Software Testing

Jamal Madni

CECS 445

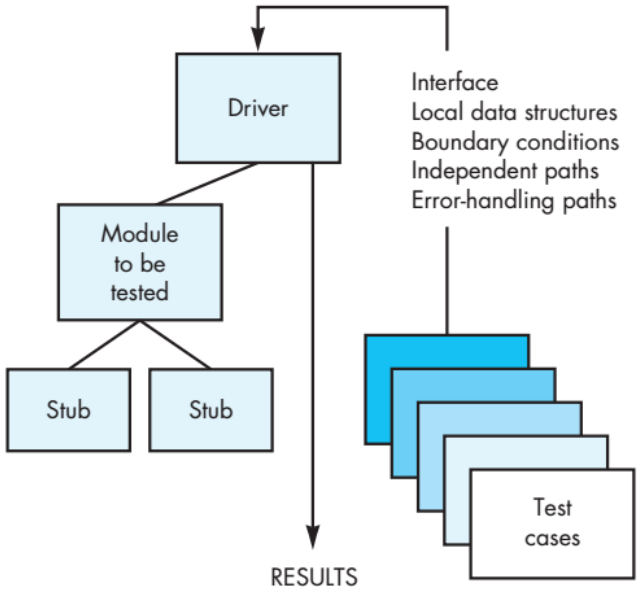
Lecture 12: March 16th, 2021

Intro to Software Testing

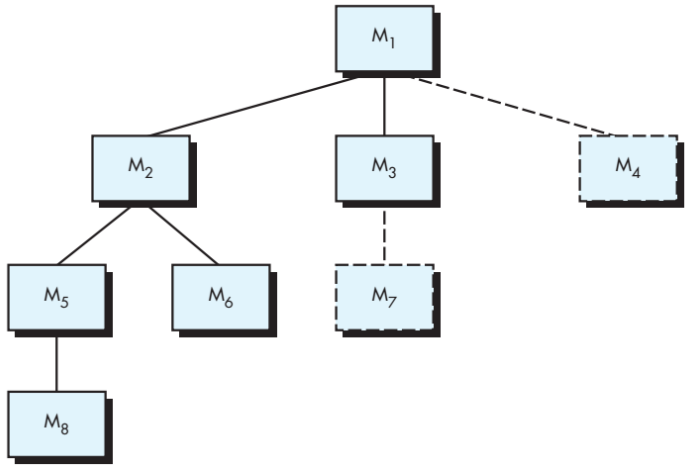
Verification: "Are we building the product right?"

Validation: "Are we building the right product?"

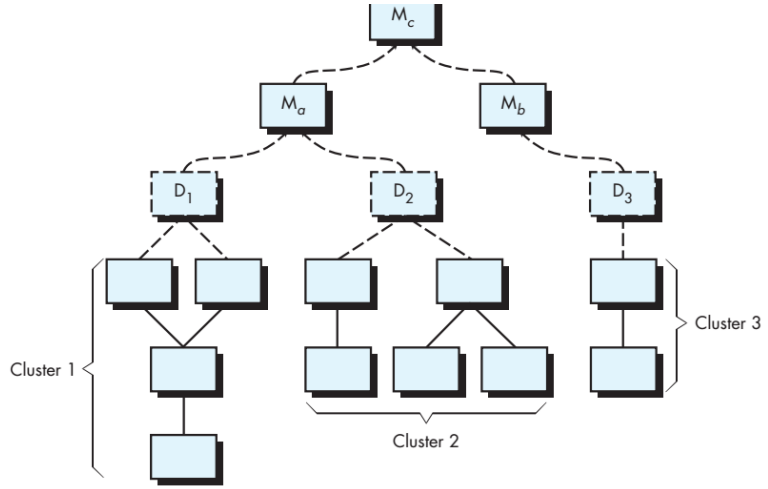
Testing Structure



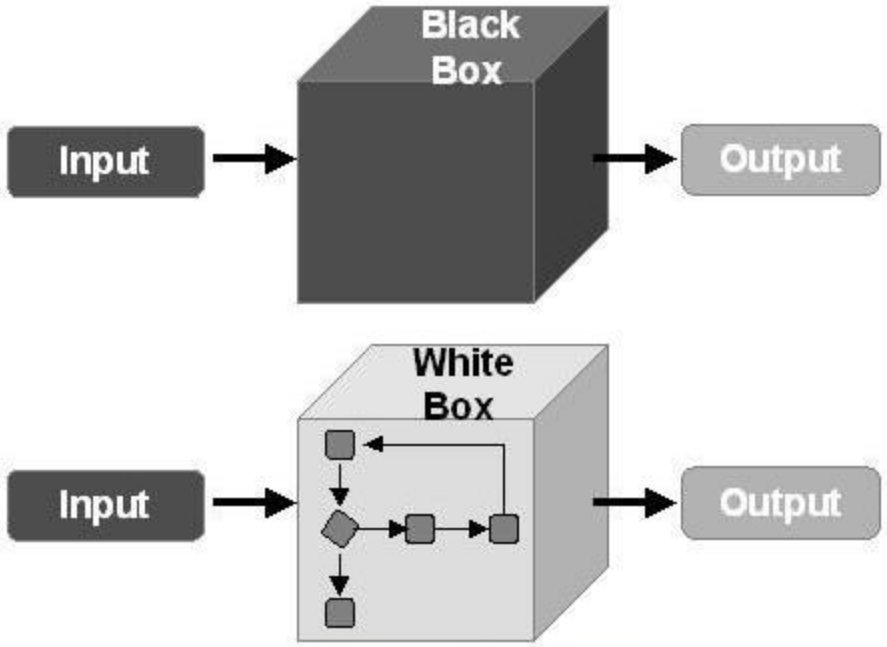
Top-Down Testing

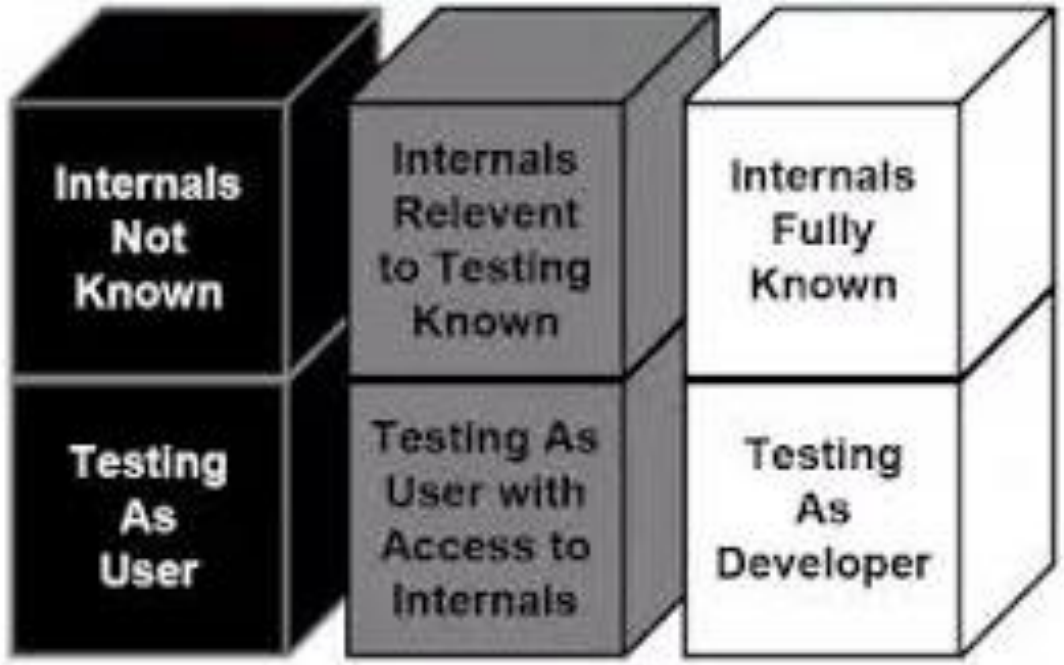


Bottom-Up Testing

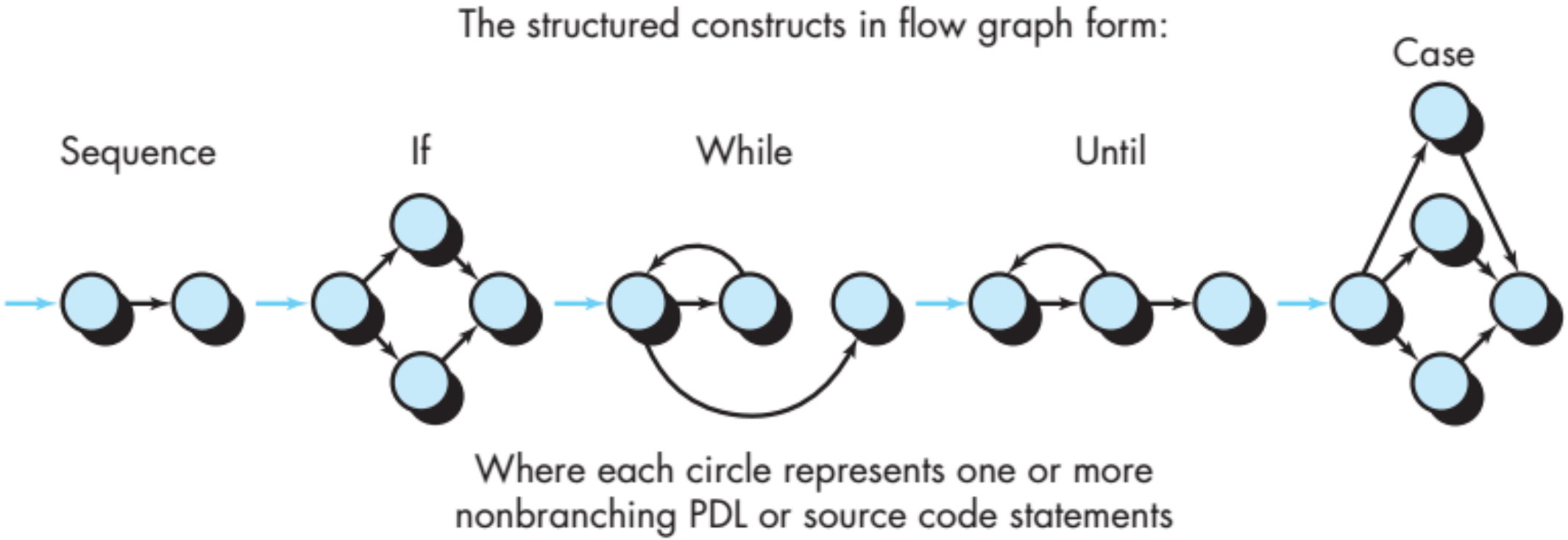


Black Box vs. White Box Testing

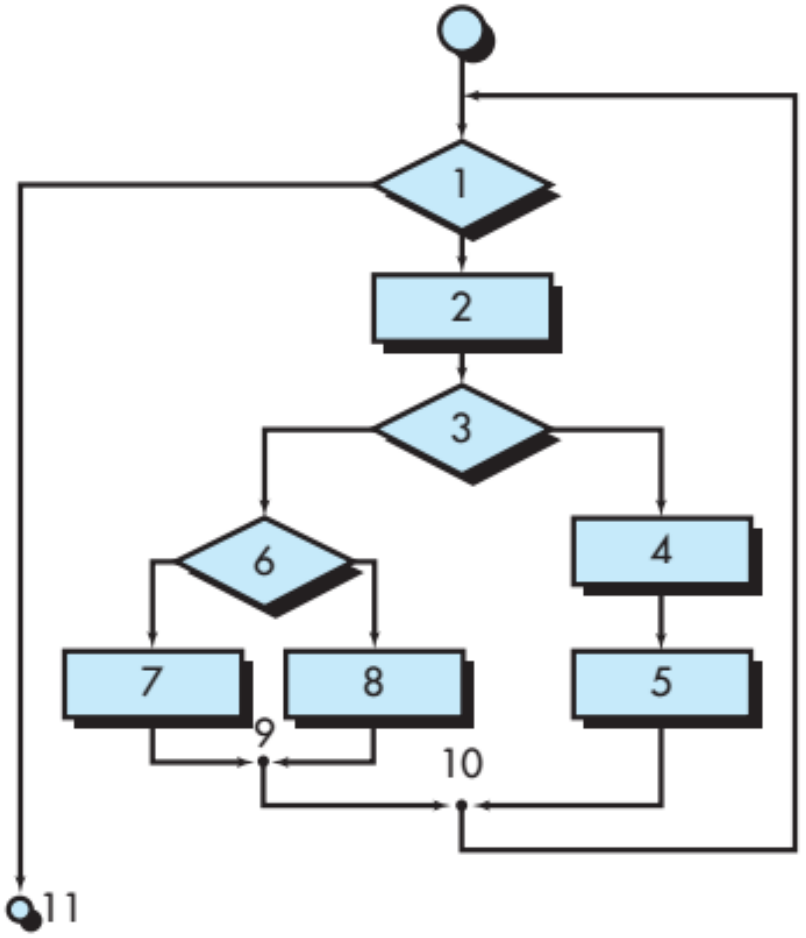




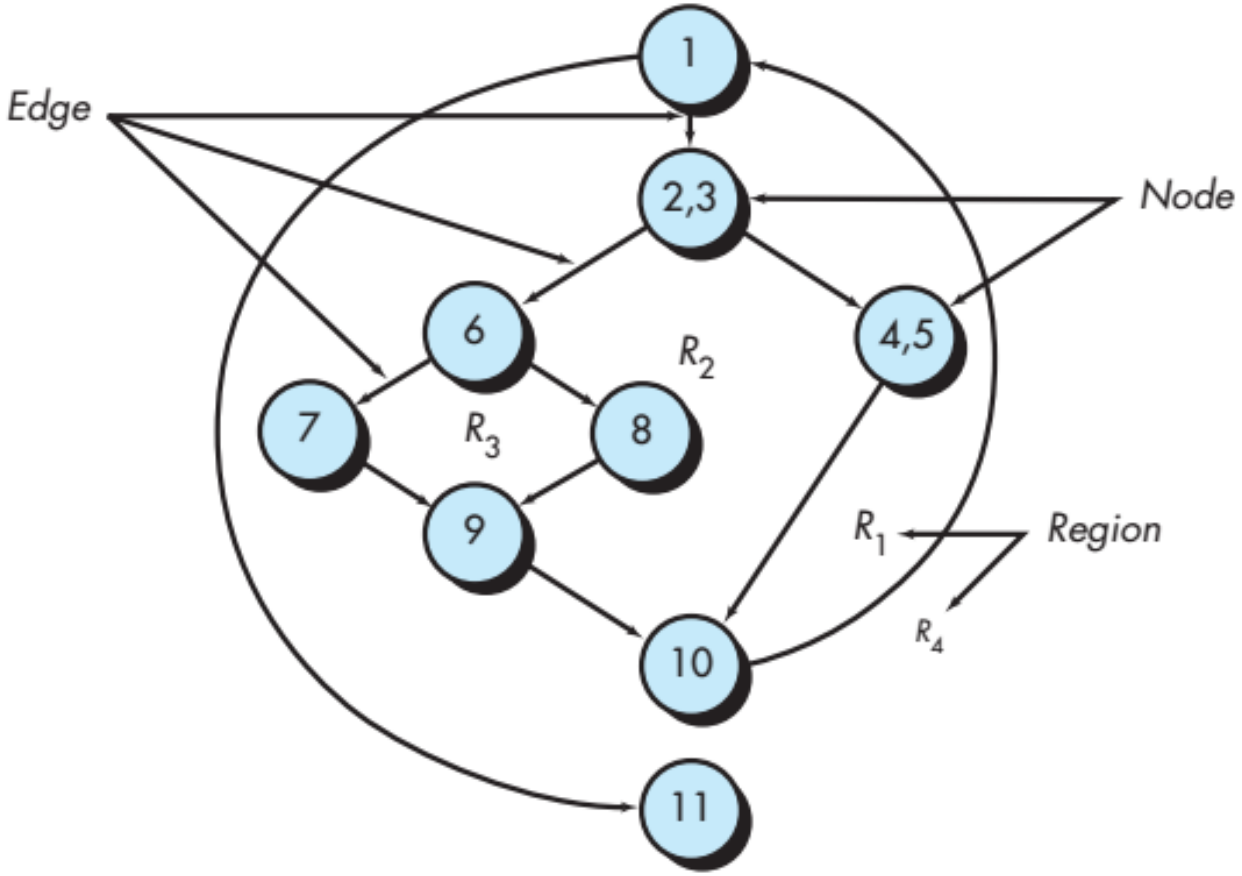
Basis-Path Testing & Flow Graph Notation



Flow Chart vs. Flow Graph

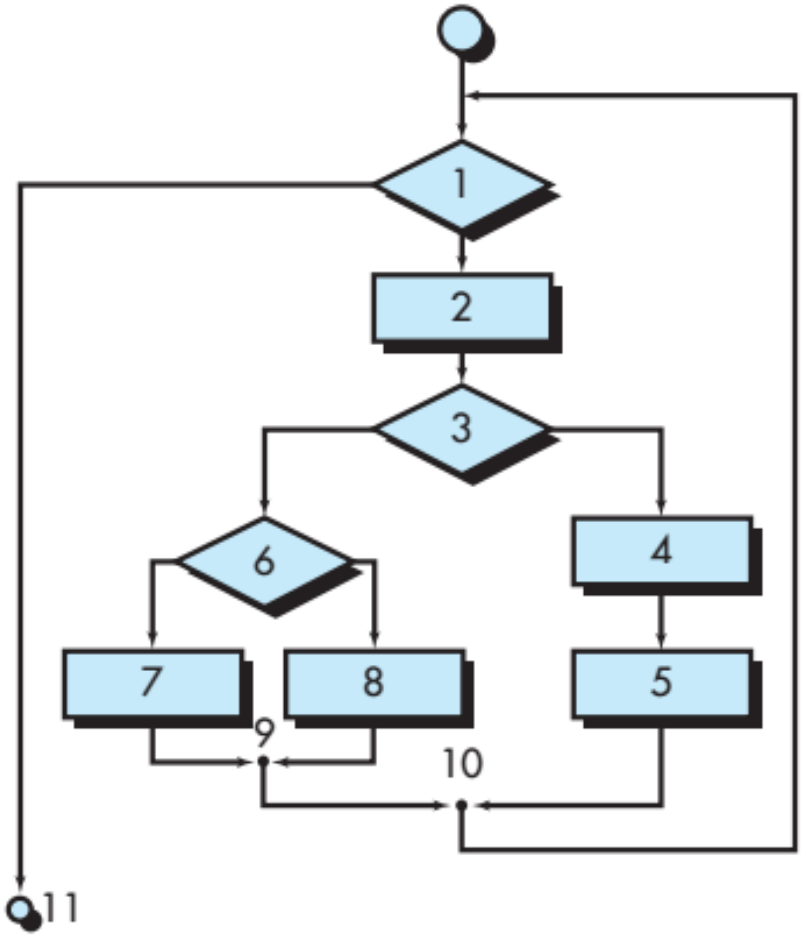


(a)

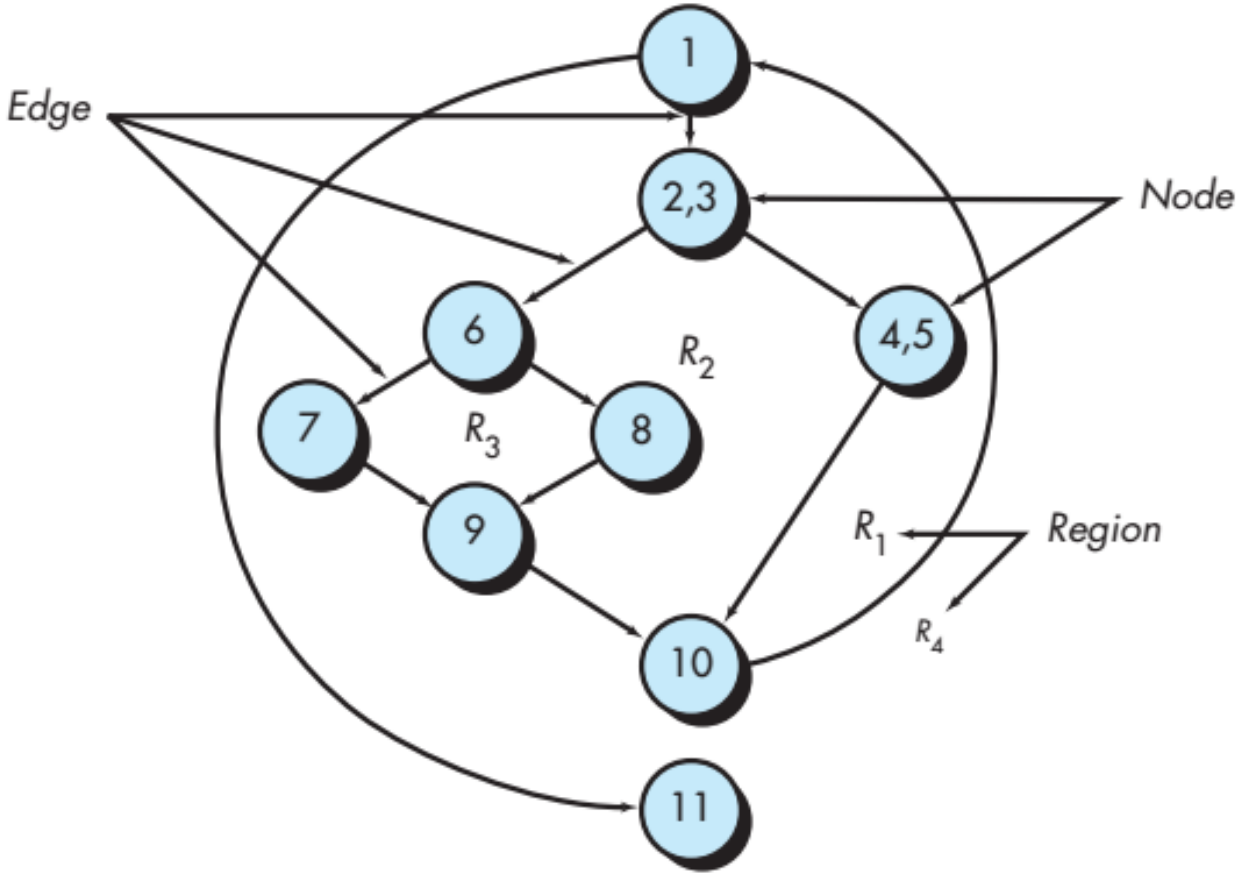


(b)

Flow Chart vs. Flow Graph

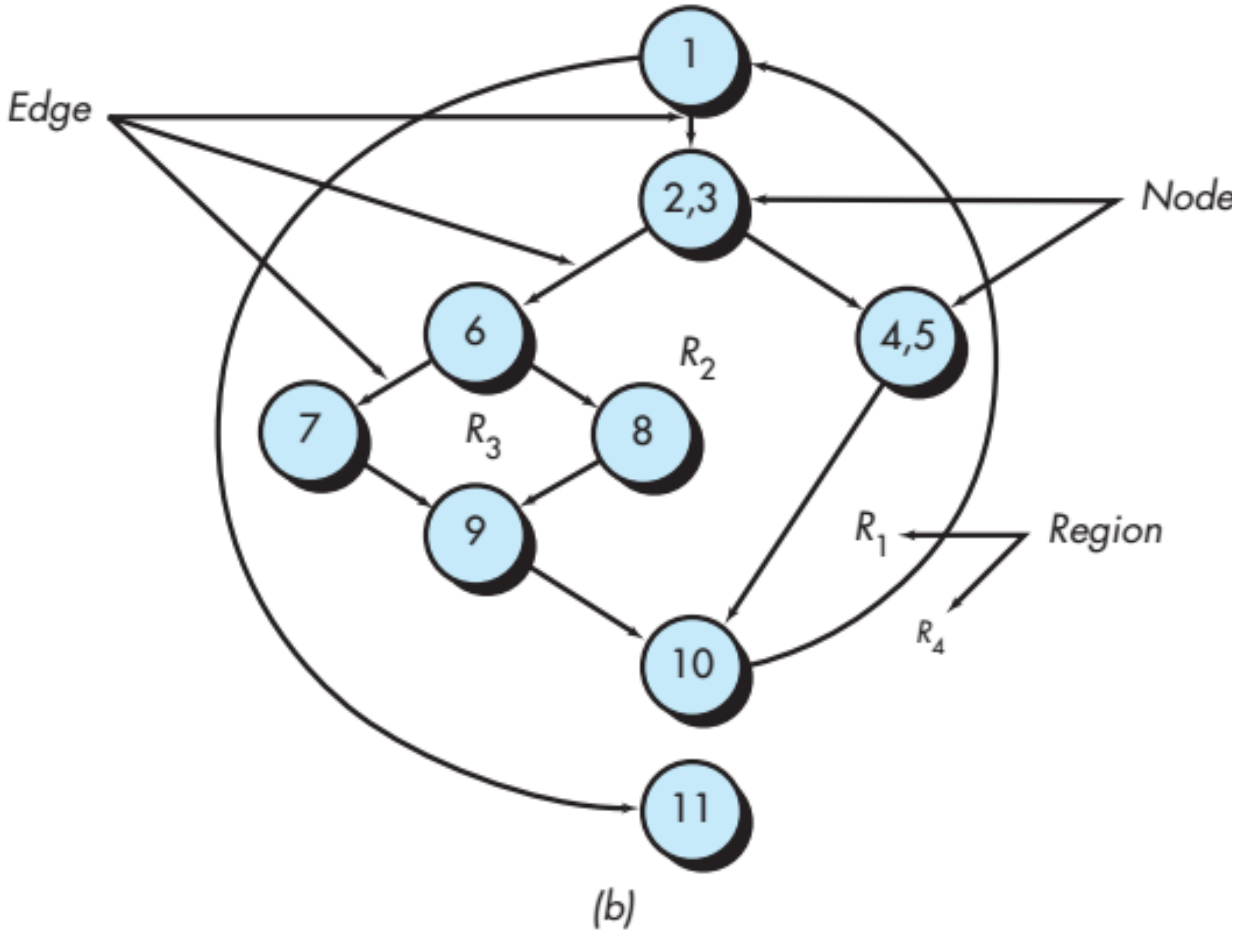


(a)



(b)

Independent Program Paths



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

Cyclomatic Complexity

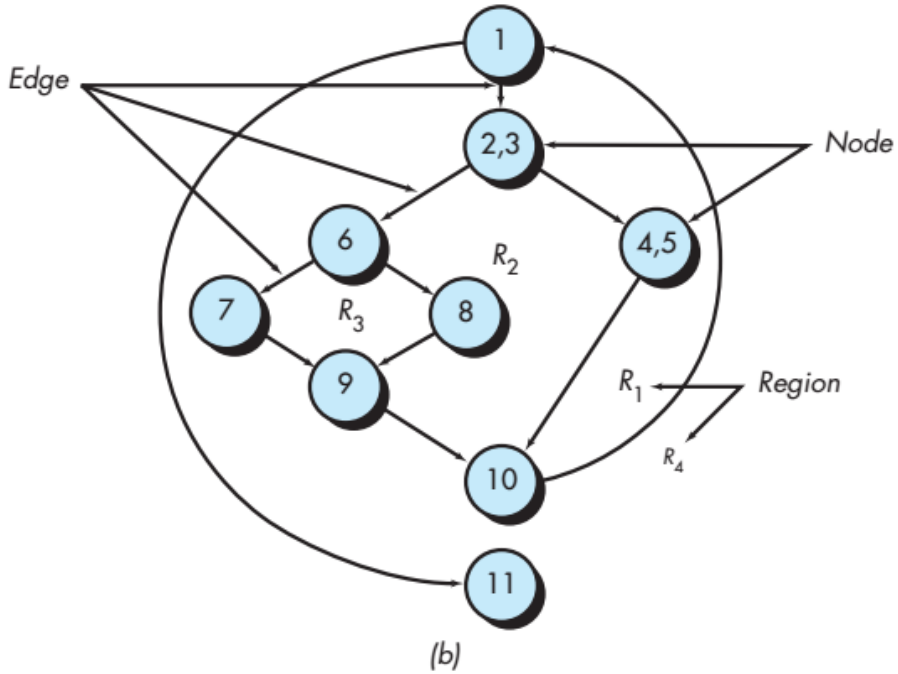
Cyclomatic complexity $V(G)$ for a flow graph G is defined as

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

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

V(G) = 11 Edges – 9 Nodes + 2
= 4

V(G) is upper bound on number of linearly independent paths



Path 1: 1-11

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

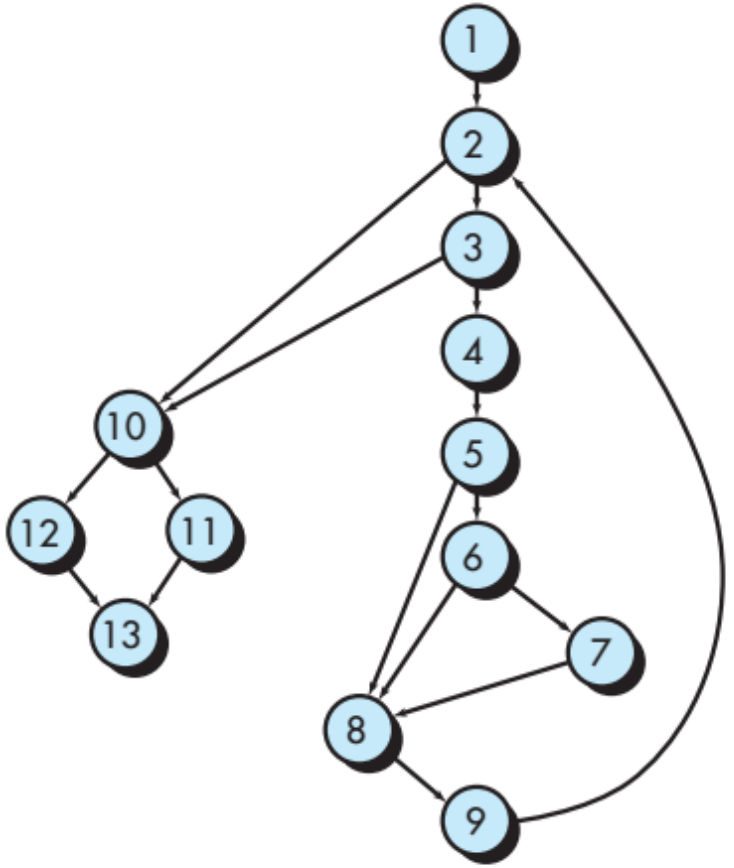
Path 3: 1-2-3-6-8-9-10-1-11

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

Deriving Test Cases

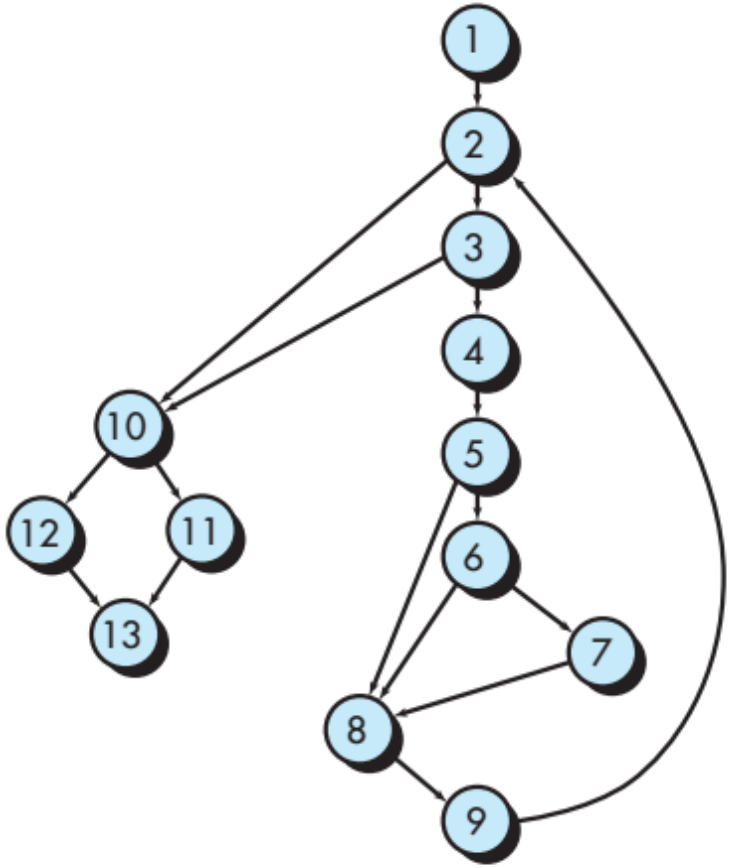
- Step 1: convert code or design to flow path

```
PROCEDURE average;

    *  This procedure computes the average of 100 or fewer
       numbers that lie between bounding values; it also computes the
       sum and the total number valid.

    INTERFACE RETURNS average, total.input, total.valid;
    INTERFACE ACCEPTS value, minimum, maximum;

    TYPE value[1:100] IS SCALAR ARRAY;
    TYPE average, total.input, total.valid;
         minimum, maximum, sum IS SCALAR;
    TYPE i IS INTEGER;
1   i = 1;
    total.input = total.valid = 0;
    sum = 0;
2   DO WHILE value[i] <> –999 AND total.input < 100
3       increment total.input by 1;
4       IF value[i] > = minimum AND value[i] < = maximum
5           THEN increment total.valid by 1;
7                sum = s sum + value[i]
            ELSE skip
6       ENDIF
8       increment i by 1;
9   ENDDO
    IF total.valid > 0
10      THEN average = sum / total.valid;
11      ELSE average = –999;
12  ENDIF
13  END average
```



Deriving Test Cases

- Step 2: determine cyclomatic complexity of the resultant flow graph



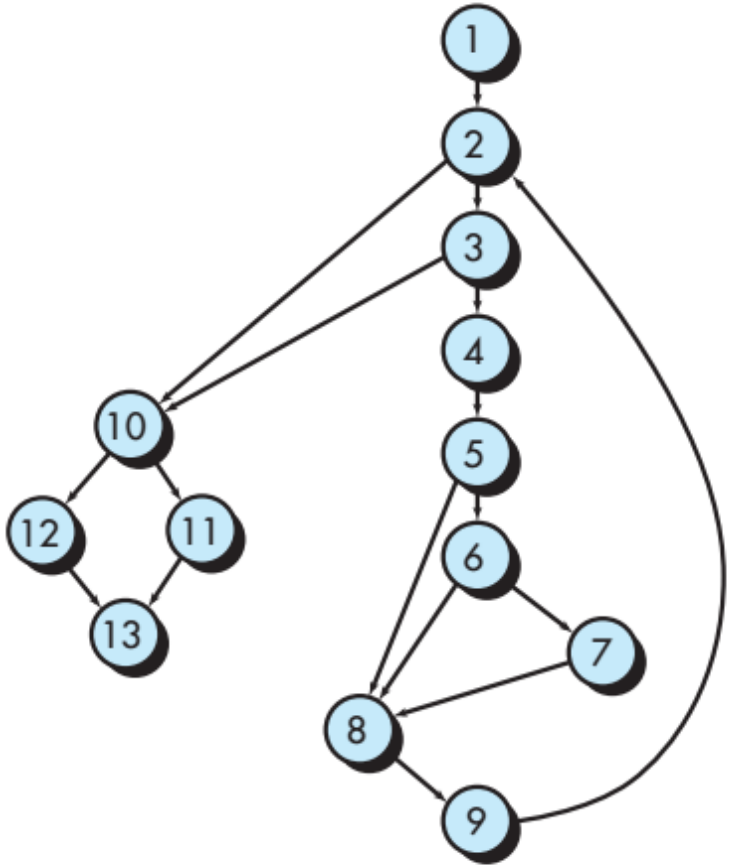
$$V(G) = 6 \text{ regions}$$

$$V(G) = 17 \text{ edges} - 13 \text{ nodes} + 2 = 6$$

$$V(G) = 5 \text{ predicate nodes} + 1 = 6$$

Deriving Test Cases

- Step 3: determine a basis set of linearly independent paths



Path 1: 1-2-10-11-13

Path 2: 1-2-10-12-13

Path 3: 1-2-3-10-11-13

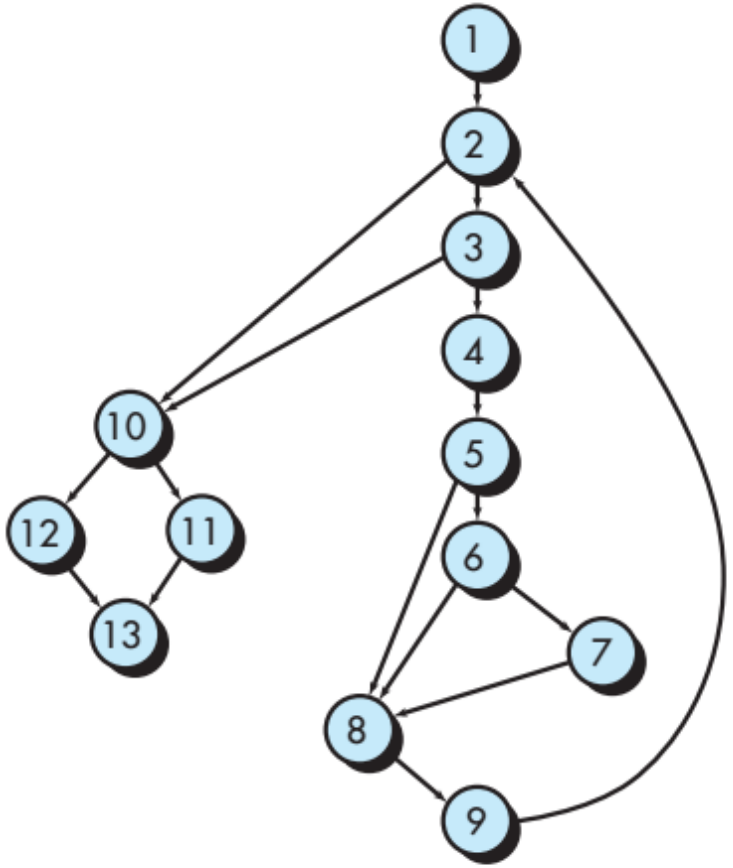
Path 4: 1-2-3-4-5-8-9-2-. . .

Path 5: 1-2-3-4-5-6-8-9-2-. . .

Path 6: 1-2-3-4-5-6-7-8-9-2-. . .

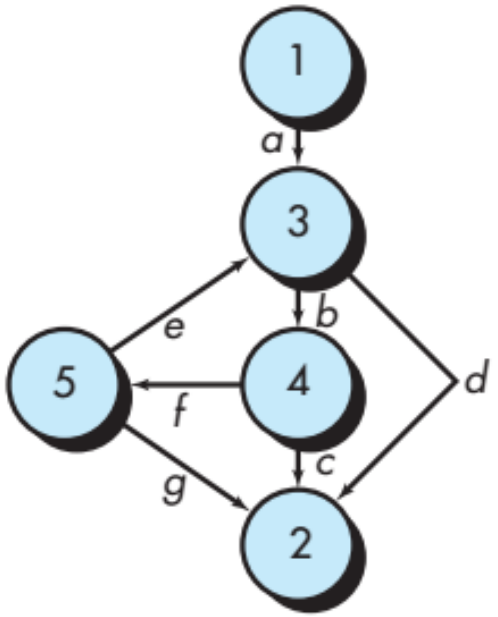
Deriving Test Cases

- Step 4: prepare test cases that will force execution of each independent path



Test case	Test parameters			
	P1	P2	P3	P4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Graph Matrix



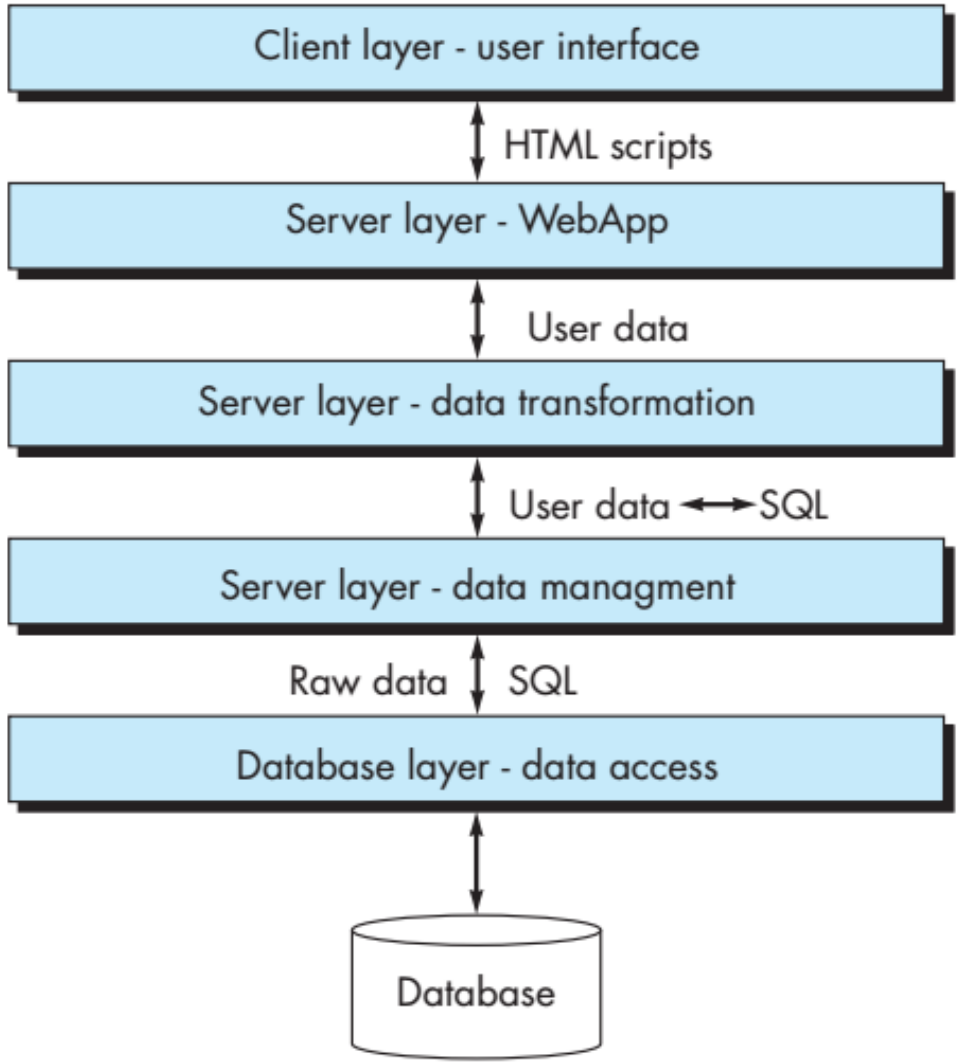
Flow graph

Node \ Connected to node	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		

Graph matrix

- 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
- The resources required during traversal of a link.

Database Flow Graph Example



Front-End Considerations