CS 3300
Intro to Software Engineering

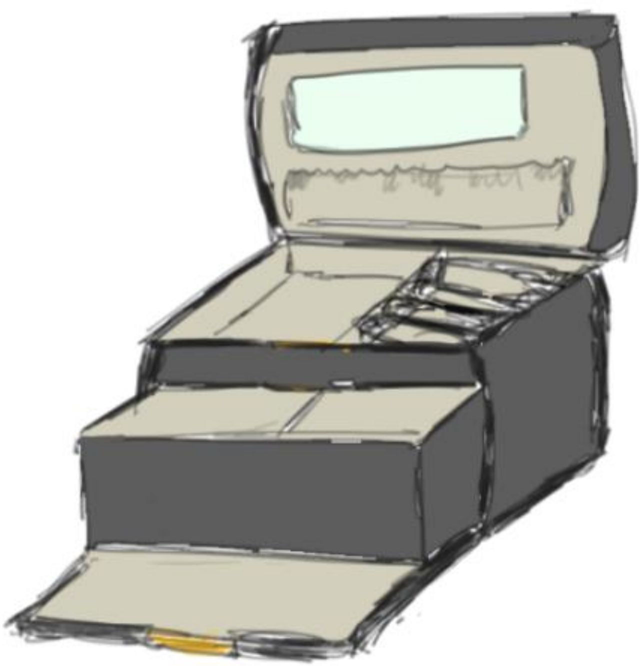
SOFTWARE TESTING

WHITE-BOX TESTING

Mahdi Roozbahani

Slides are based on Alex Orso.

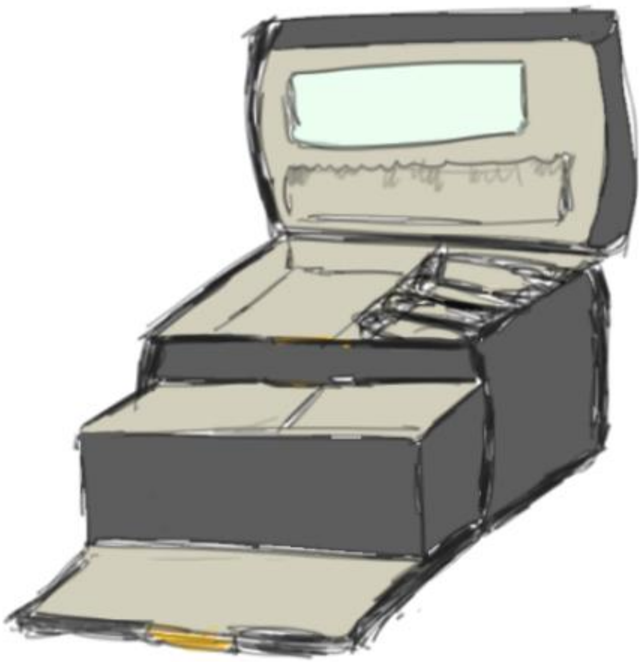
WHITE-BOX TESTING



Basic assumption

Executing the faulty statement is a necessary condition for revealing a fault

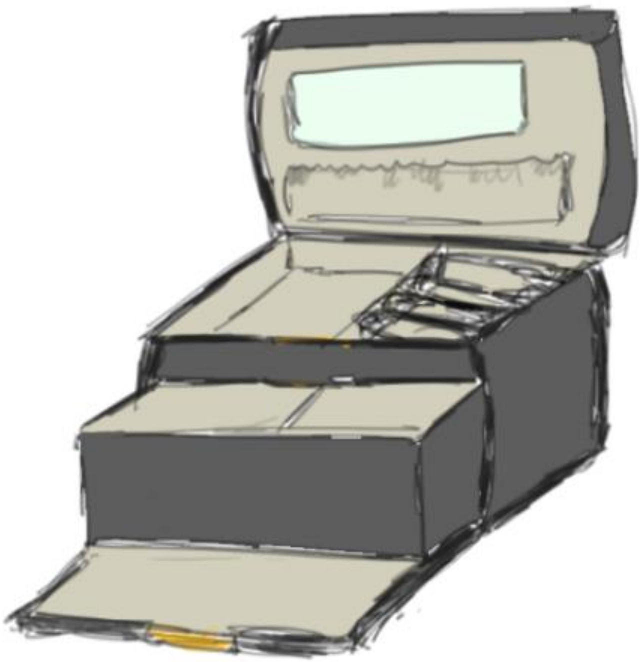
WHITE-BOX TESTING



Advantages

- Based on the code
  - ⇒ can be measured objectively
  - ⇒ can be measured automatically
- Can be used to compare test suites
- Allows for covering the coded behavior

WHITE-BOX TESTING



Different kinds:

- control-flow based
- data-flow based
- fault based

Let's go back to `printSum`

```
printSum(int a, int b)
```

Let's go back to `printSum`

```
printSum(int a, int b) {
   int result = a + b;
   if (result > 0)
     printcol("red", result);
   else if (result < 0)
     printcol("blue", result);
}
```

Test this case

and this one

COVERAGE CRITERIA

Defined in terms of

test requirements

Result in

test specifications

test cases

printSum: test requirements

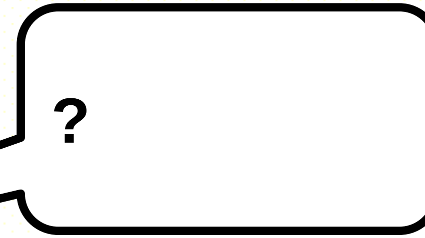
```
printSum(int a, int b) {
   int result = a + b;
   if (result > 0)
     printcol("red", result);
   else if (result < 0)
     printcol("blue", result);
}
```

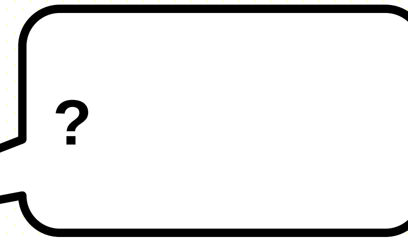
req #1

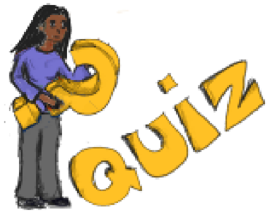
req #2

printSum: test specifications

```
printSum(int a, int b) {
   int result = a + b;
   if (result > 0)
      printcol(“red”, result);
   else if (result < 0)
      printcol(“blue”, result);
}
```







printSum: test cases

```
printSum(int a, int b) {
   int result = a + b;
   if (result > 0)
      printcol("red", result);
   else if (result < 0)
      printcol("blue", result);
}
```

Test Spec #1
a + b > 0

Test Spec #2
a + b < 0

#1 ((a=[ ], b=[ ]),(outputcolor=[ ], outputvalue=[ ]))
#2 ((a=[ ], b=[ ]),(outputcolor=[ ], outputvalue=[ ]))

printSum: test cases

```
printSum(int a, int b) {
  int result = a + b;
  if (result > 0)
    printcol(“red”, result);
  else if (result < 0)
    printcol(“blue”, result);
}
```

a == 3
b == 9

a == -5
b == -8

STATEMENT COVERAGE

Test requirements

Coverage measure

STATEMENT COVERAGE

Test requirements — statements in the program

Coverage measure

STATEMENT COVERAGE

Test requirements statements in the program

Coverage measure

$$\frac{\text{number of executed statements}}{\text{total number of statements}}$$

printSum: statement coverage

```
printSum(int a, int b) {
  int result = a + b;
  if (result > 0)
    printcol("red", result);
  else if (result < 0)
    printcol("blue", result);
}
```

`printSum`: statement coverage

```
a == 3
b == 9
```

```
printSum(int a, int b) {
  int result = a + b;
  if (result > 0)
    printcol("red", result);
  else if (result < 0)
    printcol("blue", result);
}
```

Coverage: 0%

printSum: statement coverage

a == 3
b == 9

```
printSum(int a, int b) {
   int result = a + b;
   if (result > 0)
      printcol("red", result);
   else if (result < 0)
      printcol("blue", result);
}
```

Coverage: 71%

`printSum`: statement coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {
  int result = a + b;
  if (result > 0)
    printcol("red", result);
  else if (result < 0)
    printcol("blue", result);
}
```

Coverage: 71%

`printSum`: statement coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {
   int result = a + b;
   if (result > 0)
      printcol(“red”, result);
   else if (result < 0)
      printcol(“blue”, result);
}
```

Coverage: 100%

STATEMENT COVERAGE IN PRACTICE

Most used in industry

"Typical coverage" target is 80-90%

Why don't we aim at 100%

[ ]

`printSum`: statement coverage

a == 3 b == 9	a == -5 b == -8

```
printSum(int a, int b) {
   int result = a + b;
   if (result > 0)
     printcol(“red”, result);
   else if (result < 0)
     printcol(“blue”, result);
}
```

Coverage: 100%

`printSum`: statement coverage

a == 3 b == 9	a == -5 b == -8

```
printSum(int a, int b) {
  int result = a + b;
  if (result > 0)
    printcol("red", result);
  else if (result < 0)
    printcol("blue", result);
  [else do nothing]
}
```

Coverage: 100%

`printSum`: statement coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {
   int result = a + b;
   if (result > 0)
      printcol("red", result);
   else if (result < 0)
      printcol("blue", result);
   [else do nothing]
}
```

Coverage: 100%

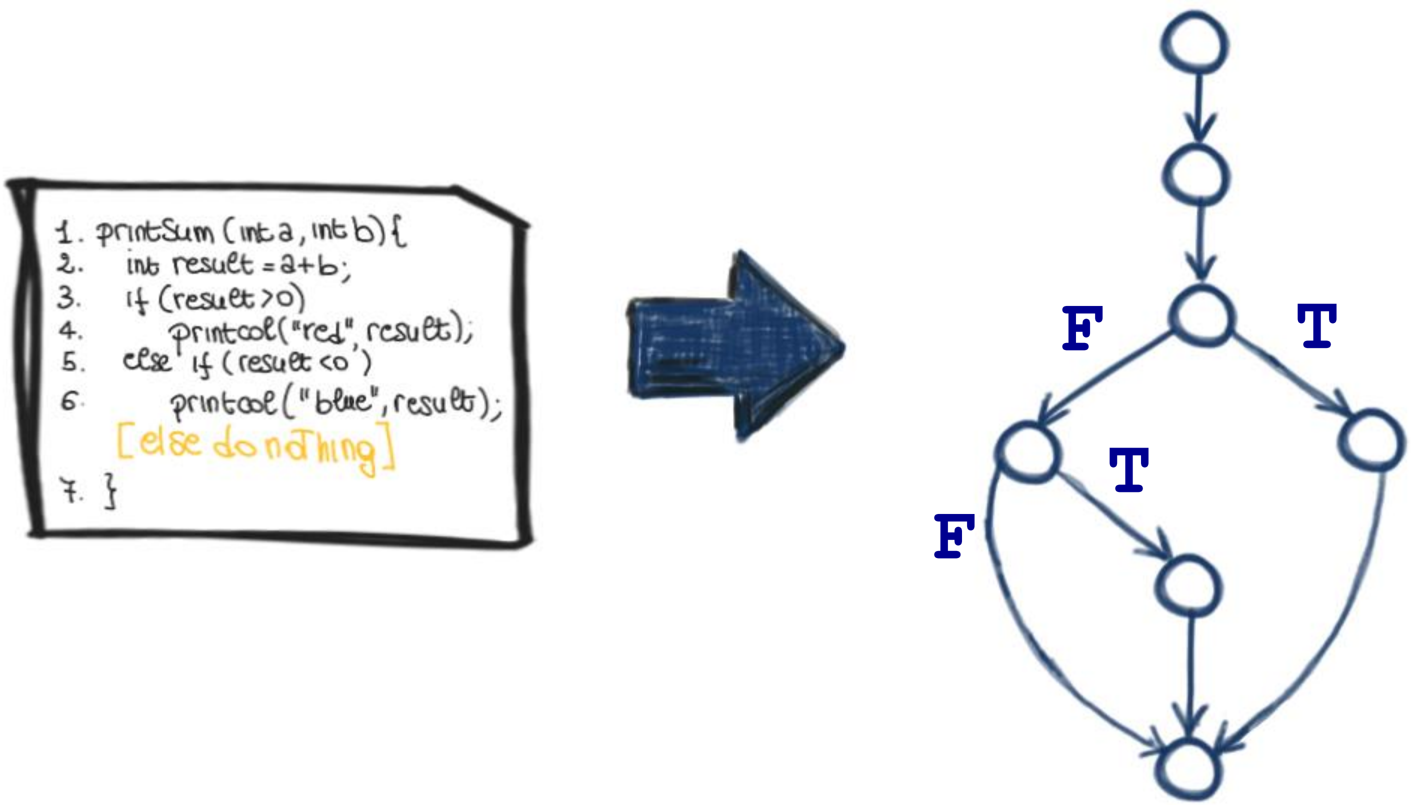
`printSum`: statement coverage

a == 3	a == -5
b == 9	b == -8

```
printSum(int a, int b) {
  int result = a + b;
  if (result > 0)
    printcol("red", result);
  else if (result < 0)
    printcol("blue", result);
  [else do nothing]
}
```

Coverage: 100%

DIGRESSION: CONTROL FLOW GRAPHS



BRANCH COVERAGE

Test requirements

Coverage measure

BRANCH COVERAGE

Test requirements

branches in the program

Coverage measure

BRANCH COVERAGE

Test requirements — branches in the program

Coverage measure — $\frac{\text{number of executed branches}}{\text{total number of branches}}$

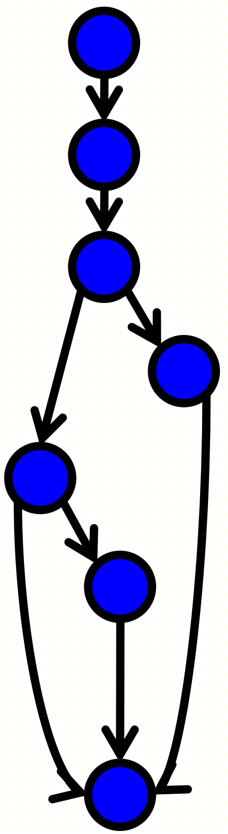
`printSum`: branch coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {
  int result = a + b;
  if (result > 0)
    printcol("red", result);
  else if (result < 0)
    printcol("blue", result);
  [else do nothing]}

```



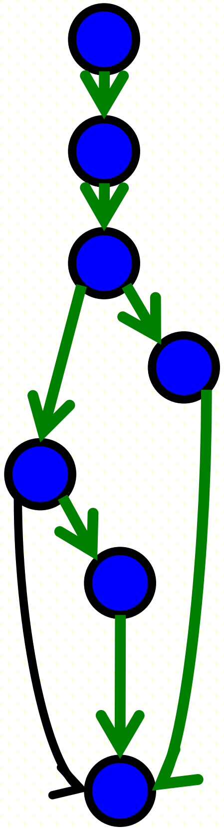
Coverage: ?

`printSum`: branch coverage

a == 3
b == 9

a == -5
b == -8

```
printSum(int a, int b) {
   int result = a + b;
   if (result > 0)
      printcol("red", result);
   else if (result < 0)
      printcol("blue", result);
  [else do nothing]}
```

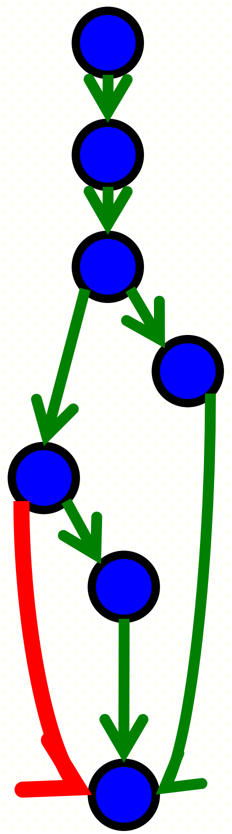


Coverage: 75%

`printSum`: branch coverage

a == 3	a == -5	a == 0
b == 9	b == -8	b == 0

```
printSum(int a, int b) {
  int result = a + b;
  if (result > 0)
    printcol("red", result);
  else if (result < 0)
    printcol("blue", result);
  [else do nothing]}
```



Coverage: 75%

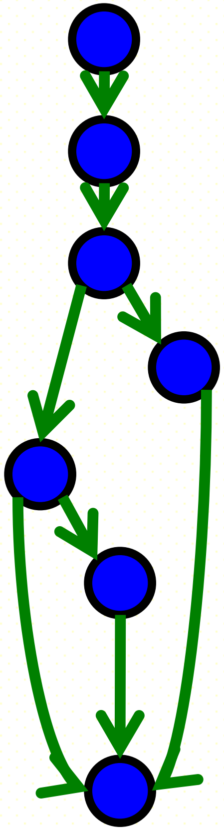
printSum: branch coverage

a == 3
b == 9

a == -5
b == -8

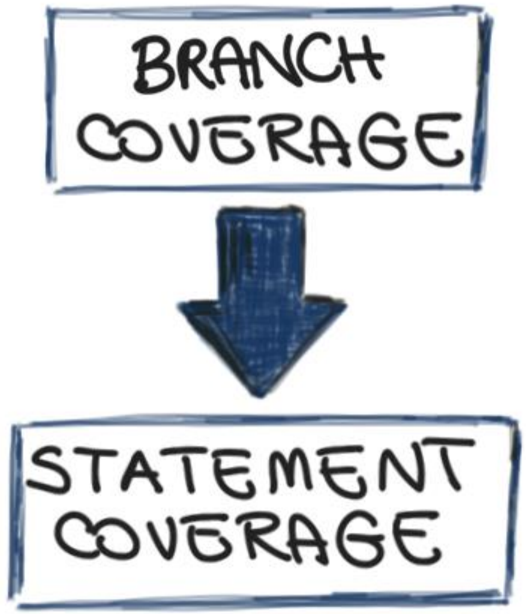
a == 0
b == 0

```
printSum(int a, int b) {
  int result = a + b;
  if (result > 0)
    printcol("red", result);
  else if (result < 0)
    printcol("blue", result);
 [else do nothing]}
```



Coverage: 100%

TEST CRITERIA SUBSUMPTION

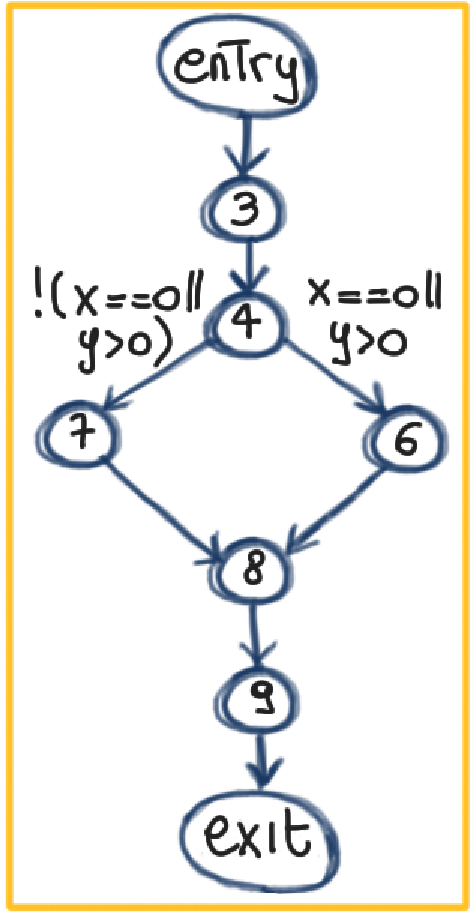


LET'S CONSIDER ANOTHER EXAMPLE

```
void main(){
  float x,y;
  read(x);
  read(y);
  if((x==0)||(y>0))
     y=y/x;
  else x=y+2;
  write(x);
  write(y);
}
```

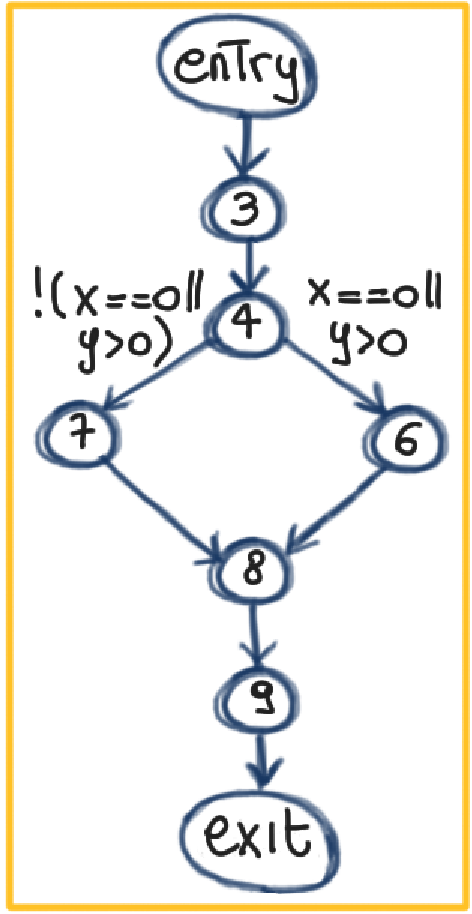
LET'S CONSIDER ANOTHER EXAMPLE

```
void main(){
    float x,y;
    read(x);
    read(y);
    if((x==0)||(y>0))
        y=y/x;
    else x=y+2;
    write(x);
    write(y);
}
```



LET'S CONSIDER ANOTHER EXAMPLE

```
1. void main(){
2.    float x,y;
3.    read(x);
4.    read(y);
5.    if((x==0)||(y>0))
6.        y=y/x;
7.    else  x=y+2;
8.    write(x);
9.    write(y);
10. }
```

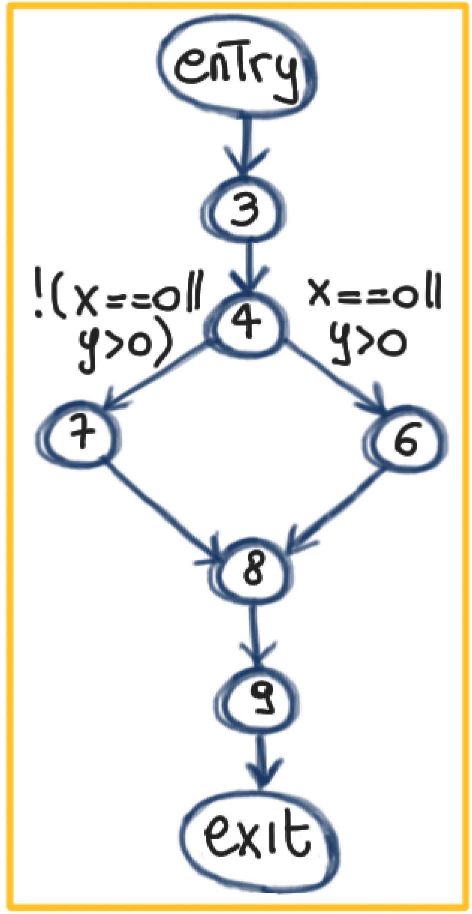


Tests: (x=5, y=5)
(x=5, y=-5)

Brench coverage:

LET'S CONSIDER ANOTHER EXAMPLE

```
void main(){
   float x,y;
   read(x);
   read(y);
   if((x==0)||(y>0))
      y = y/x;
   else x = y+2;
   write(x);
   write(y);
}
```

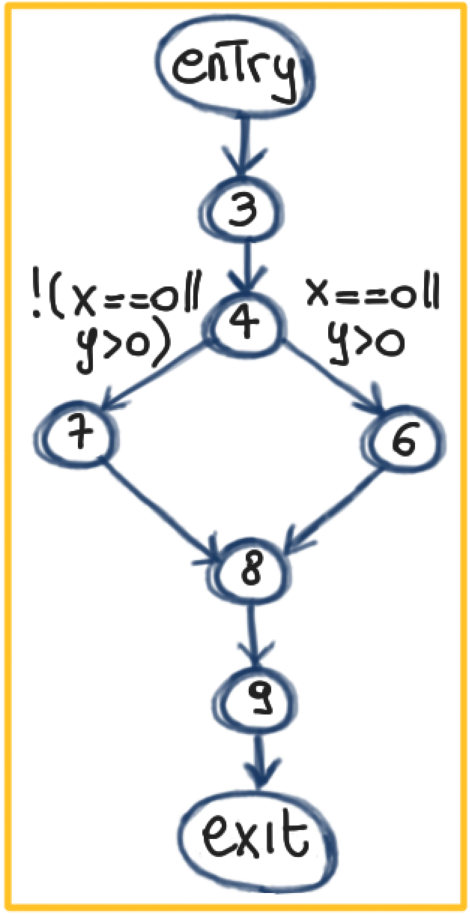


Tests: (x=5, y=5)
(x=5, y=-5)

Brench coverage: 100%

LET'S CONSIDER ANOTHER EXAMPLE

```
void main(){
    float x,y;
    read(x);
    read(y);
    if((x==0)||(y>0))
        y=y/x;
    else x=y+2;
    write(x);
    write(y);
}
```



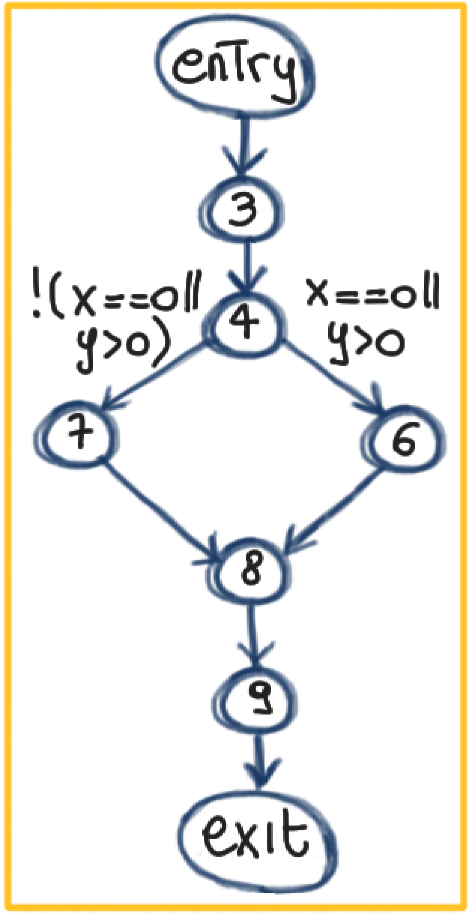
Tests: (x=5, y=5)
(x=5, y=-5)

Brench coverage: 100%

How can we be more thorough?

LET'S CONSIDER ANOTHER EXAMPLE

```
void main(){
   float x,y;
   read(x);
   read(y);
   if((x==0)||(y>0))
      y=y/x;
   else  x=y+2;
   write(x);
   write(y);
}
```



Tests: (x=5, y=5)
(x=5, y=-5)

Brench coverage: 100%

How can we be more thorough?

We can make each condition T and F

CONDITION COVERAGE

Test requirements

Coverage measure

CONDITION COVERAGE

Test requirements

individual conditions in the program

Coverage measure

CONDITION COVERAGE

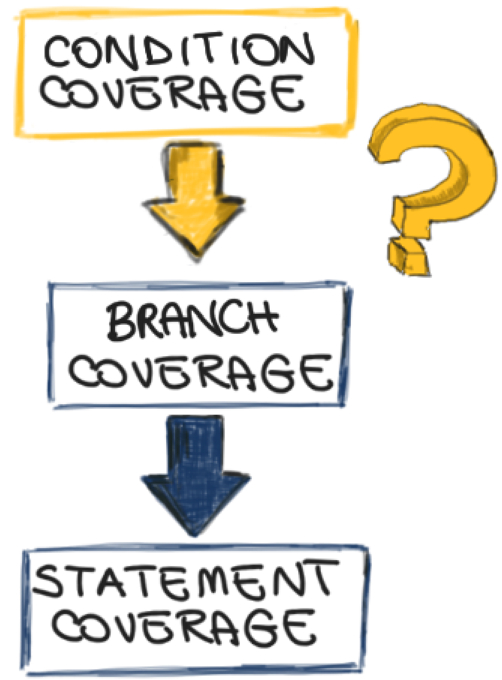
Test requirements	individual conditions in the program
Coverage measure	$\frac{\text{number of conditions that are both T and F}}{\text{total number of conditions}}$

SUBSUMPTION

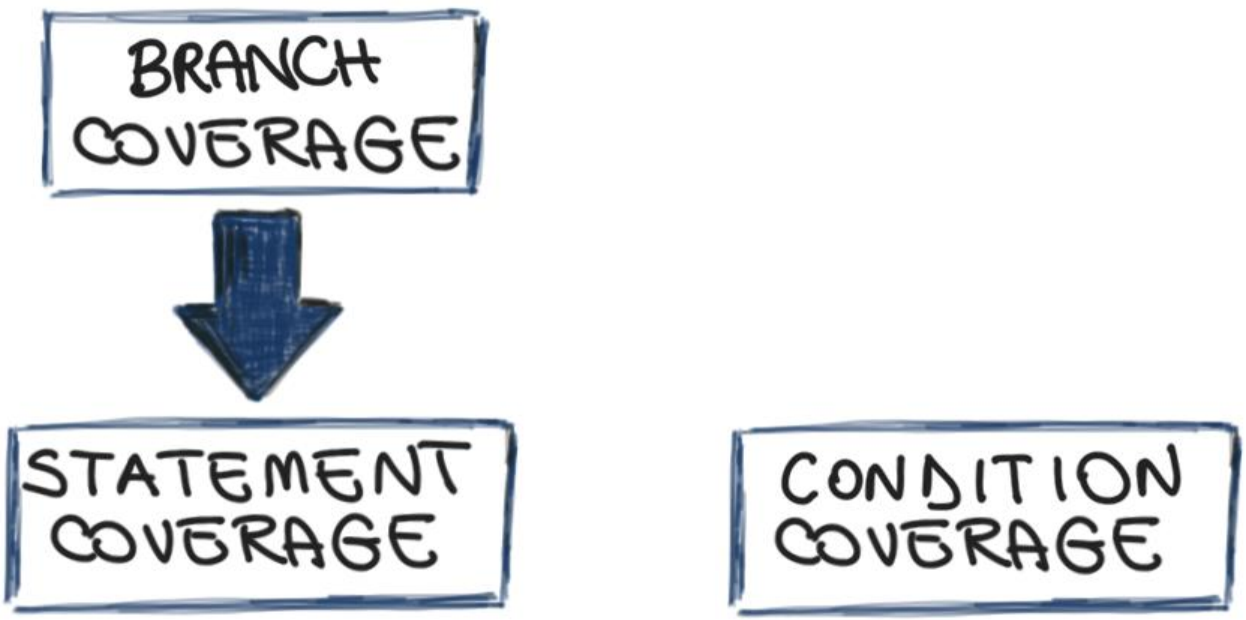
Does condition coverage imply branch coverage?

[ ] Yes

[ ] No

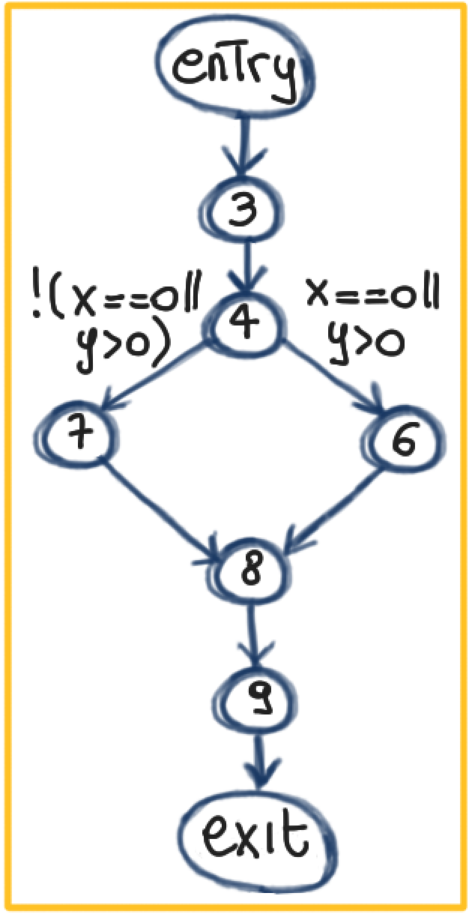


TEST CRITERIA SUBSUMPTION



LET'S CONSIDER OUR LAST EXAMPLE

```
1.  void main(){
2.     float x,y;
3.     read(x);
4.     read(y);
5.     if((x==0)||(y>0))
6.        y=y/x;
7.     else x=y+2;
8.     write(x);
9.     write(y);
10. }
```



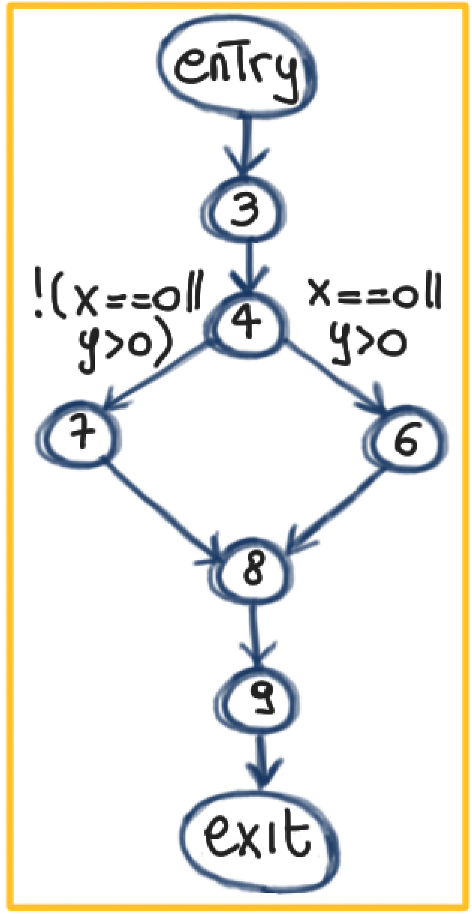
Tests: (x=0, y=-5)
(x=5, y=5)

Condition coverage: 100%

What about branch coverage?

LET'S CONSIDER OUR LAST EXAMPLE

```
1. void main(){
2.     float x,y;
3.     read(x);
4.     read(y);
5.     if((x==0)||(y>0))
6.         y=y/x;
7.     else  x=y+2;
8.     write(x);
9.     write(y);
10. }
```



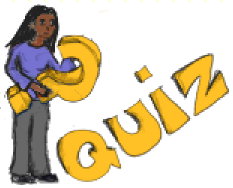
Tests: (x=0, y=-5)
(x=5, y=5)

Condition coverage: 100%

What about branch coverage? 50%

BRANCH AND CONDITION COVERAGE
(DECISION)

Test requirements	branches and individual conditions in the program
Coverage measure	Computed considering both coverage measures

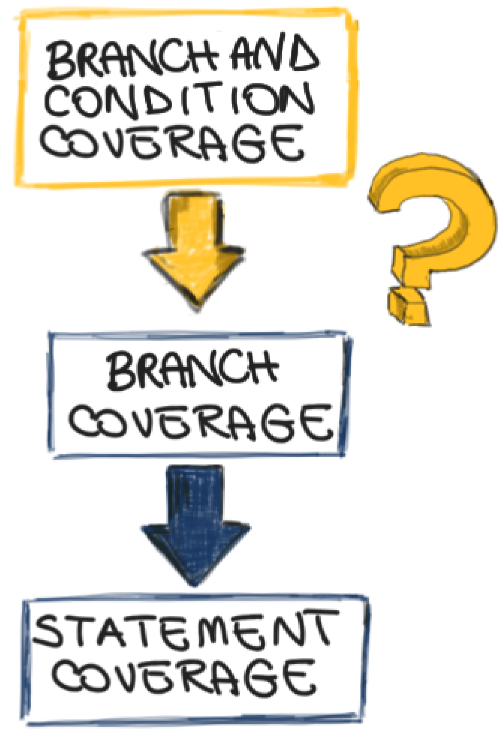


SUBSUMPTION

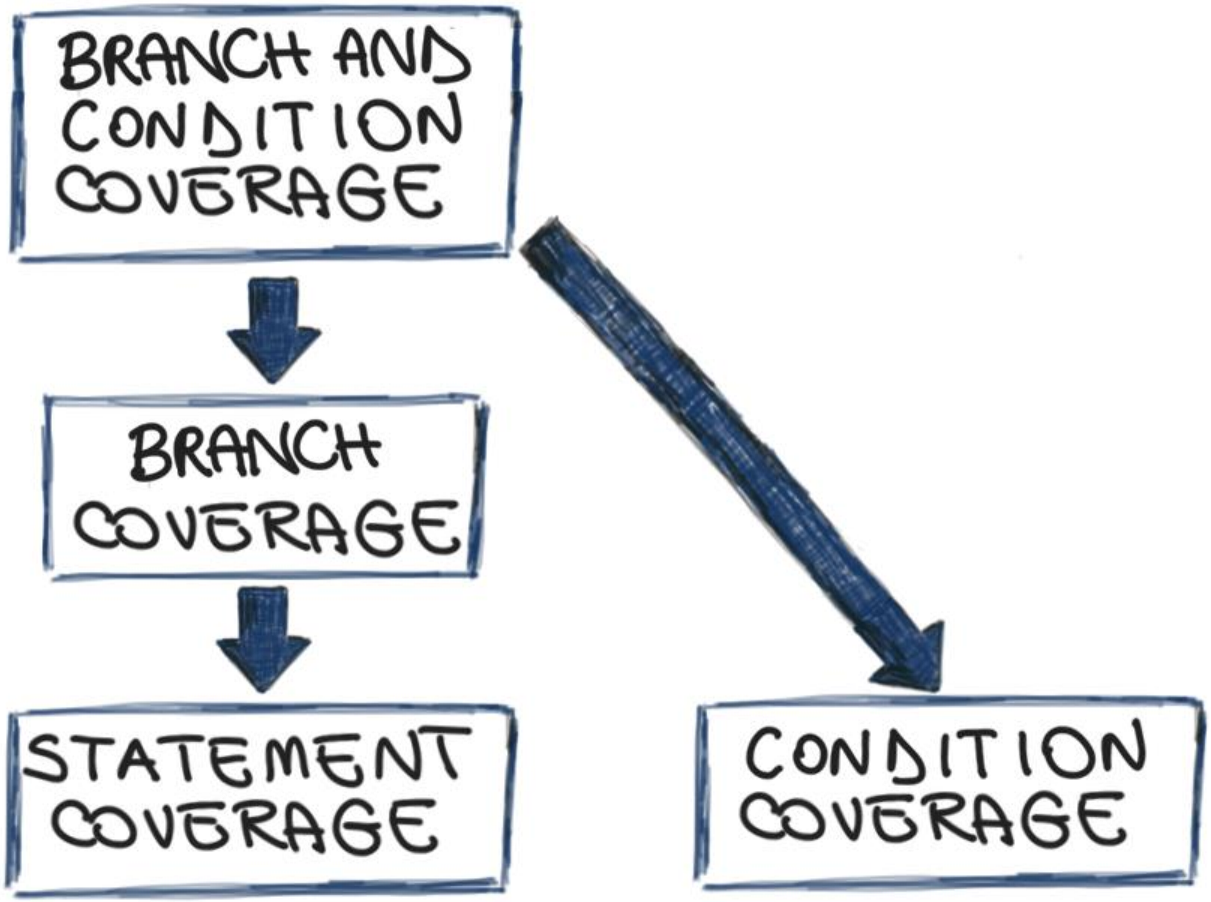
Does branch and condition coverage imply branch coverage?

[ ] Yes

[ ] No

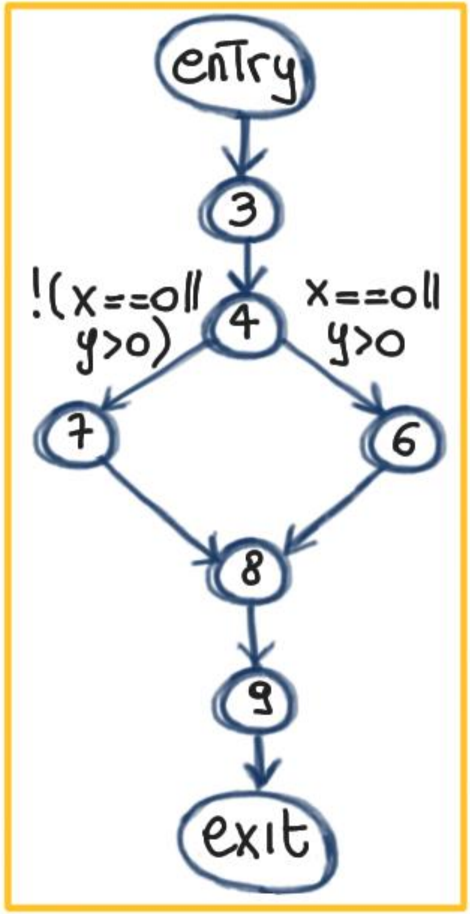


TEST CRITERIA SUBSUMPTION



ACHIEVING 100% B&C COVERAGE

```
1.  void main(){
2.     float x,y;
3.     read x;
4.     read y;
5.     if((x==0)||(y>0))
6.        y=y/x;
7.     else x=y+2;
8.     write(x);
9.     write(y);
10. }
```



Test cases

(x=0, y=-5)

(x=5, y=5)

Add a Test case
To achieve 100%
B&C coverage

(x=[ ], y=[ ])

MODIFIED CONDITION/DECISION COVERAGE (MC/DC)

Key idea: test important combinations of conditions and limited testing costs

⇒ extend branch and decision coverage with the requirement that each condition should affect the decision outcome independently.

MC/DC : EXAMPLE

a && b && c

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

MC/DC : EXAMPLE

a && b && c

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

MC/DC : EXAMPLE

a && b && c

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

1	True	True	True	True
5	False	True	True	False

MC/DC : EXAMPLE

a && b && c

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

1	True	True	True	True
5	False	True	True	False

MC/DC : EXAMPLE

a && b && c

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

1	True	True	True	True
5	False	True	True	False
3	True	False	True	False

MC/DC : EXAMPLE

a && b && c

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

1	True	True	True	True
5	False	True	True	False
3	True	False	True	False

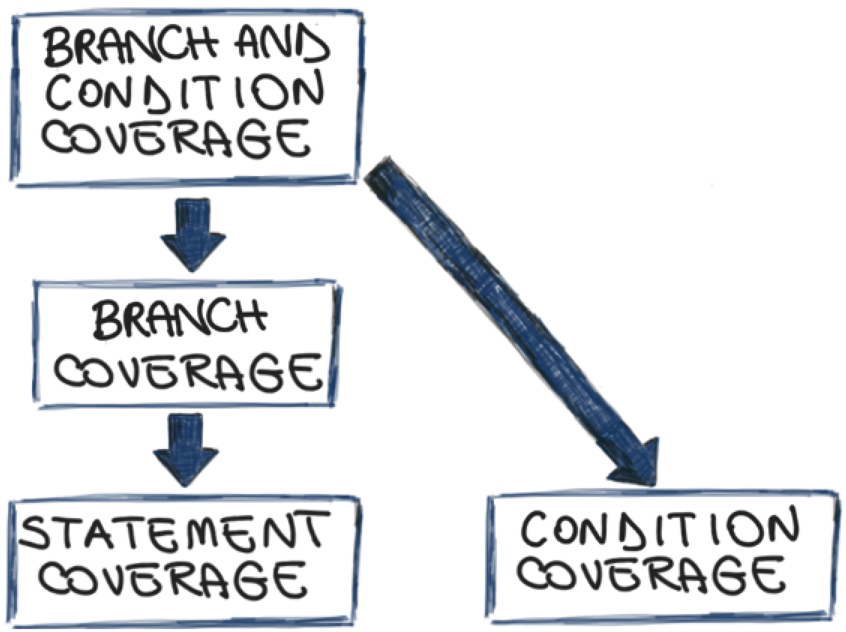
MC/DC : EXAMPLE

a && b && c

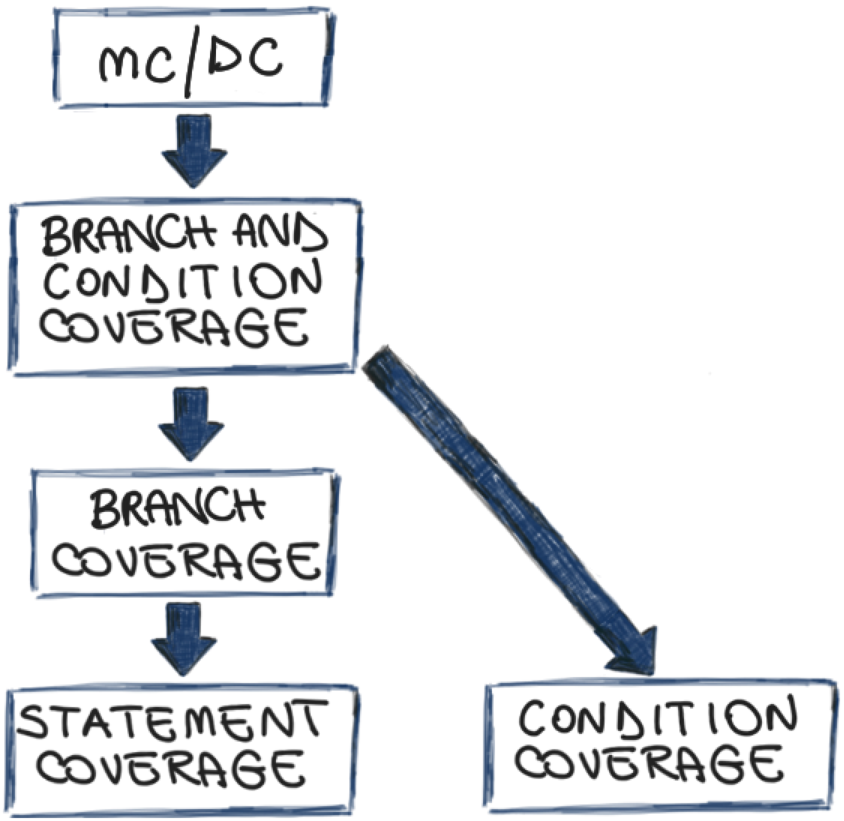
Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False

1	True	True	True	True
5	False	True	True	False
3	True	False	True	False
2	True	True	False	False

TEST CRITERIA SUBSUMPTION



TEST CRITERIA SUBSUMPTION



https://en.wikipedia.org/wiki/Code_coverage

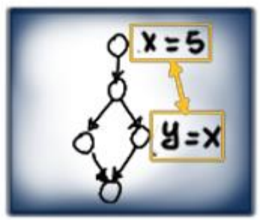
OTHER CRITERIA

OTHER CRITERIA

Path coverage

OTHER CRITERIA

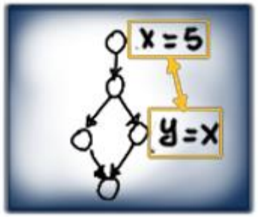
Path coverage



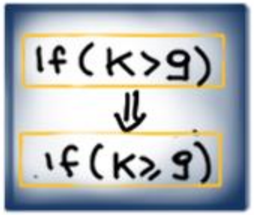
Data-flow coverage

OTHER CRITERIA

Path coverage

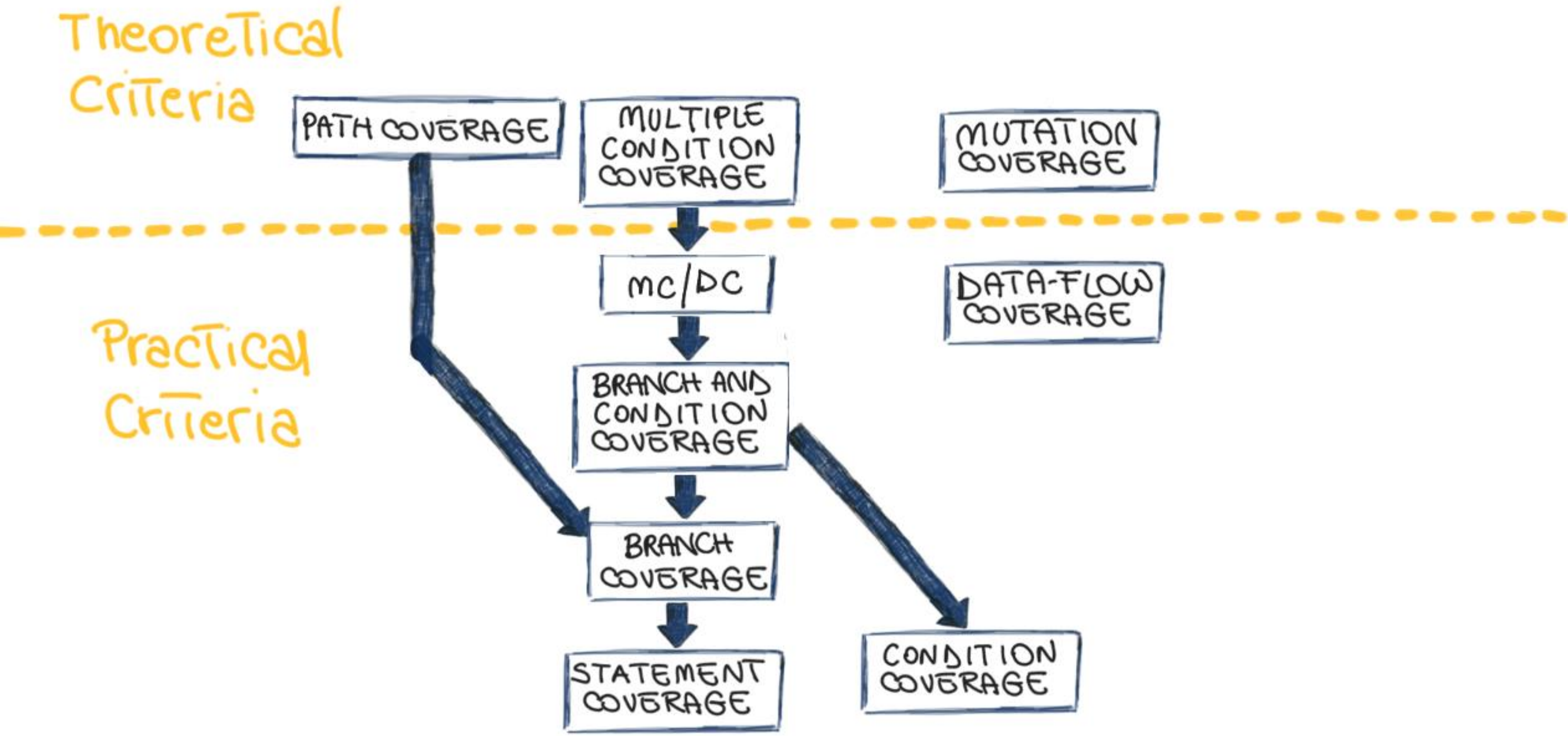


Data-flow coverage



Mutation coverage

TEST CRITERIA SUBSUMPTION




```
1. int i;
2. read(i);
3. print(10/(i-3));
```

Test suite: (1, -5), (-1, 2.5), $(0, -3.\overline{3})$


```
int i;
read(i);
print(10/(i-3));
```

Test suite: (1, -5), (-1, 2.5), (0, $-3.\overline{3}$)

Does it achieve path coverage?

[ ] Yes
[ ] No


```
1. int i;
2. read(i);
3. print(10/(i-3));
```

Test suite: (1, -5), (-1, 2.5), (0, $-3.\overline{3}$)

Does it achieve path coverage?

[ ] Yes

[ ] No

Does it reveal the fault at line 3?

[ ] Yes

[ ] No


```
int i=0;
int j;
read(j);
if ((j>5)&&(i>0))
    print(i);
```

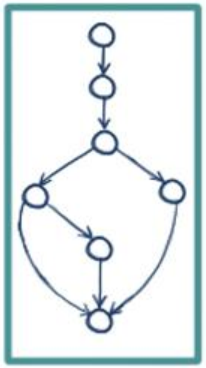

```
int i=0;
int j;
read(j);
if ((j>5)&&(i>0))
    print(i);
```

Can you create a test suite to achieve statement coverage?

[ ] Yes
[ ] No

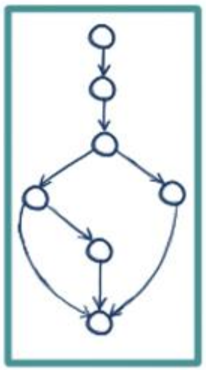
WHITE-BOX TESTING SUMMARY

WHITE-BOX TESTING SUMMARY

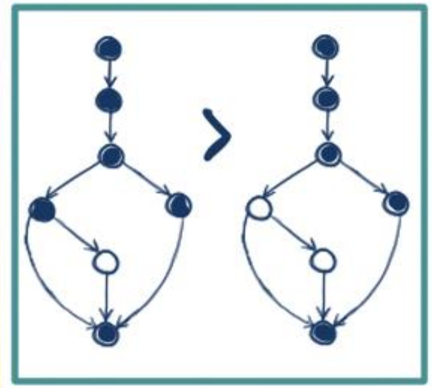


works on a formal model

WHITE-BOX TESTING SUMMARY

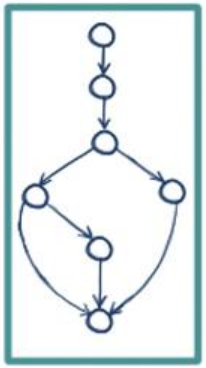


works on a formal model

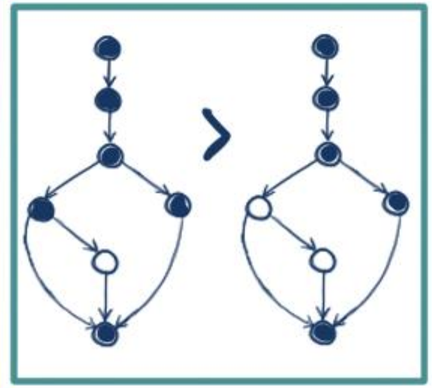


Comparable

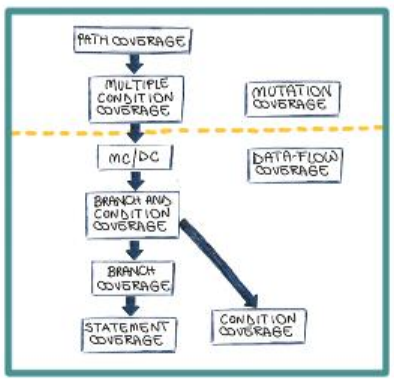
WHITE-BOX TESTING SUMMARY



Works on a formal model

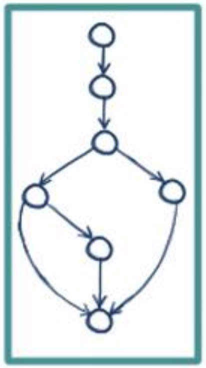


Comparable

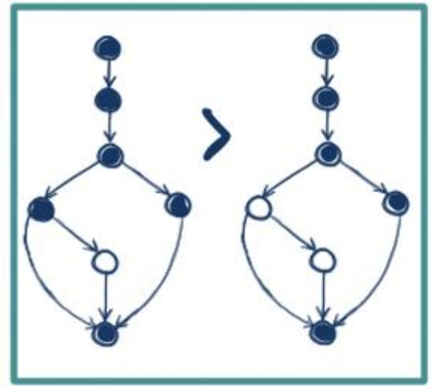


Two broad classes:
practical
Theoretical

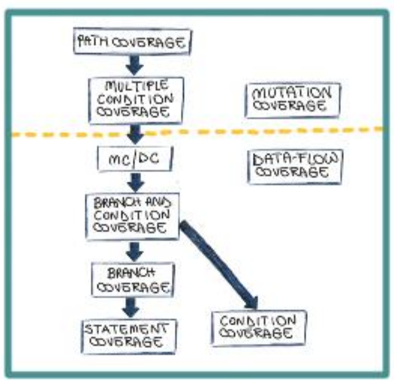
WHITE-BOX TESTING SUMMARY



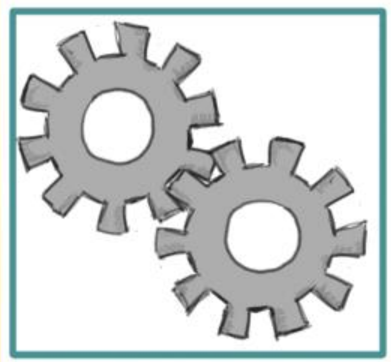
Works on a formal model



Comparable



Two broad classes:
practical
Theoretical



Fully automatable