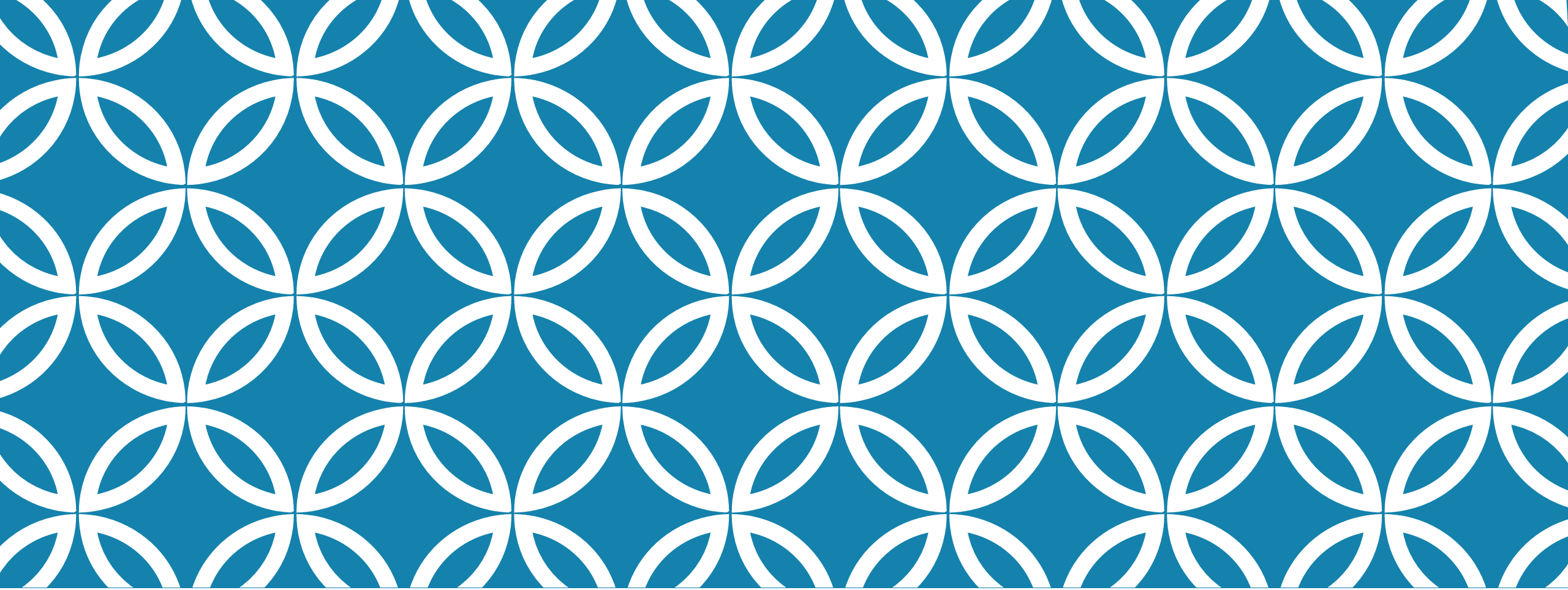


WHITEBOX TESTING

CS-HU 374 Lecture 5

TEST ADEQUACY CRITERIA

A test selection criterion defines “coverage” items that a test suite T should include.

Two families of test adequacy criteria

- For Blackbox “coverage” are tests inputs derived from category partition and boundary approaches.
- For Whitebox “coverage” are the execution of structural elements of the program code
  - Statements
  - Branches
  - MC/DC (modified condition/decision coverage) – required in aeronautics industry
  - Execution paths

Test adequacy measurement :

$$\frac{items\ covered\ by\ T}{total\ items}$$

WHITEBOX TESTING

Selection of the inputs is based on whether they execute some elements of the code

Premise: execution of a faulty element is a necessary condition for revealing a fault

Commonly used structural elements

- Statements (default meaning of test coverage)
- Branches
- Conditions

CONTROL FLOW GRAPH (CFG)

CFG of a program is a directed graph where nodes are program statements and edges represent the flow of control in the program, i.e., what statements could be executed next.

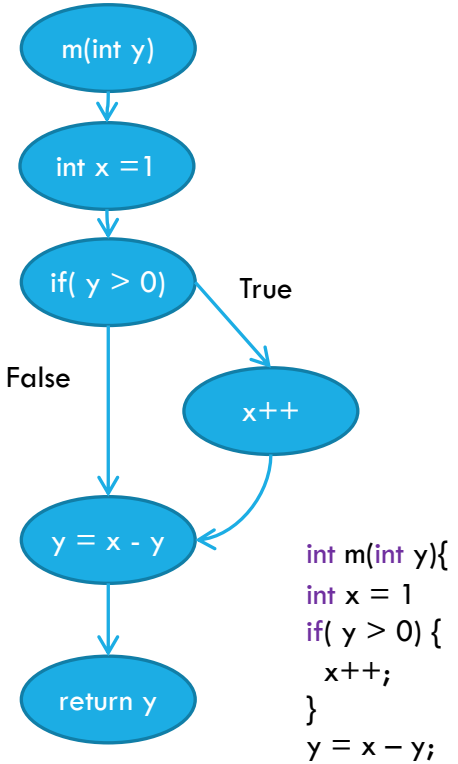
Has one entry and one exit nodes.

Conditional statements have two outgoing edges:

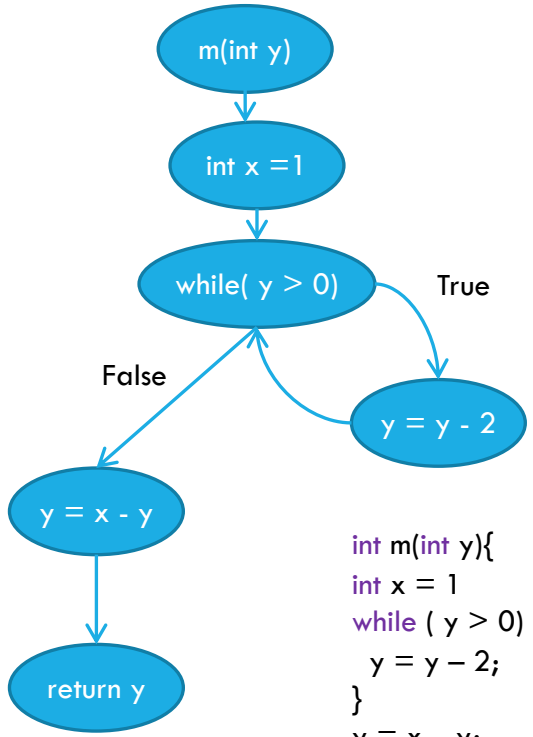
- True branch
- False branch

Statement after a conditional statement has two incoming edges.

For and While loops create cycles in the graph.




```
int m(int y){
int x = 1
if( y > 0) {
  x++;
}
y = x – y;
return y;
}
```




```
int m(int y){
int x = 1
while ( y > 0) {
  y = y – 2;
}
y = x – y;
return y;
}
```

CFG IN-CLASS EXERCISE

Draw CFG on the following methods

```
public int m1(int x, int y) {
    int z = y%x;
    if(z < 5) {
        while ( z != 5) {
            y++;
            x--;
            z = y%x;
        }
    }

    return x;
}
```

```
public void m2(boolean c, int j) {
    int z = j + 24;
    while ( z > 0) {
        if (c) {
            j --;
            c = !c;
        } else {
            j++;
            z = z + j;
        }
    }
}
```

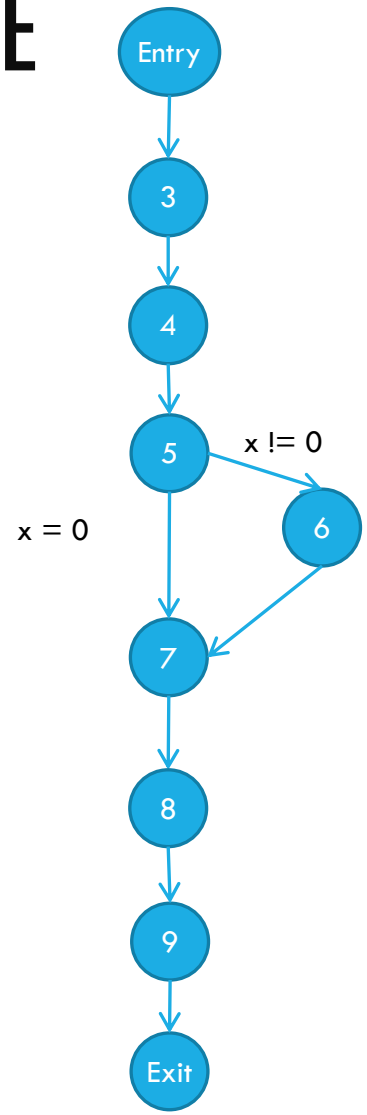
STATEMENT COVERAGE

Test requirement, i.e., the adequacy criterion is the number of executed statements in the program

$$C_{stmt} = \frac{number\ of\ executed\ statement}{number\ of\ all\ statements}$$

C_{stmt} EXAMPLE

```
1: void main(){
2:    int x, y;
3:    read(x);
4:    read(y);
5:    if(x != 0){
6:      x = x + 10;}
7:    y = y/x;
8:    write(x);
9:    write (y);
}
```



Identify test inputs for statement coverage

- Test requirement – nodes 3, …, 9
- A test that executes all statements should have x!=0 and any value of y
  - t: (x= 20, y =30)
- t executes all statements
- t does not reveal fault in line 7

To reveal the division by zero fault we need to execute edge 5 → 7

Such requirement is defined by the branch coverage C_{branch} adequacy criterion

BRANCH COVERAGE

Test requirement, i.e., the adequacy criterion is the number of executed branches in the program

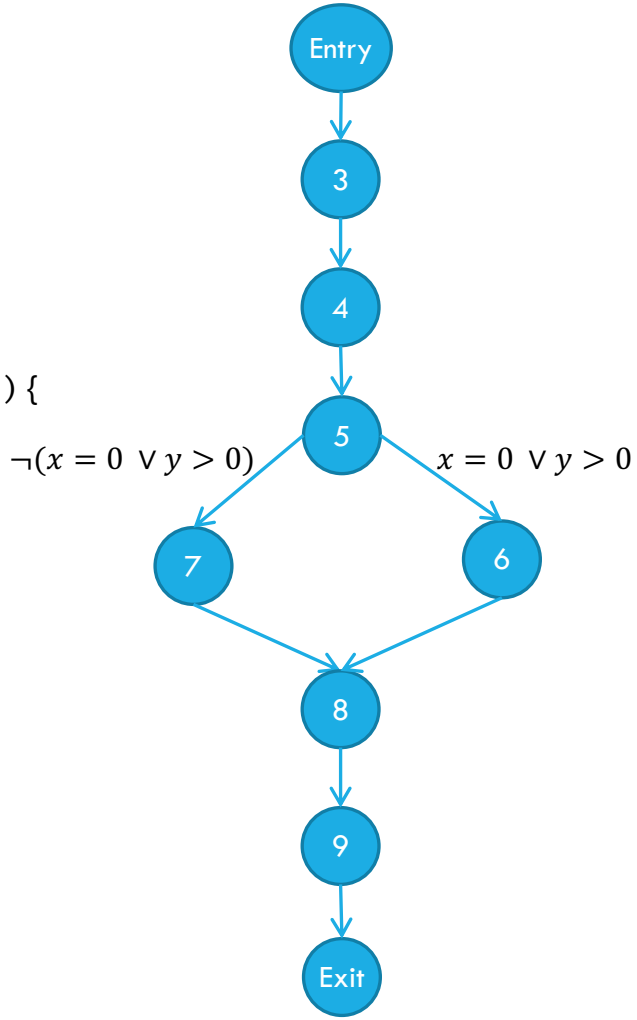
$$C_{branch} = \frac{number\ of\ executed\ branches}{number\ of\ all\ branches}$$

Each conditional statement must be evaluated to true and to false

Number of branches = (number of conditional statements + number of loops) x 2

C_{branch} EXAMPLE

```
1: void main(){
2:    int x, y;
3:    read(x);
4:    read(y);
5:    if(x == 0 || y > 0){
6:       y = y/x;
       } else {
7:    x = y + 2;}
8:    write(x);
9:    write (y);
}
```



Identify test inputs for branch coverage

Test requirement:

- edge 5 → 6 and 5 → 7
- Consider test inputs T
  - x = 5, y = 5
  - x = 5, y = -5
- T covers all the edges but does not reveal the fault in line 6
- Each condition in line 5 could be true or false
- Such requirement is defined by the condition coverage C_{cond} adequacy criterion

BASIC CONDITION COVERAGE

Test requirement, i.e., the adequacy criterion is the number truth values assumed by basic conditions

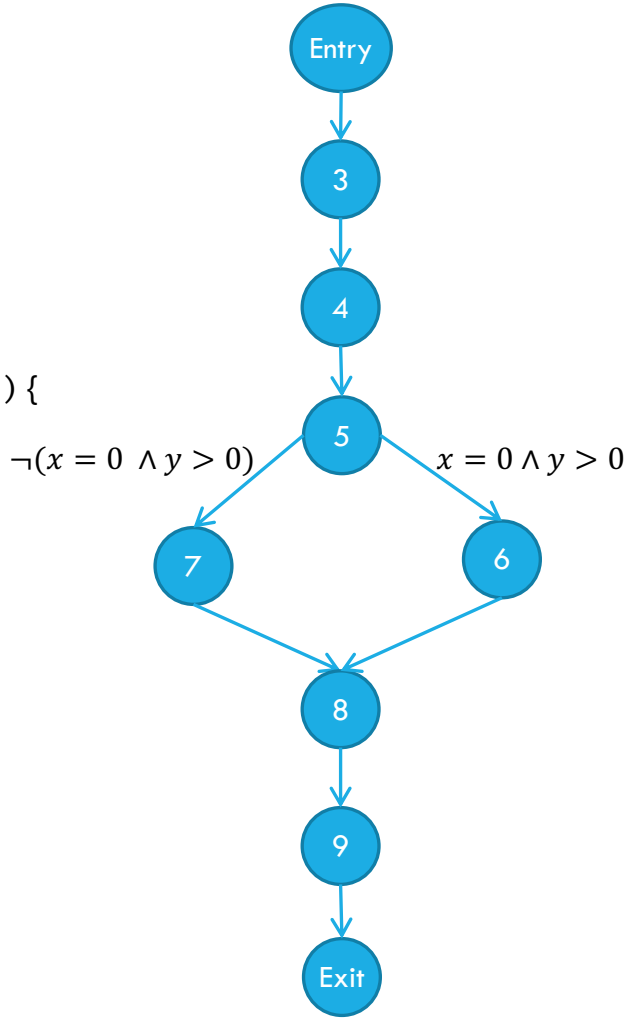
$$C_{cond} = \frac{number\ of\ executed\ conditions}{number\ of\ all\ conditions}$$

Each condition in a compound conditional statement must be evaluated to true and to false

Number of conditions = (number of all basic conditions) x 2

C_{cond} EXAMPLE

```
1: void main(){
2:    int x, y;
3:    read(x);
4:    read(y);
5:    if(x == 0 && y > 0){
6:       y = y/x;
      } else {
7:    x = y + 2;}
8:    write(x);
9:    write (y);
}
```



Identify test inputs for basic condition coverage

Test requirement:

- $x = 0, x \neq 0, y > 0, y \leq 0$
- Test case
  - t1: x = 0, y = -5 (covers $x = 0$ and $y \leq 0$)
  - t2: x = 5, y = 5 (covers $x \neq 0$ and $y > 0$)
- Test suite is not adequate for branch (and statement)
- Branch and condition coverage $C_{cond} + C_{banch}$

PATH COVERAGE C_{path}

Test requirement: all paths trough program's CFG

Combination of branch coverages with a single condition

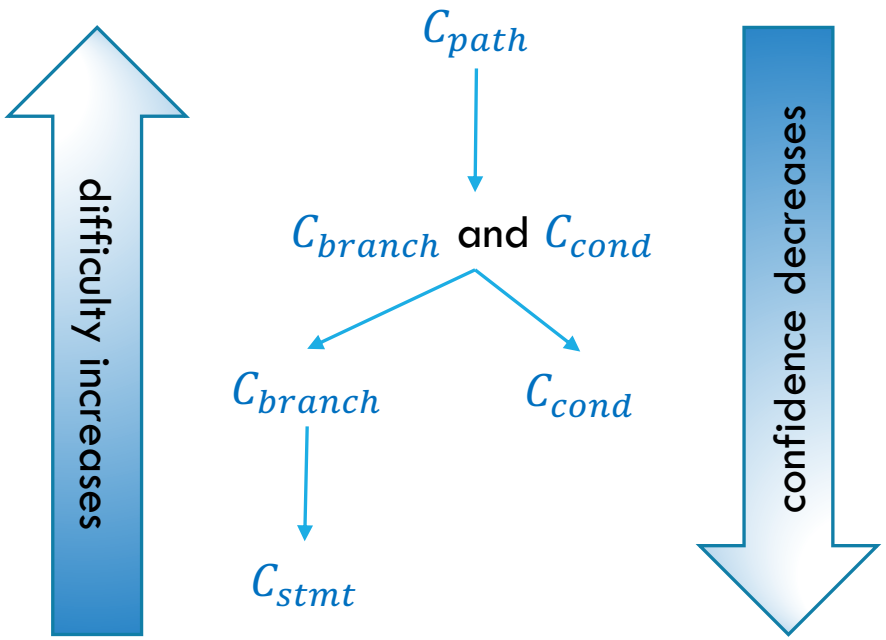
```
1: void main(){
2:   int x, y;
3:   read(x);
4:   read(y);
5:   if(x < 0){
6:     x++;}
7:   if(y > 0){
8:     y--;}
}
```

- Branch on line 5: b5 = T, b5 = F
- Branch on line 7: b7 = T, b7 = F
- Adequate inputs that cover:
  - Path1: b5 = T, b7 = T
  - Path2: b5 = T, b7 = F
  - Path3: b5 = F, b7 = T
  - Path4: b5 = F, b7 = F
- Inputs
  - Path1: x = -1, y = 1;
  - Path2: x = -1, y = -1
  - Path3: x = 1, y = 1
  - Path4: x = 1, y = -1

Difficult to define in the presence of loops

Usually set to a fixed number of loop iterations (one or two)

STRUCTURAL COVERAGE CRITERIA



Arrows indicate the relations between criteria:

- If a test suite T is C_{branch} adequate (covers all branches) then it is also C_{stmt} adequate (covers all statements)
- C_{stmt} and C_{cond} have no relations, that is the coverage of one cannot guarantee the coverage of another

The confidence of a C_{stmt} adequate test suite is lower than the confidence of C_{branch} (C_{branch} is stronger than C_{stmt})

It is more difficult to generate C_{branch} adequate test inputs than C_{stmt} ones

The size of a C_{branch} adequate test suite is usually larger than the size of a C_{stmt} adequate one, but can be exactly the same.

ADEQUACY IN-CLASS EXERCISE

```
public void m3(boolean b, int x, int y) {
    if (b) {
        if (x > y) {
            y = y + 10;
        } else {
            y = y - 10;
        }
    } else {
        if (x > y && y != 0) {
            y = y + 1;
        } else {
            y = y - 1;
        }
    }
}
```

How many statement should be covered?

How many branches should be covered?

How many conditions should be covered?

Consider the following set of inputs:

- (true, -1, 1), (false, 1, 1), (false, -1, 0)

How is it adequate (in terms of %) for each adequacy criteria?

Add test cases that increase the adequacy of test suite for each criterion.

NEXT TIME

Using EclEmma – an Eclipse plug-in coverage tool (you might already have it installed – if not see separate instructions)

Evaluating test suites in terms of their coverage.

Using coverage report to derive statement and branch adequate test suites.

Quiz 3 (the last one!) is due by the beginning of the next class