SENG 275

SOFTWARE TESTING

DR. NAVNEET KAUR POPLI

DEPT. OF ELECTRICAL AND COMPUTER ENGINEERING

MC/DC

University of Victoria

MC/DC (Modified Condition/Decision Coverage)

- Modified condition/decision coverage (MC/DC) looks at the **combinations of conditions**.
- However, instead of aiming at testing all the possible combinations, we follow a process in order to identify the "**important**" combinations.
- The goal of focusing on these important combinations is to manage the large number of test cases that one needs to devise when aiming for 100% path coverage.

MC/DC (Modified Condition/Decision Coverage)

- The idea of MC/DC is to ***exercise each condition in a way that it can, independently of the other conditions, affect the outcome of the entire decision*.**
- In short, this means that **every possible condition** of each parameter must have **influenced the outcome at least once**.
- **Linear** not exponential.
- The MC/DC criteria were developed to provide many of the benefits of exhaustive testing of Boolean expressions without requiring exhaustive testing.

University of Victoria

Branch and Condition Testing

- **Compound predicates:**
  - (((a || b) && c) || d) && e

- Should we test the effect of *individual* conditions on the outcome?

1. *Basic* ***condition***: each cond. true, false
2. *Branch and condition*: same, + branch
3. *Compound condition(path)*: each combination, 2^N (costly)
4. *Modified Condition / Decision Coverage* (MC/DC)

MC/DC: Modified Condition + Decision Coverage

- Basic condition + decision coverage + ...
  - *each basic condition should **independently** affect outcome of each decision*
- Requires:
  - For each basic condition C, two test cases,
  - values of all *evaluated* conditions except C are the same
  - compound condition as a whole evaluates to *true* for one and *false* for the other
  - N + 1 cases, for N conditions.

This is how normally testers work

- The rule of thumb is start with **branch coverage**: They always try to at least reach all the branches of the program.
- Whenever they see a more complicated expression, they evaluate the need for **condition + branch** coverage.
- If they see an even more complex expression, they consider **MC/DC**.

MC/DC is the industry standard in avionics

- MC/DC is used in **avionics** software development guidance DO-178B and DO-178C to ensure adequate testing of the most critical (Level A) software, which is defined as that software which could provide (or prevent failure of) continued safe flight and landing of an aircraft.
- **We want to test important combinations of conditions, without exponential blowup to test suite size.**

University of Victoria

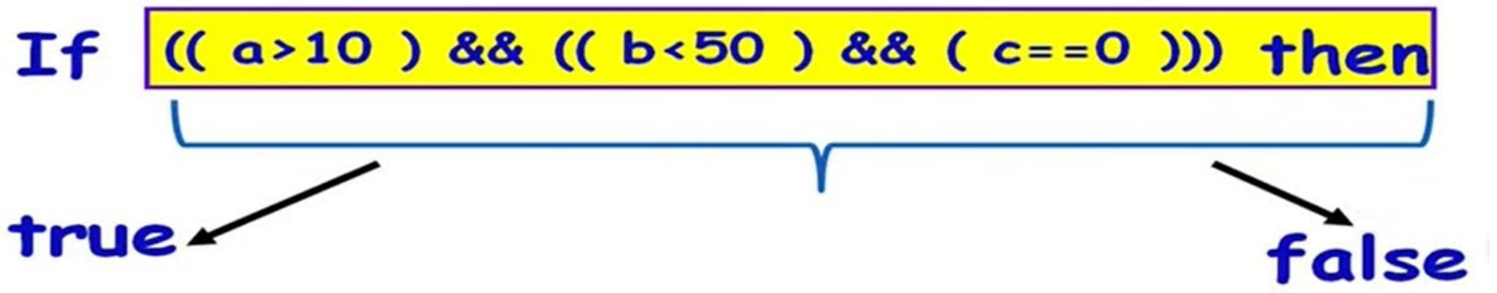
DO-178B/ED-12B Software Considerations in Airborne Systems and Equipment Certification

Failure Condition	Software Level	Coverage
Catastrophic	A	MC/DC
Hazardous / Severe	B	Decision Coverage
Major	C	Statement Coverage
Minor	D	
No Effect	E	

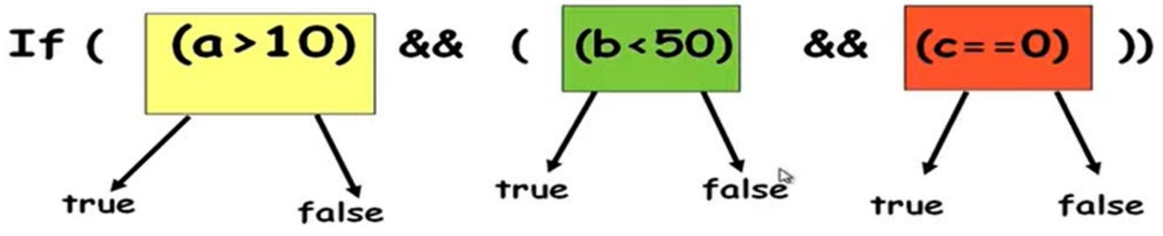
- The worldwide avionics software standard which all airborne software is required to comply.
- The world's strictest software standard
- Influences other domains including medical devices, transportation, and telecommunications.

Requirements

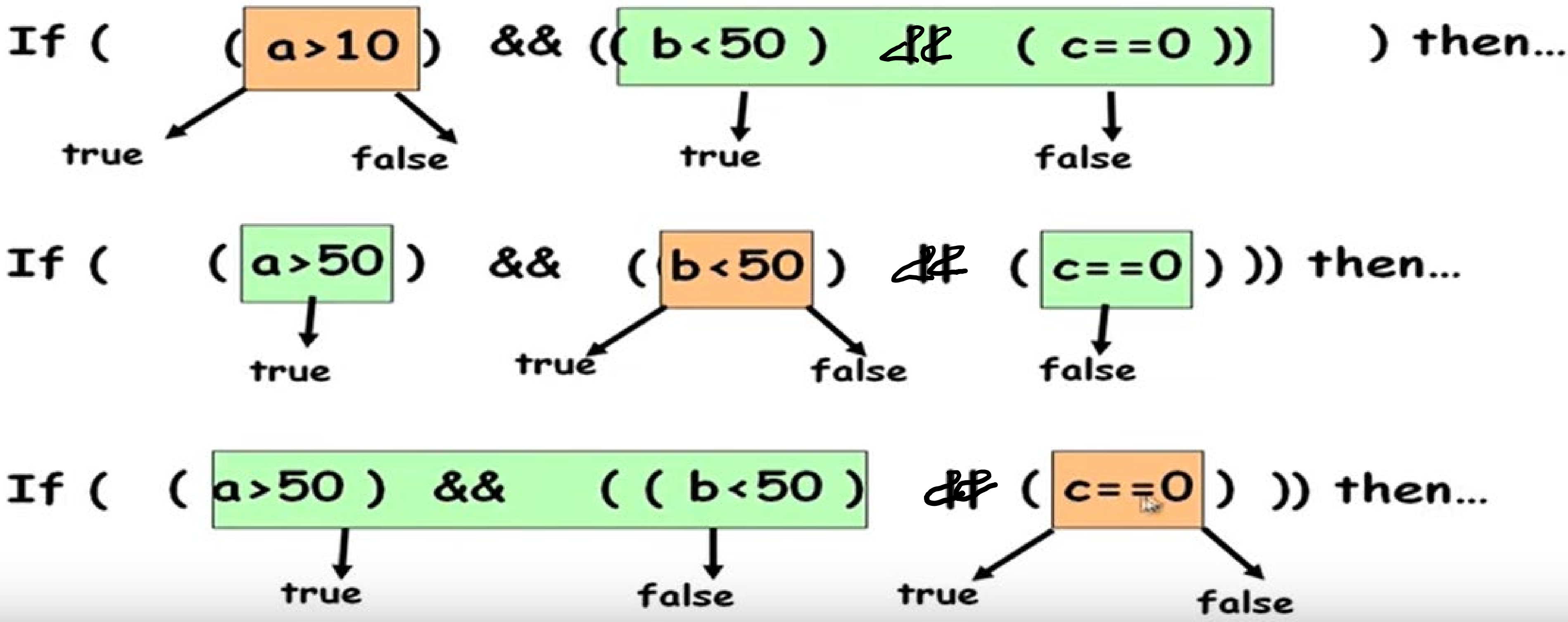
- **Req 1**: every decision in a program must take T/F values.



- **Req 2**: every atomic condition in each decision must take T/F values.



- **Req 3**: each condition in a decision should independently affect the decision's outcome. Other condition's value must hold constant.



If ((a>10) && ((b<50) && (c==0))) then...
true
false
true
false
If ((a>50) && (b<50) && (c==0))) then...
true
true
false
false
If ((a>50) && ((b<50) && (c==0))) then...
true
false
true
false

Creating MC/DC test cases

- Create truth table for conditions.
- Extend the truth table to represent test case pair that lead to show the independence influence of each condition.

Example : If (A and B) then . . .

Test Case Number	A	B	Decision	Test case pair for A	Test case pair for B
1	T	T	T	3	2
2	T	F	F		1
3	F	T	F	1	
4	F	F	F		

- Show independence of A
  - Take 1 + 3
- Show independence of B :
  - Take 1 + 2
- Resulting test cases ar
  - 1 + 2 + 3

Create MC/DC test cases for this program segment

```
If ( (( MyUserID = = ValidUserID) || (MySSN = = ValidSSN)) &&
(MyPassword = = ValidPassword) )
{
    return (ALLOW);
}
else
{
    return (DENY);
}
```

((U || S) & P)

	U	S	P	Outcome	U_T	S_T	P_T	MC/DC
1	0	0	0	0				
2	0	0	1	0	6	4		2
3	0	1	0	0			4	3
4	0	1	1	1		2	3	4
5	1	0	0	0			6	5
6	1	0	1	1	2		5	6
7	1	1	0	0			8	7
8	1	1	1	1			7	8

Show ind. of U = 6 + 2 P = 4+3 or 6 + 5 or
S = 4 + 2 8+7 ⇒ (6, 4, 2, 3/5)

University of Victoria

$((U \parallel S) \,\&\, P)$

	U	S	P	Outcome	U_T	S_T	P_T	MC/DC
1	0	0	0	0				
2	0	0	1	0	6	4		2
3	0	1	0	0			4	3
4	0	1	1	1		2	3	4
5	1	0	0	0			6	5
6	1	0	1	1	2		5	6
7	1	1	0	0			8	7
8	1	1	1	1			7	8

Min MC/DC $= \{6, 2, 4, 3\}$ or $\{6, 2, 4, 5\}$

Create MC/DC test cases for this program segment

```
If ( (( MyUserID = = ValidUserID) || (MySSN = = ValidSSN)) && (MyPassword
= = ValidPassword) )
{    return (ALLOW);}
else
{    return (DENY);}
```

100

Min MC/DC = {6, 2, 4, 3} or {6, 2, 4, 5}

Test Cases

	U	S	P
6	1	0	1
2	0	0	1
4	0	1	1
3	0	1	0

TC #1: (VU, IS, VP)

TC #2: (IU, IS, VP)

TC #3: (IU, VS, VP)

TC #4: (IU, VS, IP)

or

TC #4: (VU, IS, IP)

University of Victoria

$((a > 10) \&\& (b >= 20)) \,||\, (c == 5))$

Find min. MC/DC test cases for this condition.

((a>10) && (b>=20)) || (c == 5))

	a	b	c	Res.	at	bt	ct
1	0	0	0	0			2
2	0	0	1	1			1
3	0	1	0	0	7		4
4	0	1	1	1			3
5	1	0	0	0		7	6
6	1	0	1	1			5
7	1	1	0	1	3	5	
8	1	1	1	1			

University of Victoria

$((a > 10) \&\& (b >= 20)) || (c == 5))$

min MC/DC = {3, 4, 5, 7}

	a	b	c	Res.	at	bt	ct
1	0	0	0	0			2
2	0	0	1	1			1
3	0	1	0	0	7		4
4	0	1	1	1			3
5	1	0	0	0		7	6
6	1	0	1	1			5
7	1	1	0	1	3	5	
8	1	1	1	1			

Test cases are

	a	b	c
3	0	1	0
4	0	1	1
5	1	0	0
7	1	1	0

Test cases:

(5, 30, 4)

(5, 30, 5)

(12, 15, 4)

(12, 30, 4)

((a>10) && (b>=20)) || (c == 5))

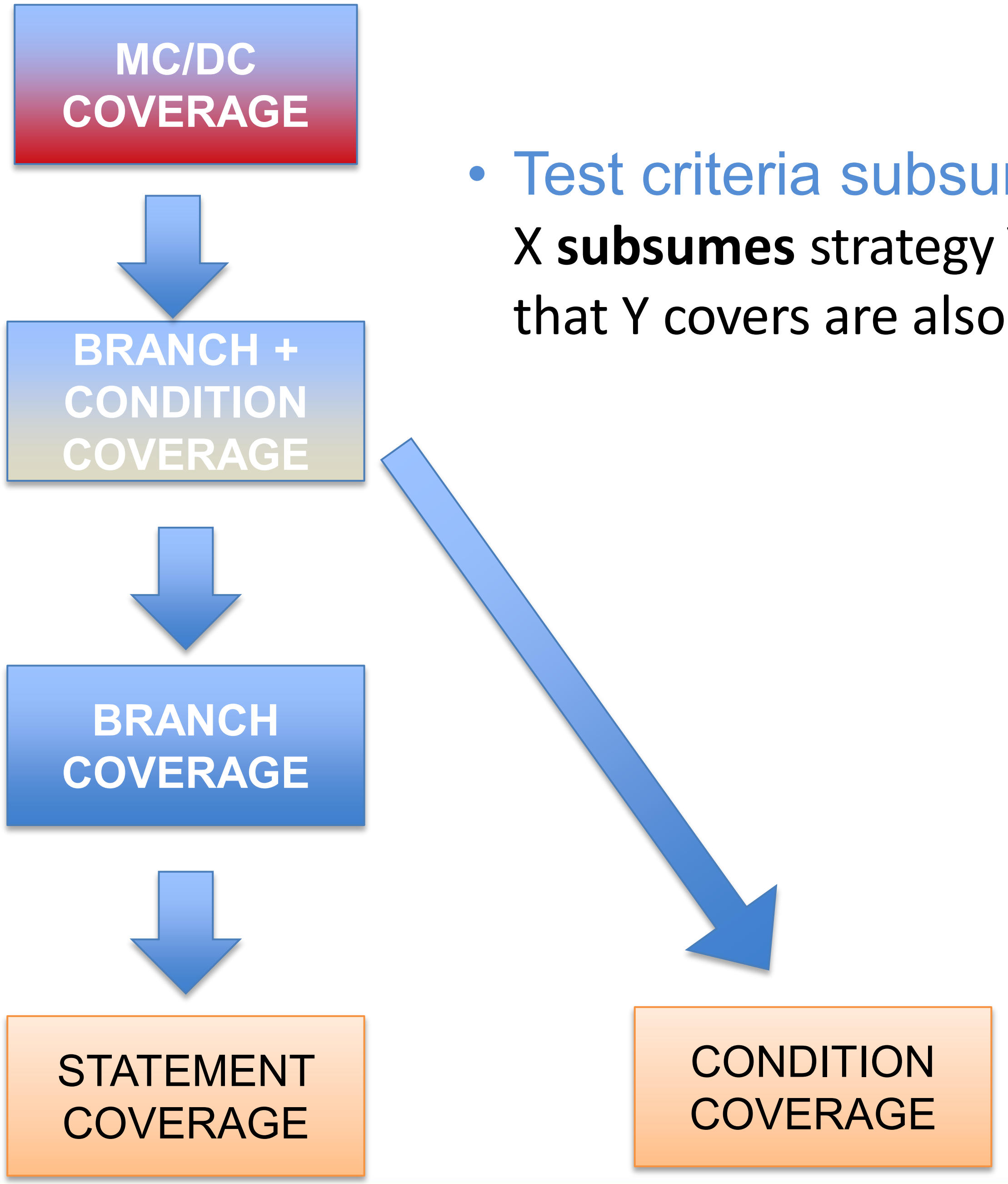
University of Victoria

It may not be possible to achieve MC/DC in some expressions

- Consider (A & B) II (A & $\overline{\mathrm{B}}$).
- While the independence pairs (TT, FT) would show the independence of A, there are no pairs that show the independence of B.

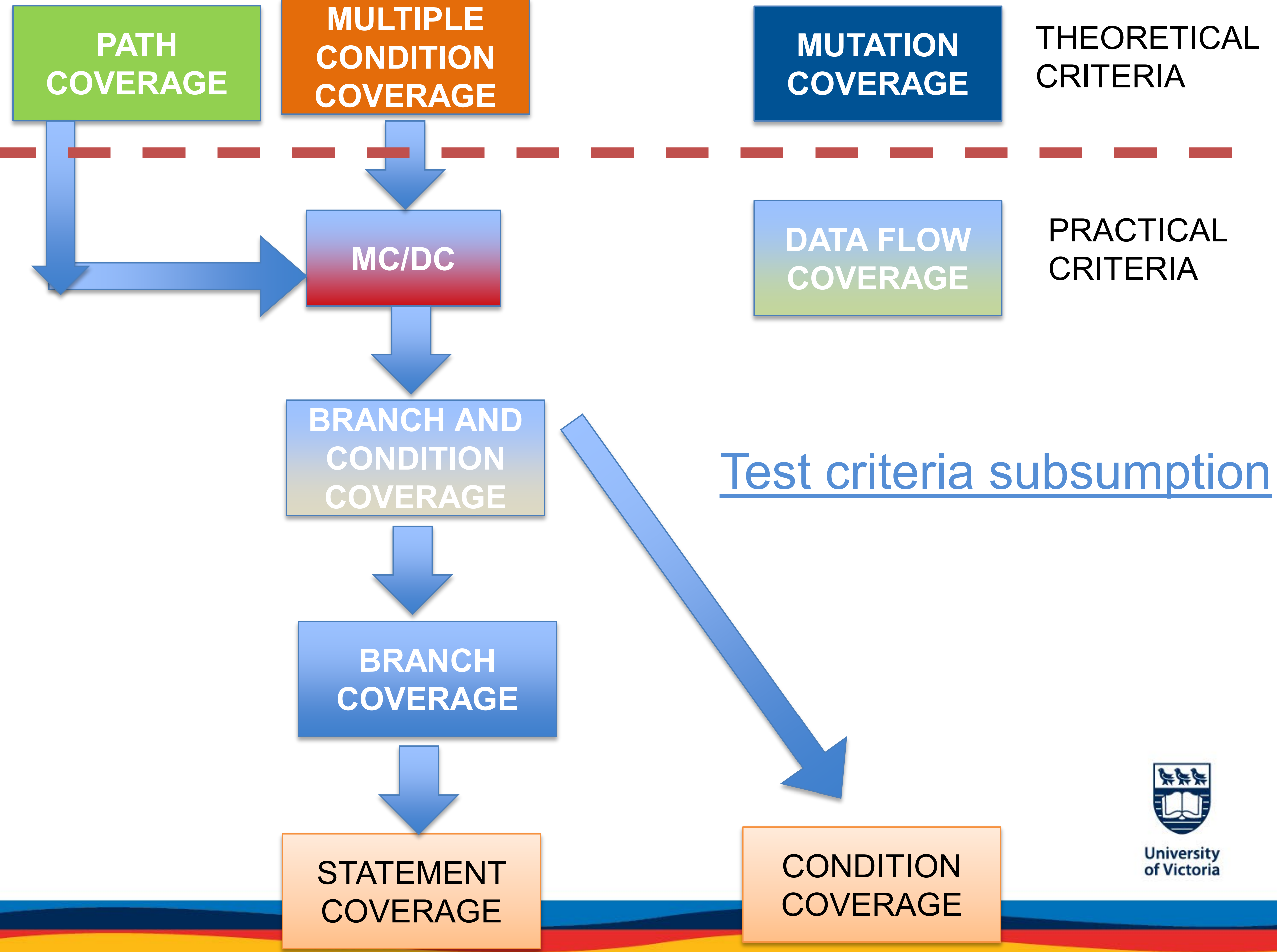
A	B	RESULT
T	T	F
T	F	F
F	T	T
F	F	T

- In such cases, revisit the expression, as it may have been poorly designed.
- In this example, the expression could be reformulated to simply A.



- Test criteria subsumption:Strategy X **subsumes** strategy Y if all elements that Y covers are also covered by X.

Test criteria subsumption



LOOP TESTING

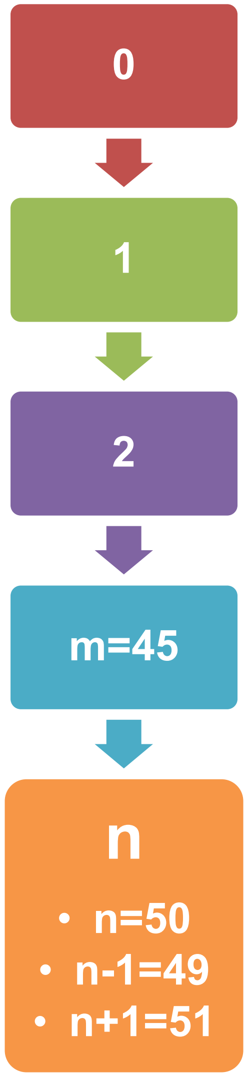
University of Victoria

Loop testing

- Loop Testing is a type of software testing type that is performed to validate the loops.
- Types of Loops:
  - –***Simple Loops*** are loops whose loop bodies contain no other loops.
  - –***Nested Loops*** are combinations of loops such that each is contained inside the loop body of the next.
  - –***Concatenated Loops*** are loops such that each follows the next in the code - that is, the execution of the next loop begins after the previous terminates.
  - –***Unstructured Loops*** are loops which are combinations of nested and concatenated loops.

Simple loops

```
// Program to print a text 5 times

class Main {
  public static void main(String[] args) {
    int n = 50;
    for (int i = 1; i <= n; ++i) {
      System.out.println("Java is fun");
    }
  }
}
```



Guidelines for Simple Loop Testing

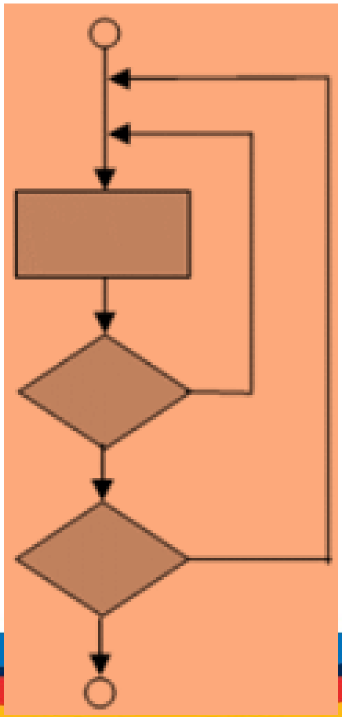
- Try to design a test in which the loop body **isn't executed at all**.
- Try to design a test in which the loop body is executed **exactly once**.
- Try to design a test in which the loop body is executed **exactly twice**.
- Design a test in which a loop body is executed some ``**typical**'' number of times.

If there is an **upper bound, *n***, on the number of times the loop body can be executed, then the following cases should also be applied.

- Design a test in which the loop body is executed exactly ***n*-1 times**.
- Design a test in which the loop body is executed exactly ***n* times**.
- Try to design a test causing the loop body to be executed exactly ***n*+1 times.**

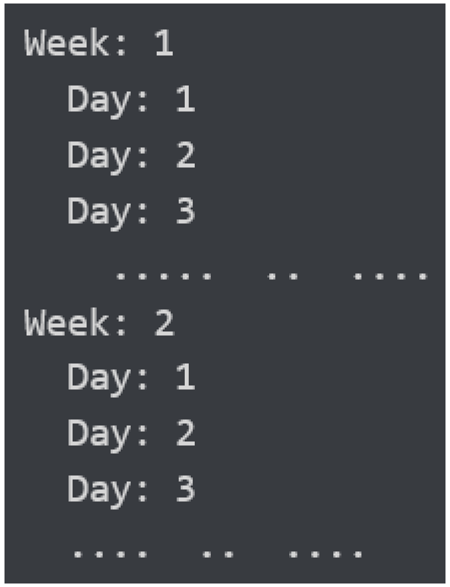
Guidelines for Nested Loops

1. Set all the other loops to minimum value and start at the innermost loop
2. For the innermost loop, perform a simple loop test and hold the outer loops at their minimum iteration parameter value
3. Perform test for the next loop and work outward.
4. Continue until the outermost loop has been tested.



Nested loop

```
class Main {
  public static void main(String[] args) {

    int weeks = 24;
    int days = 7;

    // outer loop prints weeks
    for (int i = 1; i <= weeks; ++i) {
        System.out.println("Week: " + i);

        // inner loop prints days
        for (int j = 1; j <= days; ++j) {
                 System.out.println("  Day: " + j);
         }
    }
  }
}
```



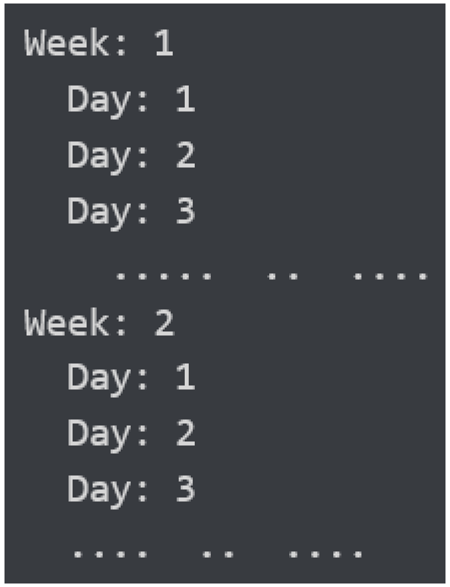
```
Week: 1
   Day: 1
   Day: 2
   Day: 3
      .....  ..  ....
Week: 2
   Day: 1
   Day: 2
   Day: 3
   ....  ..  ....
```

Testing:

1. Outer 'for loop' runs 1 time.
2. Inner 'for loop' runs: 0,1,2,4,7,8,6 times.
3. Outer 'for loop' runs: 0,1,2,20,24,25,23 times

Nested loop, check for Weeks=25

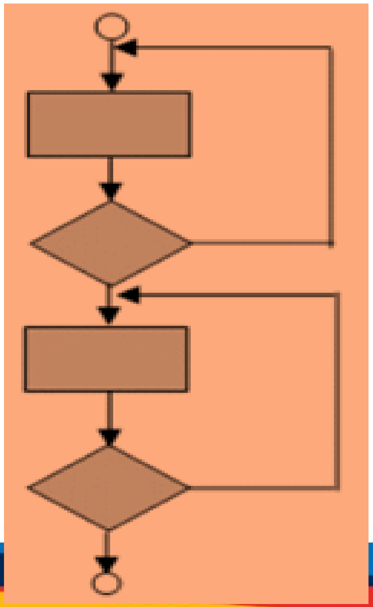
```
class Main {
 public static void main(String[] args) {

   int weeks = 25;
   int days = 7;

   // outer loop prints weeks
   for (int i = 1; i <= weeks; ++i) {
    if(weeks>24)
      {
         System.out.println(“A Week value more than 24 is not allowed”);
         break;
      }
    System.out.println("Week: " + i);

      // inner loop prints days
      for (int j = 1; j <= days; ++j) {
              System.out.println("  Day: " + j);
   }  }  }}
```



```
Week: 1
  Day: 1
  Day: 2
  Day: 3
     .....  ..  ....
Week: 2
  Day: 1
  Day: 2
  Day: 3
   ....  ..  ....
```

Guidelines for Concatenated Loops

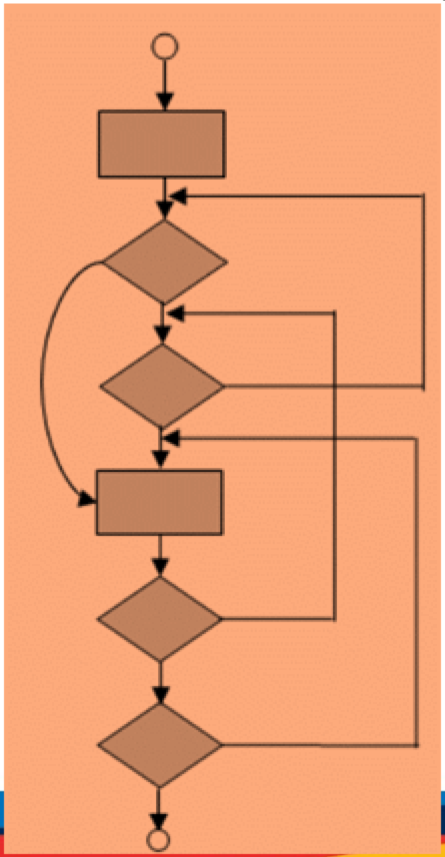
- **The difference between nested and** concatenated loops is that in nested loops, the loop is inside the loop, but in concatenated loops, the loop is after the loop.
- If two loops are **independent** of one other, they are tested as **simple** loops.
- However, if one loop's loop counter is utilized as the beginning value for the others, the loops are **not** considered **independent**. So, use **nested** loop testing approach.



Unstructured Loops

- The combination of nested and concatenated loops is known as an unstructured loop.
- It is necessary to **restructure** the architecture of unstructured loops to reflect the use of structured programming techniques.

```
while(){
   for()
   {}
   while()
   {}
}
```



University of Victoria

Is 100% Coverage *Feasible*?

- Mutually exclusive conditions
  - (a < 0 && a < 10)
    - (**T && F)** is not feasible
- Dead code / unreachable code
- “This should never happen” code

Systems in practice:
Statement coverage 85-90%
feasible

Coverage: Useful or Harmful?

- Measuring coverage (% of satisfied test obligations) can be a useful indicator ...
  - Of progress toward a thorough test suite, of trouble spots requiring more attention
- ... or a dangerous seduction
  - Coverage is only a proxy for thoroughness or adequacy
  - It’s easy to improve coverage without improving a test suite (much easier than designing good test cases)
- The only measure that really matters is **effectiveness**