| 2 | 50 |
| 99 | 50 |
| 100 | 50 |
| 50 | 50 |

| | |
|---|---|
| A nom | B max- |
| A nom | B min+ |
| A nom | B min |
| A nom | B max |
| A min | B nom |
| A max | B nom |
| $\boxed{\text{A nom}}$ | $\boxed{\text{B nom}}$ |
| A min+ | B nom |
| A max- | B nom |

eg if we have 3 variable a, b, c

Test case $= 4n+1$
$= 4\times3+1$
$= 13.$

| a | b | c |
|---|---|---|
| 50. | 1 | 50 |
| 50 | 2 | 50 |
| 50 | 99 | 50 |
| 50 | 100 | 50 |
| 1 | 50 | 50 |
| 2 | 50 | 50 |
| 99 | 50 | 50 |
| 100 | 50 | 50 |
| 50 | 50 | 1 |
| 50 | 50 | 2 |
| 50 | 60 | 99 |
| 50 | 50 | 100. |
| 50 | 50 | 50 |

Q- Consider a prgm. for the determination of nature of roots of a quadratic eqn.
Its input is triple of integers (a, b, c) & values may be from interval [0, 100]
→ O/p must have one of the following
( Real roots, Img. roots, equal roots, Not a quadratic eq.)
Design boundary value test cases

eg. $ax^2 + bx + c = 0$

| a | b | c | Expected Output |
|---|---|---|---|
| 0 | 50 | 50 | not quadratic |
| 1 | 50 | 50 | Real Roots |
| 50 | 50 | 50 | Img. Roots |
| 99 | 50 | 50 | " |
| 100 | 50 | 50 | " |
| 50 | 0 | 50 | " |
| 50 | 1 | 50 | " |
| 50 | 99 | 50 | " |
| 50 | 100 | 50 | equal roots |
| 50 | 50 | 0 | Real roots |
| 50 | 50 | 1 | Real roots |
| 50 | 50 | 99 | Img. roots |
| 50 | 50 | 100 | Img. roots |

Real if $(b^2 - 4ac) > 0$

img $(b^2 - 4ac) < 0$

eq $(b^2 - 4ac) = 0$

Not quad (if $a = 0$)

② Robust Testing Method

a) Min
b) $Min^+$
c) $Min^-$
d) Nominal/average value
e) Max
f) $Max^+$
g) $Max^-$

1 → -min, ~~max~~ ; 100 → max, $max^+$ ; nom

0, 1, 2, 50, 99, 100, 101

eg 2 variable (a, b) range (0-100)

test cases :

$= 6n+1$

$= 6 \times 2 + 1$

$= \underline{13}.$

(Here also, n is no. variable)

| a | b |
|---|---|
| 50 | 0 |
| 50 | 1 |
| 50 | 2 |
| 50 | 99 |
| 50 | 100 |
| 50 | 101 |
| 0 | 50 |
| 1 | 50 |
| 2 | 50 |
| 99 | 50 |
| 100 | 50 |
| 101 | 50 |
| 50 | 50 |

③ Worst case Testing Method :

test cases : $5^n$.

- used where req. of temp, pressure, speed etc. is required.
- Not useful for boolean variable