Slide 19, 20, 21: White-box Testing

Made by Md. Mehedi Hasan Rafy

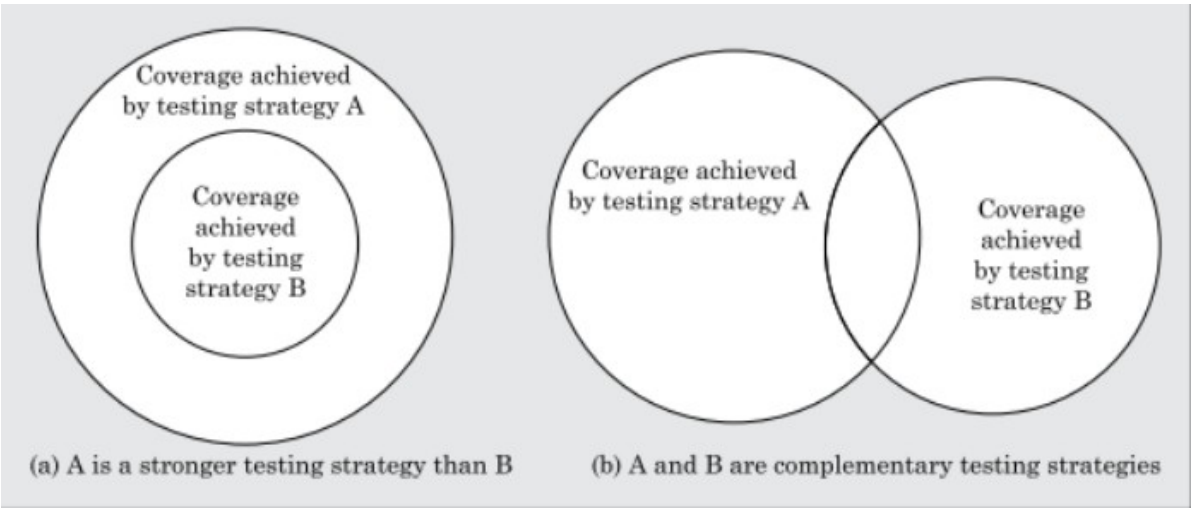
White-box testing

- A type of unit testing that designs test cases by analyzing source code.
- Two main strategies:
    - ***Fault-based*** – aims to reveal specific kinds of faults (e.g., mutation testing).
    - ***Coverage-based*** – aims to execute certain program elements to ensure they are tested.

Coverage-based testing

- Uses a criterion that defines what must be executed.
- Example: statement coverage → requires all statements to run at least once.
- A test suite is adequate if it covers all program elements defined by the chosen criterion.

Stronger vs Weaker Testing



Stronger: If strategy A covers everything strategy B does and more, then A is stronger than B.

Weaker: In that case, B is weaker because it only covers a subset of what A does.

Complementary: If A and B each cover some unique elements that the other doesn't, they're complementary (neither is strictly stronger).

Think of it like two sets:

- If Set A ⊃ Set B → A is stronger.
- If A and B overlap but neither fully contains the other → they're complementary.

Control Flow Testing

- A white-box testing technique that checks the execution order of statements in a program through a control structure.
- Relies on the program's control structure (like loops, if/else, branches) to design test cases.
- Often applied during unit testing.
- Test cases are represented using a control flow graph (CFG), which maps possible execution paths.

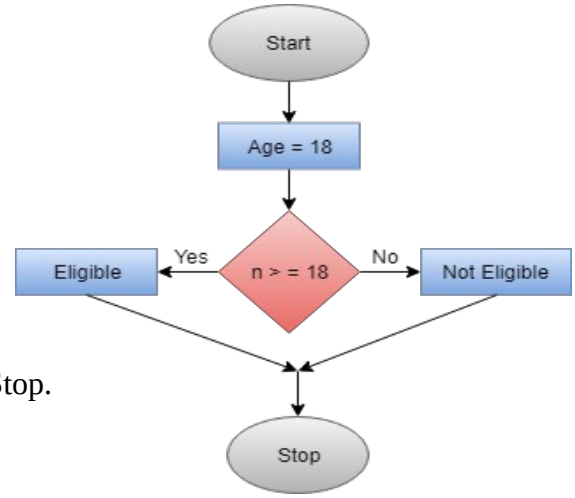
Control Flow Graph (CFG)

- A CFG shows all possible execution paths of a program.
- Built using:
    - ***Node*** → represents a step or procedure.
    - ***Edge*** → represents flow from one step to another.
    - ***Decision node*** → shows branching (e.g., if/else).
    - ***Junction node*** → where multiple flows come together.

Example: VoteEligibilityAge program

```
public class VoteEligibilityAge{
      public static void main (string []args) {
            int n = 45;
            if (n >= 18) {
                  system.out.println(“You are eligible for voting.”);
            } else {
                  system.out.println(“You are not eligible for voting.”);
            }
      }
}
```

In this example,



- Start node → assign n.
- Decision node → check n >= 18.
  - If true → go to “Eligible” node.
  - If false → go to “Not Eligible” node.
- Both paths join at a junction node, then end at Stop.

Purpose:

- Helps testers design test cases that cover all execution paths.
- Ensures correct order and flow of execution.

<u>Statement Coverage Testing</u>

- Ensures every statement in the code is executed at least once.
- It is used to calculate the total number of executed statements in the source code out of total statements present in the source code.
- ***Formula:*** $Statement\ Coverage = \frac{Number\ of\ executed\ statements}{Total\ number\ of\ statements} \times 100$
- ***Goal:*** In the internal source code, there is a wide variety of elements like operators, methods, arrays, looping, control statements, exception handlers, etc. Based on the input given to the program, some code statements are executed and some are not. The goal is to maximize executed statements (cover loops, branches, conditions, etc).

Example:

```
print(int a, int b) {
   int sum = a+b;
   if (sum > 0)
       print("Positive result");
   else
       print("Negative result");
}
```

In this example, total number of statements is 7. They are – function entry and exit points, declaration + assignment (`int sum = a+b;`), the `if` condition, the positive `print`, the `else` keyword, the negative `print`.

- ***Test suite 01:*** $a=5, b=4$.

  In this case, number of executed statements are 5.

  So, $statement\ coverage = \frac{5}{7} \times 100 = 71\%$

- ***Test suite 02:*** $a=-2, b=-7$

  In this case, number of executed statements are 6.

  Hence, $statement\ coverage = \frac{6}{7} \times 100 = 85\%$

<u>Branch Coverage Testing</u>

- A type of white-box testing using the control flow graph (CFG) that ensures every branch (true/false outcome of each decision) is executed at least once.
- Similar to decision coverage, but broader:
  - ***Decision coverage*** → each decision outcome.
  - ***Branch coverage*** → all branches from every decision point in the code.
- It uses a control flow graph to calculate the number of branches.
- To calculate the branch coverage, path-finding method is used. In path-finding method, the number of executed branches is used to calculate branch coverage.

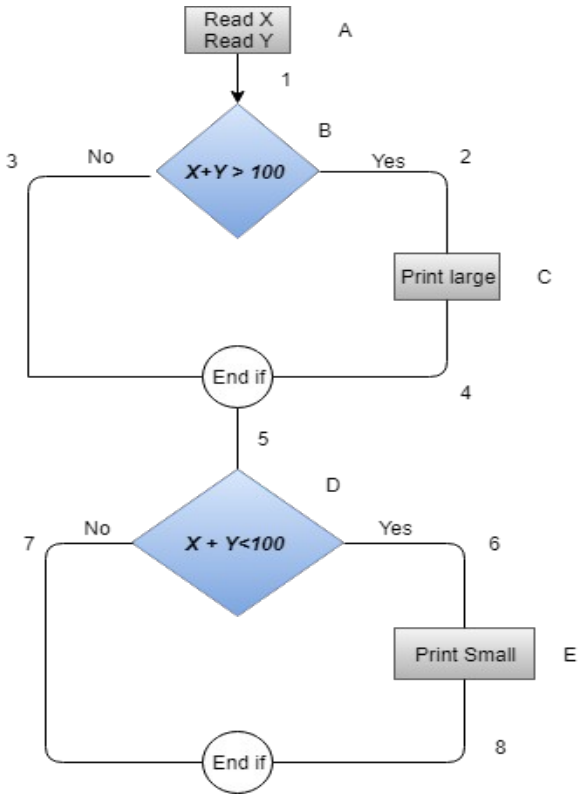
Example:

```
Read X
Read Y
IF X+Y > 100 THEN
   Print "Large"
ENDIF

IF X+Y < 100 THEN
   Print "Small"
ENDIF
```

There are two decision points here:

- `X+Y > 100` (true or false)
- `X+Y < 100` (true or false)



That makes 4 possible outcomes, but only some combinations are logically possible (since if `X+Y = 100`, both are false).

Control Flow Paths

- ***Path 1****:* Condition 1 = true, Condition 2 = false → print "Large" → end.
    - CFG edges: A1-B2-C4-5-D7
- ***Path 2****:* Condition 1 = false, Condition 2 = true → print "Small" → end.
    - CFG edges: A1-B3-5-D6-E8

Together, these 2 paths ensure:

- `X+Y > 100` is tested (true branch taken).
- `X+Y > 100` is tested (false branch taken).
- `X+Y < 100` is tested (true branch taken).
- `X+Y < 100` is tested (false branch taken).

Case	***Covered Branches***	***Path***	***Branch coverage***
Yes-No	1, 2, 4, 5, 7	A1-B2-C4-5-D7	2
No-Yes	1, 3, 5, 6, 8	A1-B3-5-D6-E8	

***** Have to recheck this topic again.*****

Path Coverage

Test suite should execute every *linearly independent path* (basis path) at least once.

Control Flow Graph (CFG)

A control flow graph consisting of a set of nodes and edges (N, E), such that each node n ε N corresponds to a unique program statement and an edge exists between two nodes if control can transfer from one node to the other.

- Directed graph: ***nodes = statements***, ***edges = possible control transfers***.
- Built using only 3 constructs: ***sequence, selection, iteration***.

- To draw: number program statements → make nodes → connect edges where control can transfer.
- **Iteration (loops)**: condition tested at start, control flows back from loop end to start, exit happens only when condition is false.

For example, CFG of the program given in figure (a) can be drawn as shown in figure (b).

```
int compute_gcd(int x, int y) {
  1 while(x!=y) {
  2      if(x>y) then
  3        x=x-y;
  4      else y=y-x;
  5    }
  6    return x;
     }
```

Fig – (a)

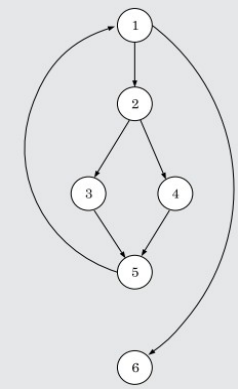


Fig – (b)

Some example of CFGs :

```
int compute_gcd(int x, int y) {
  1 while(x!=y) {
  2      if(x>y) then
  3        x=x-y;
  4      else y=y-x;
  5    }
  6    return x;
     }
```

Fig – (a)

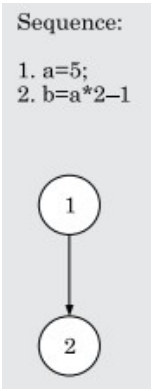


Fig – (b)

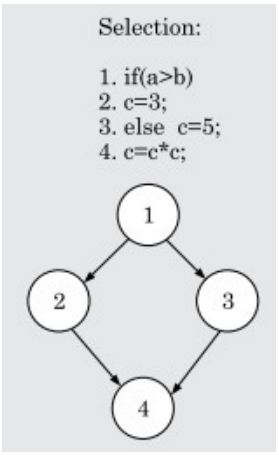


Fig – (c)

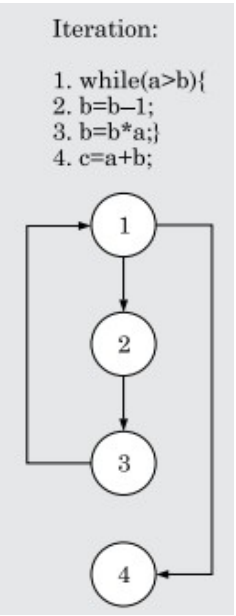


Fig – (d)

Fig – (a) : Example program.

Fig – (b) : CFG for Sequence of statements

Fig – (c) : CFG for Selection Construct

Fig – (d) : CFG for Iterative Construct

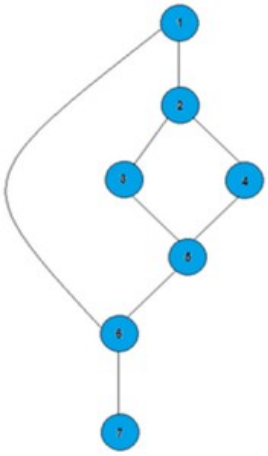
Path

- Sequence of nodes and edges from start node to terminal node in a CFG.
- Programs may have multiple terminal nodes (due to exits/returns).
- Covering all paths is impractical → loops can create infinitely many paths.
- Path coverage instead focuses on testing a subset: the linearly independent (basis) paths.

Linearly Independent set of paths

- A set of paths is linearly independent if each path introduces at least one new edge (or node) not already in other paths in the set.
- These paths form the basis set → the minimum set needed to test all unique flows.
- Example (with conditionals):
    - Path 1: 1 → 2 → 3 → 5 → 6 → 7
    - Path 2: 1 → 2 → 4 → 5 → 6 → 7
    - Path 3: 1 → 6 → 7

1. If A= 50
2. THEN IF B>C
3. THEN A =B
4. ELSE A=C
5. ENDIF
6. ENDIF
7. Print A



McCabe's Cyclomatic Complexity Metric

- Used to compute the ***upper bound*** (not an exact number) on the number of linearly independent paths through a program.
- Useful for ***complex programs*** where counting paths directly is hard.
- ***Note:*** It gives the maximum number of paths, not the exact set.

Three Methods to Compute V(G)

1. ***Graph formula:*** $V(G) = E - N + 2$

   Where,

   E → number of edges.

   N → number of nodes.

   Example: $E = 7, N = 6 \rightarrow V(G) = 3$.

2. ***Bounded regions:*** $V(G) = Total\,number\,of\,bounded\,areas + 1$

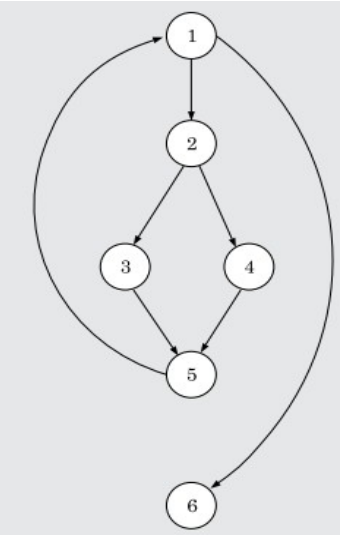
   Example: 2 regions → $2+1=3$.

   Note: In CFG any region enclosed by nodes and edges can be called as a bounded area. It increases with the number of decision statements and loops.

3. ***Decisions/loops:*** $V(G) = No.\,of\,decision/loop\,statements + 1$

   Example: 2 decisions/loops → $2+1=3$.

```
int compute_gcd(int x, int y) {
  1 while(x!=y) {
  2     if(x>y) then
  3       x=x-y;
  4     else y=y-x;
  5   }
  6   return x;
    }
```



Steps for Path Coverage–Based Testing

1. Draw the CFG of the program.
2. Compute McCabe's metric V(G).
3. Use V(G) to find the minimum number of test cases required.
4. Design & run test cases → check actual path coverage.
5. Repeat until ~90% path coverage is achieved.