

Announcements

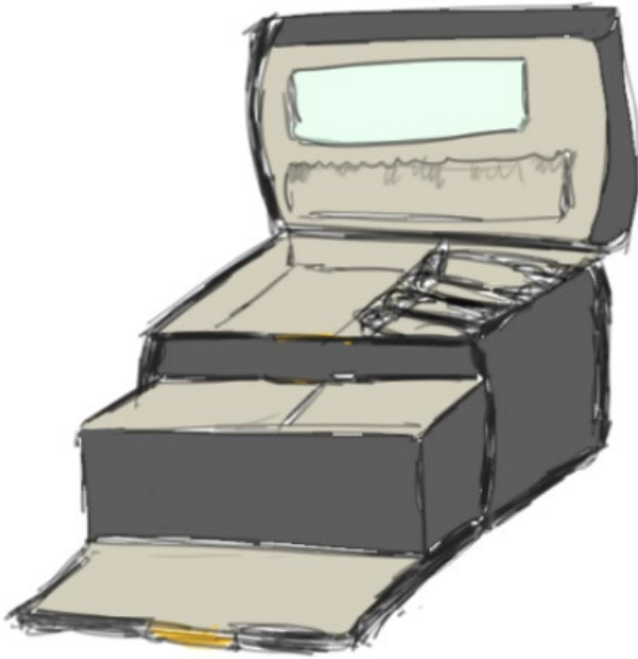
- Project 2 Progress Report Due 11/7
- Project 2 Test Assignment Due 11/9
- **Extra Credit opportunities in today's lecture.**
 - Scaled - not binary

CS3300 Introduction to Software Engineering

Lecture 17: White-Box Testing

Dr. Nimisha Roy ▶ nroy9@gatech.edu

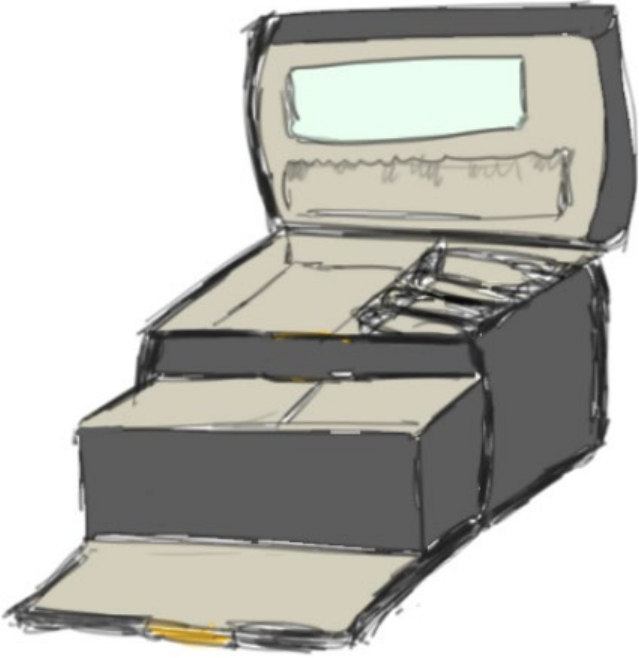
White- Box Testing



Basic Assumption

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

White- Box Testing



Different Kinds

- Control-Flow Based
- Data-flow based
- Fault based

Coverage Criteria

Defined in terms of

Test requirements - Elements/entities in the code that we need to execute

Result in

Test specifications

Test cases

Coverage Criteria: Statement Coverage

Test
Requirements

Statements in the program

Coverage
Measure

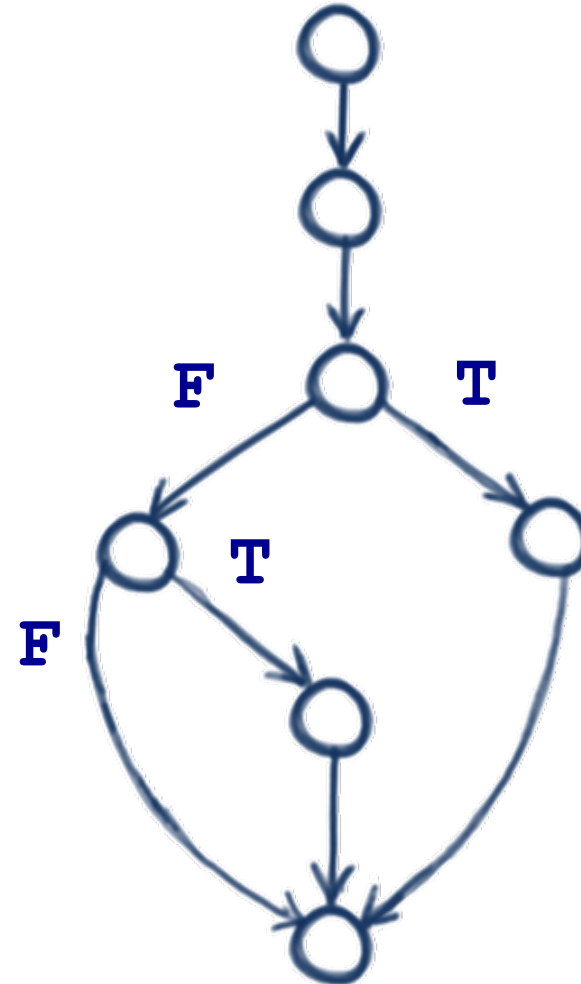
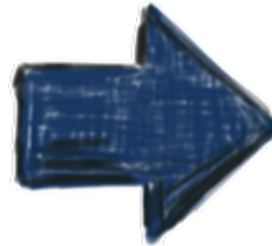
$$\frac{\text{Number of executed Statements}}{\text{Total number of Statements}}$$

Control Flow Graphs

Representation for the code that is very convenient when we run our reason about the code and its structure.

Represents statement with nodes and the flow of control within the code with edges.

```
1. printSum(int a, int b){  
2.   int result = a+b;  
3.   if (result > 0)  
4.     printool("red", result);  
5.   else if (result < 0)  
6.     printool("blue", result);  
    [else do nothing]  
7. }
```



Coverage Criteria: Branch Coverage

Test
Requirements

Branches in the program: outgoing
edges from a decision point

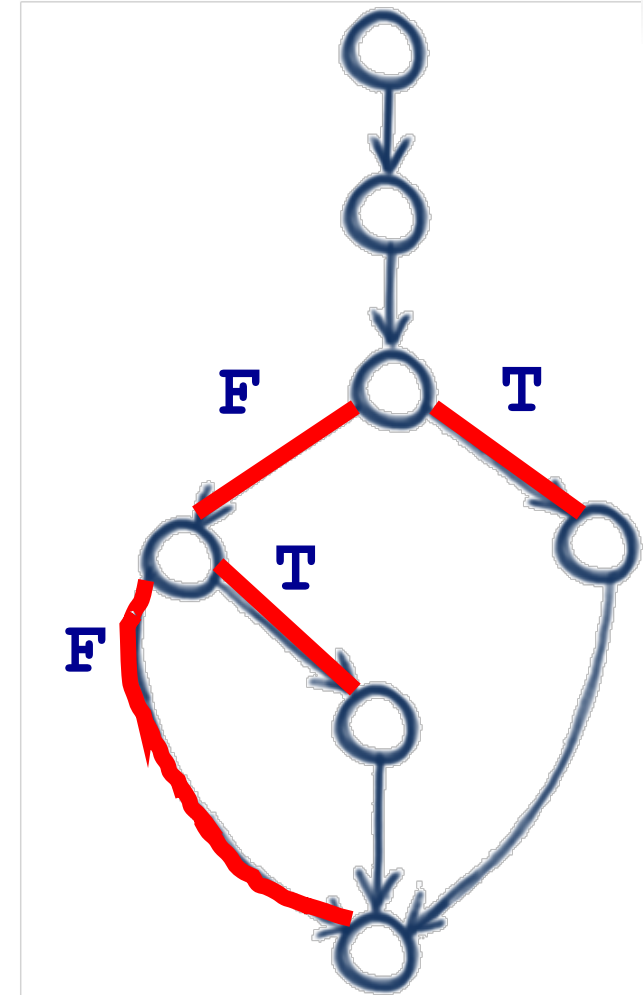
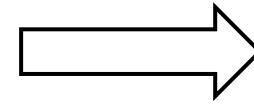
Coverage
Measure

$$\frac{\text{Number of executed Branches}}{\text{Total number of Branches}}$$

printSum: Branch coverage



```
1. printSum (int a, int b) {  
2.   int result = a+b;  
3.   if (result > 0)  
4.     printcol("red", result);  
5.   else if (result < 0)  
6.     printcol("blue", result);  
7.   [else DO NOTHING]  
8. }
```



How many branches? [4]

printSum: Branch coverage

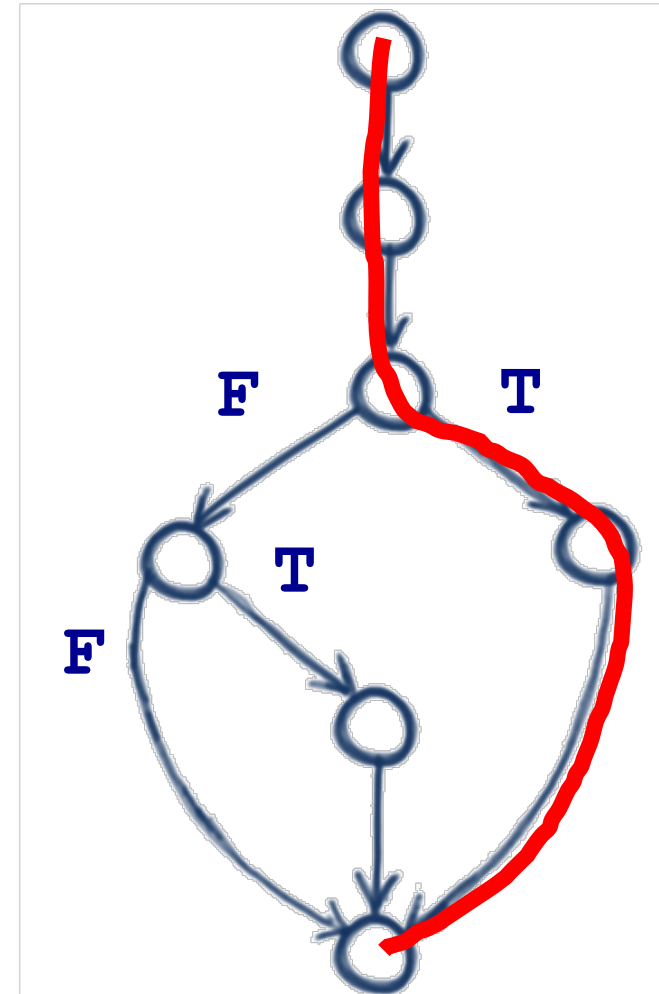
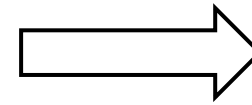
TC #1

a == 5

b == -4

1. printSum (int a, int b) {
2. int result = a+b;
3. if (result > 0)
4. printcol("red", result);
5. else if (result < 0)
6. printcol("blue", result);
7. [else DO NOTHING]
8. }

Coverage [25 %]



printSum: Branch coverage

TC #1

a == 5

b == -4

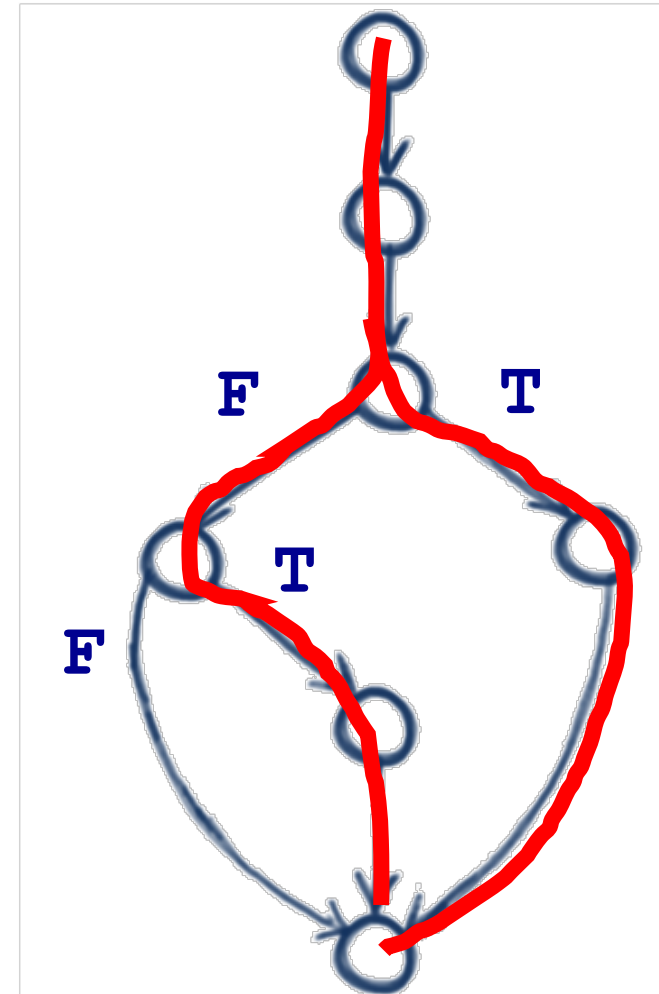
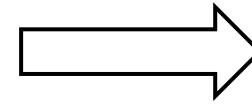
TC #2

a == 0

b == -1

1. printSum (int a, int b) {
2. int result = a+b;
3. if (result > 0)
4. printcol("red", result);
5. else if (result < 0)
6. printcol("blue", result);
7. [else DO NOTHING]
8. }

Coverage [75 %]



printSum: Branch coverage



TC #1

a == 5

b == -4

TC #2

a == 0

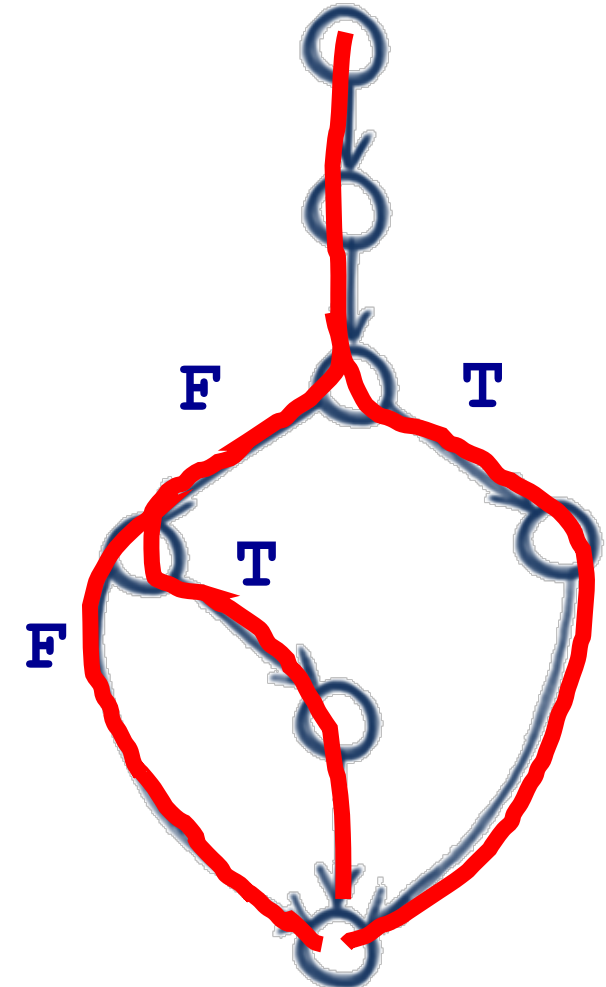
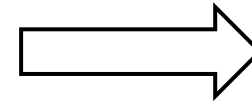
b == -1

TC #3

a == 0

b == 0

```
1. printSum (int a, int b) {  
2.   int result = a+b;  
3.   if (result > 0)  
4.     printcol("red", result);  
5.   else if (result < 0)  
6.     printcol("blue", result);  
7.   [else DO NOTHING]  
8. }
```

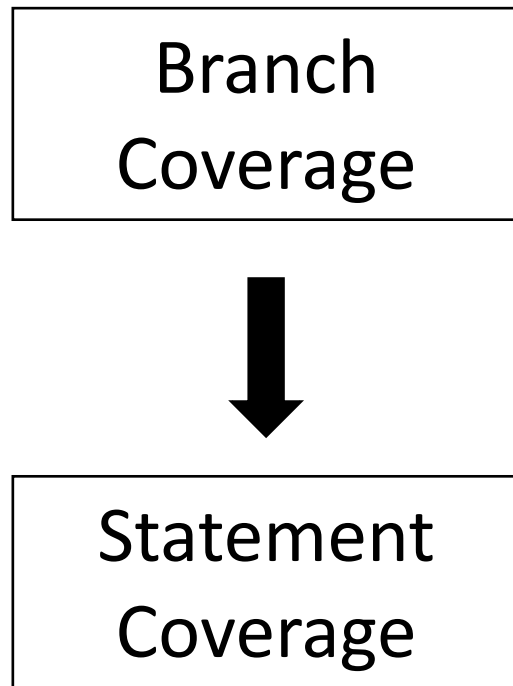


Coverage [100 %]

Note: 100% coverage does not provide any guarantee of finding the problems in the code.

Test Criteria Subsumption

One test criteria subsumes another criteria when all the test suites that satisfy that criteria will also satisfy the other one



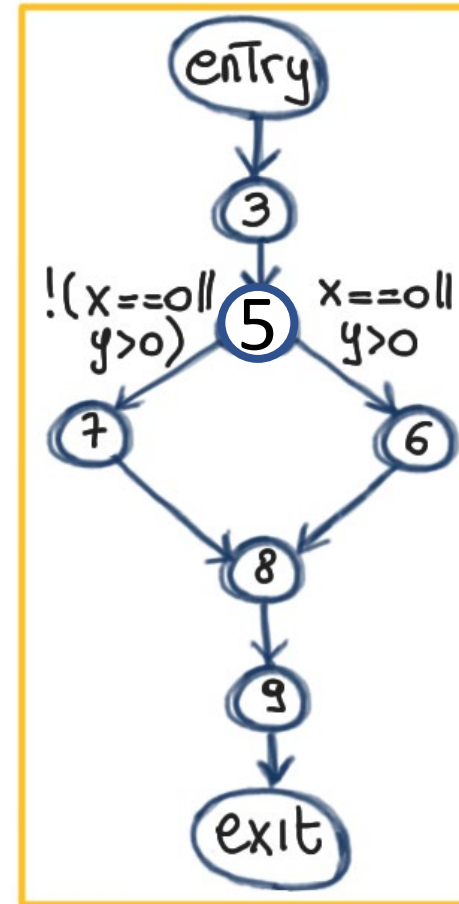
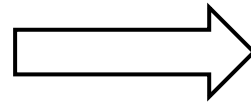
Branch Coverage is a stronger criteria than Statement Coverage. There is no way of covering all branches but leaving out some statements.

Lets 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.}
```

Lets consider another example

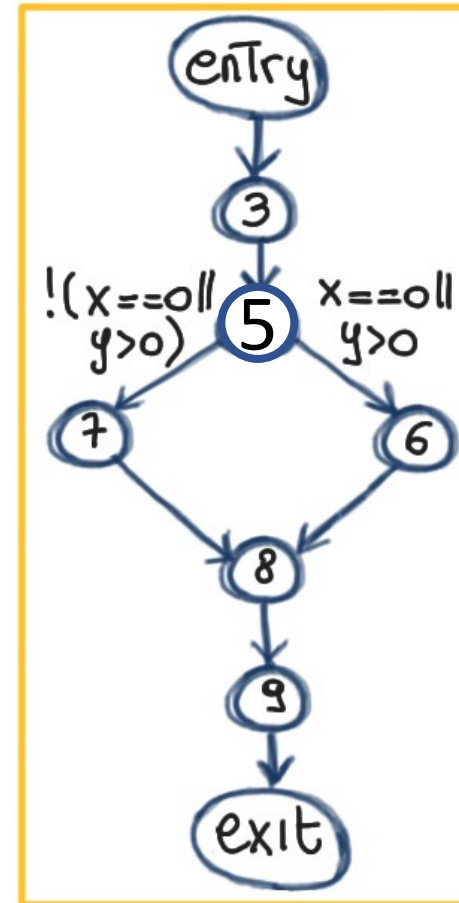
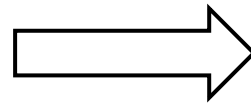
```
1. void main () {  
2.   float x, y;  
3.   read (x);  
4.   read (y);  
5.   if ((x== 0) || (y > 0))  
6.     y = y/x;  
7.   else      x = y+2;  
8.   write (x);  
9.   write(y);  
10.}
```



Lets 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.}
```



x = 5; y = 5;
x = 5; y = -5;

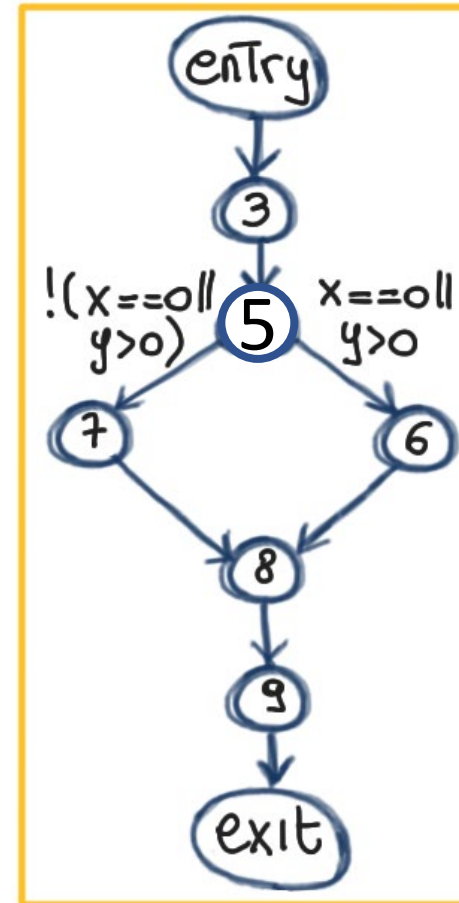
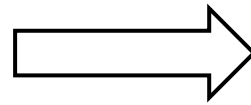
Branch Coverage: ?

100%

Lets 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.}
```



x = 5; y = 5;
x = 5; y = -5;

Branch Coverage:

100%

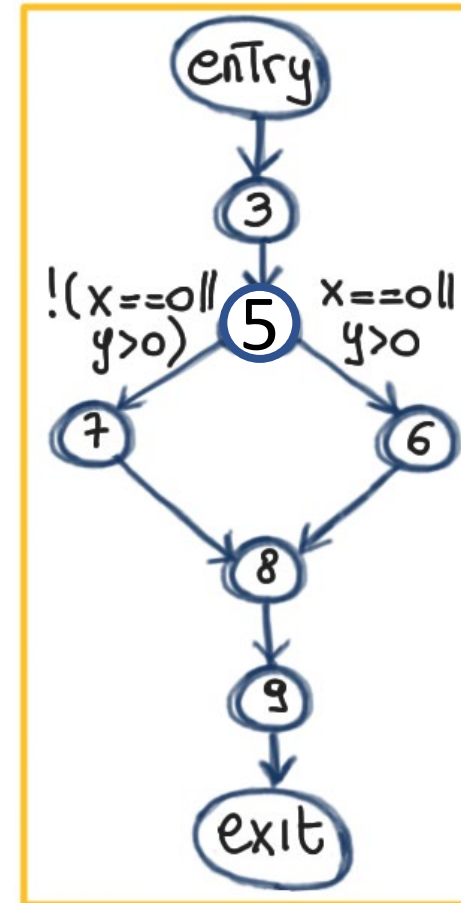
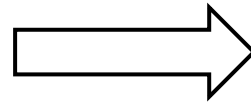
Identify a test case when code can fail:

x = 0, y can be anything

Lets 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.}
```



x = 5; y = 5;
x = 5; y = -5;

Branch Coverage:

100%

Identify a test case when code can fail:

x = 0

How can we be more thorough?

Each condition T and F

Coverage Criteria: Condition Coverage

Test
Requirements

Individual Conditions in the program

Coverage
Measure

$$\frac{\text{Number of conditions that are both T and F}}{\text{Total number of Conditions}}$$

Has each condition evaluated to true and false?

Subsumption



Does Condition Coverage imply
branch coverage?

[☐] Yes

[☒] No

Condition
Coverage

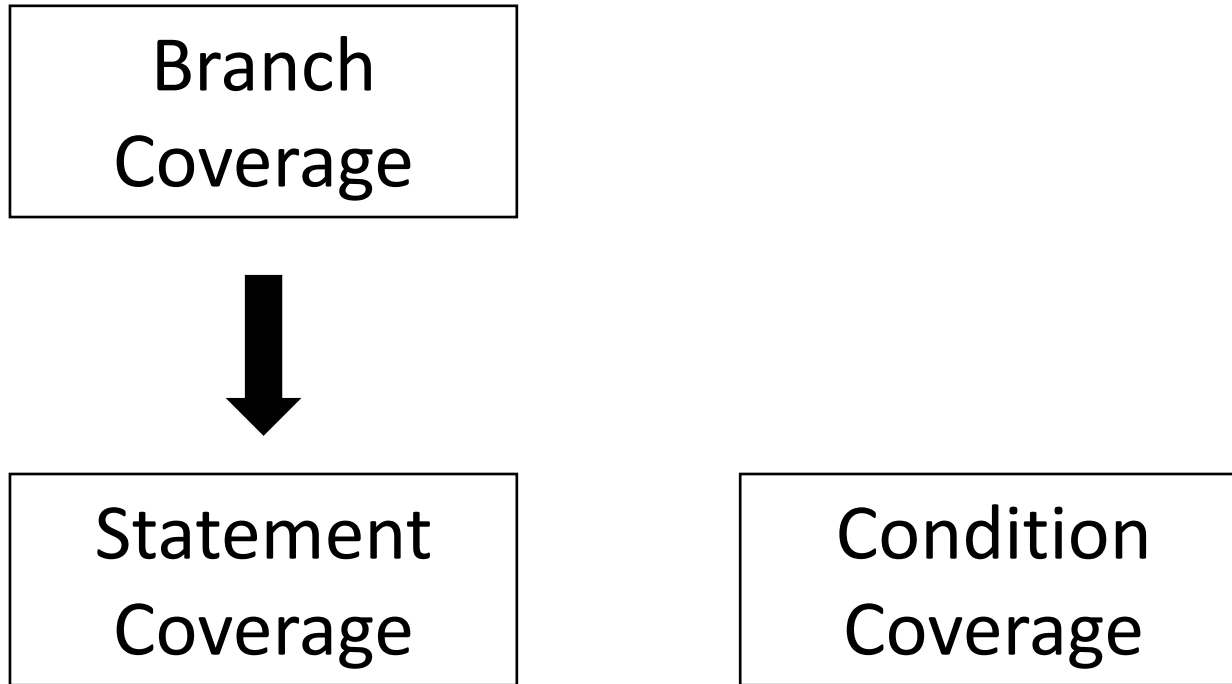


Branch
Coverage



Statement
Coverage

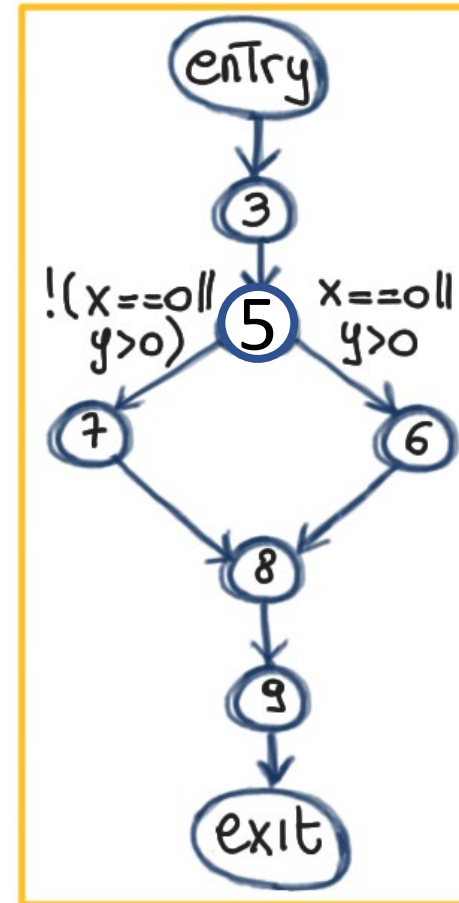
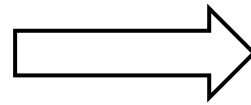
Test Criteria Subsumption



Lets consider the previous 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.}
```



x = 0; y = -5;
x = 5; y = 5;

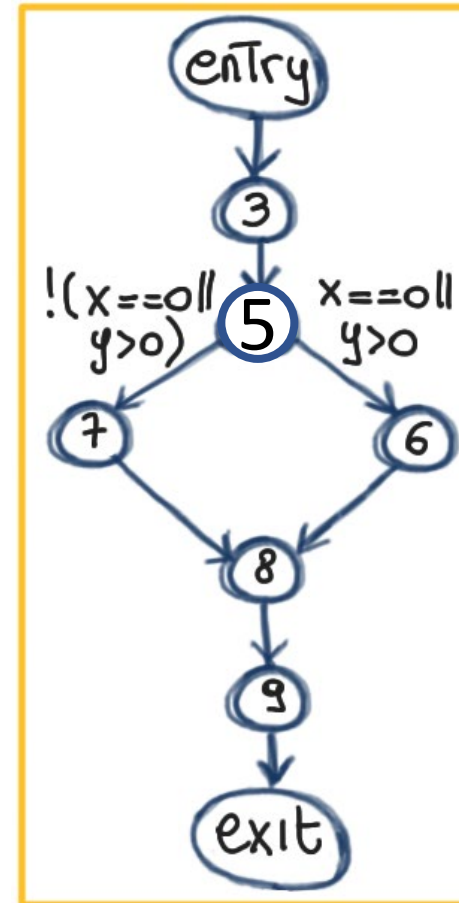
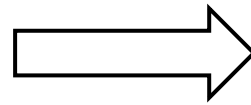
Condition Coverage: ?

100%

Lets consider the previous 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.}
```



x = 0; y = -5;
x = 5; y = 5;

Condition Coverage: ?

100%

Branch Coverage:?

50 %

Coverage Criteria: Branch and Condition Coverage

Test
Requirements

Branches and Individual Conditions in
the program

Coverage
Measure

Computed using both coverage
measures

Subsumption



Does Branch and Condition Coverage imply branch coverage?

- ☒ Yes
- ☐ No

Branch and Condition Coverage

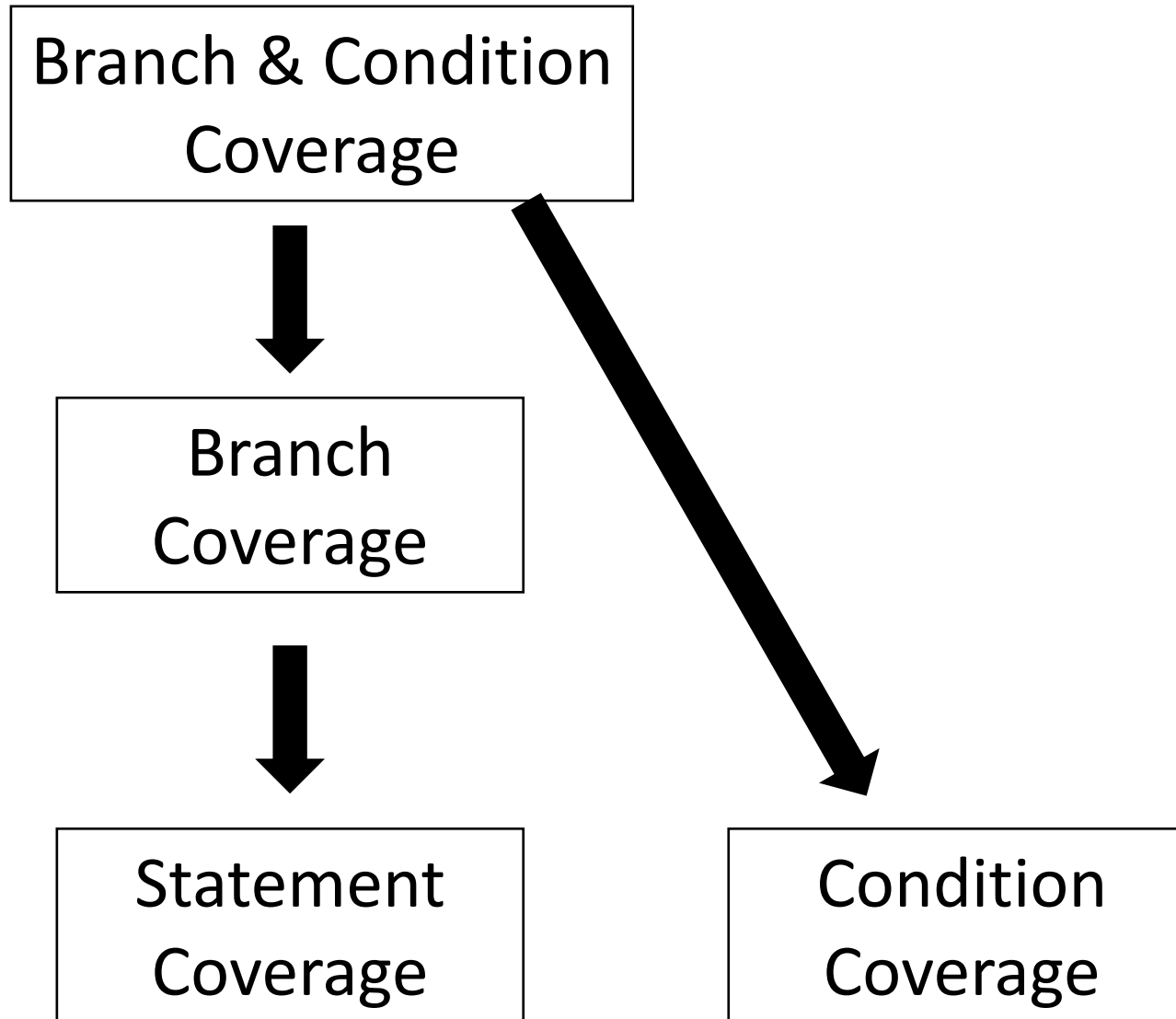


Branch Coverage



Statement Coverage

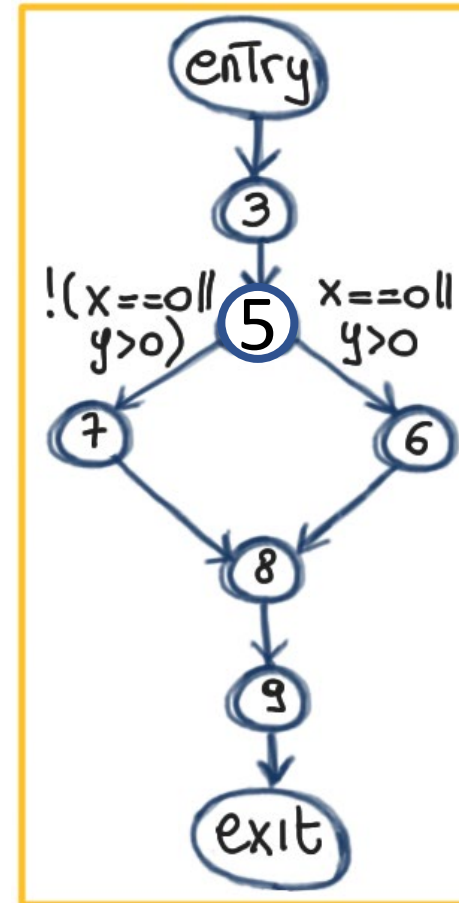
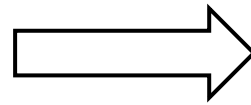
Test Criteria Subsumption



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.}
```



$x = 0; y = -5;$
 $x = 5; y = 5;$

Add a test case
to achieve 100%
B&C Coverage

$x = 3, y = -2$

**Multiple Condition
Coverage** –
permutation-
combination of
conditions in a
decision statement

Coverage Criteria: Modified Condition/Decision Coverage

Very Important Criteria; Often required for safety critical applications. For example: FAA requires SW that runs on commercial airplanes to be tested according to this criteria

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



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

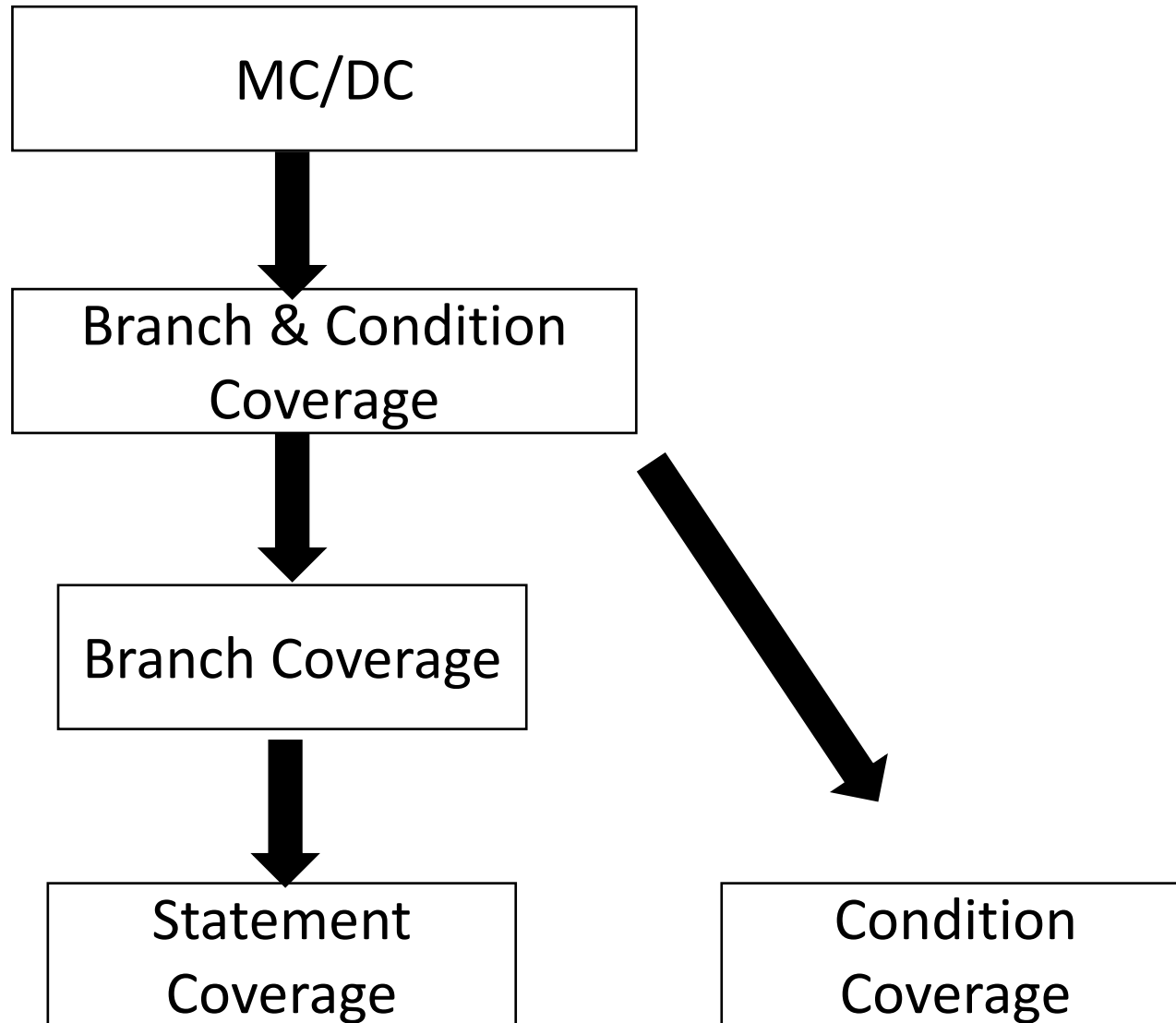
8 TC

To

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

4 TC

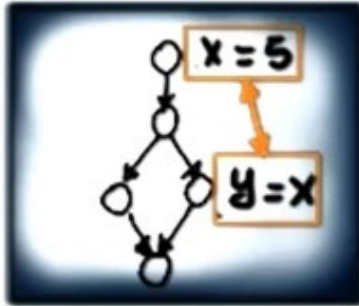
Test Criteria Subsumption



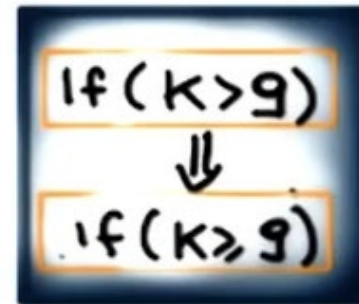
Other Criteria



Path Coverage (all paths are covered- incredibly expensive)

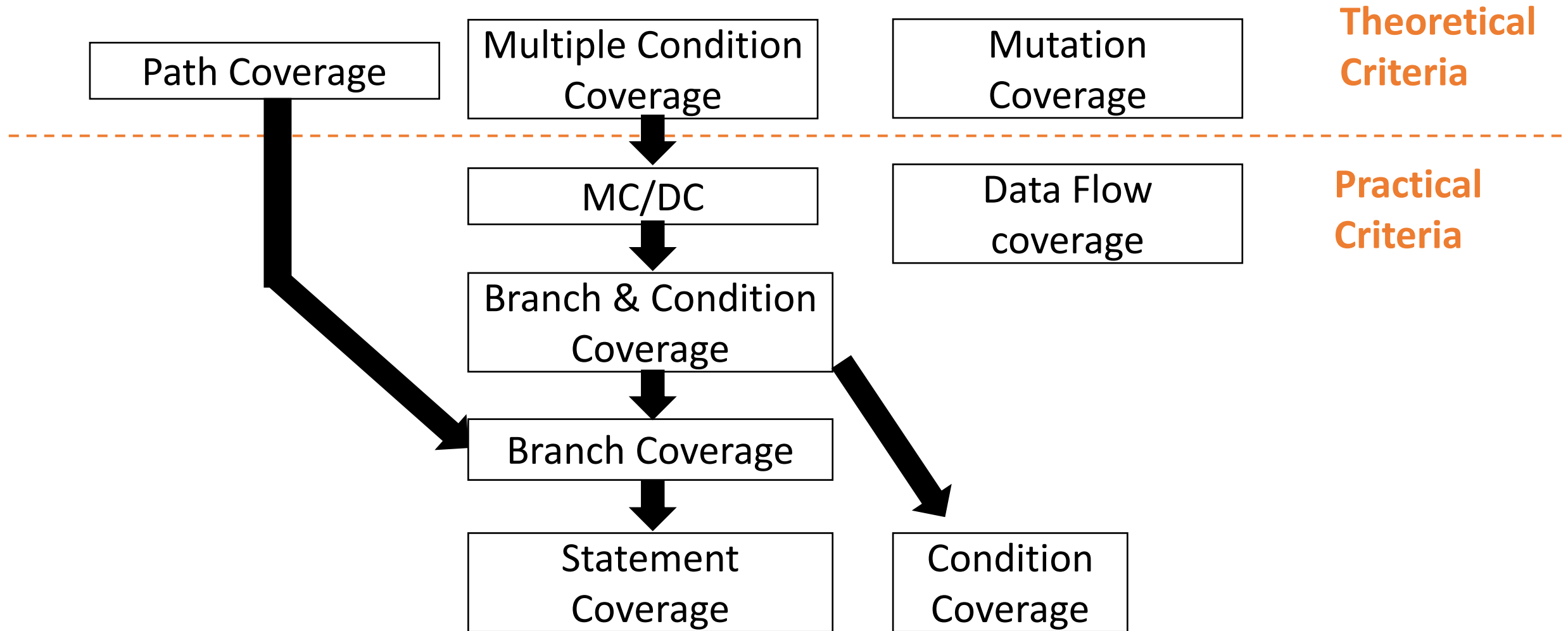


Data-Flow Coverage (coverage of pairs of elements; coverage of Statements, in which the content of some memory locations are modified, and statements in which the content of the same memory location is used)



Mutation Coverage (evaluate goodness of test by modifying the code; The more mutants identified by test, the better they are at identifying real faults)

Test Criteria Subsumption



White box testing Quiz



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

Test Suite: (1, -5), (-1, 2.5), (0, -3.3)

Does it achieve path coverage?

Yes

Does it reveal the fault at line 3?

No

Even path coverage couldn't detect the fault. Exhaustive testing is the only way to ensure all possible test cases.

White box testing Quiz



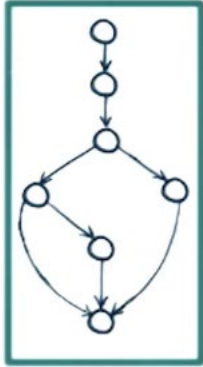
```
1. int i = 0;  
2. int j;  
3. read (j);  
4. if ((j > 5) && (i > 0))  
5.   print (i)
```

Can you create a test suite to adhere statement coverage?

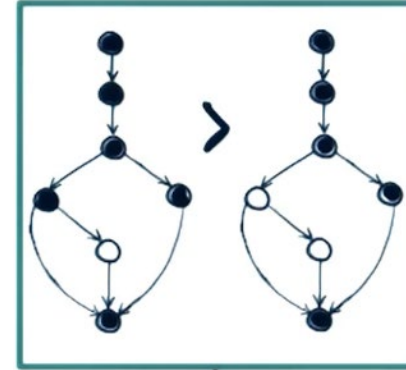
No; Dead/ Unreachable Code.

infeasible paths, inexecutable statements, conditions that can never be true all are present in codes. Hence industry targets ~80% coverage

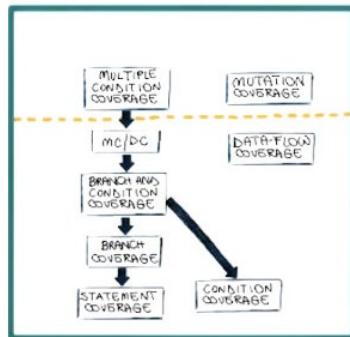
White-box testing Summary



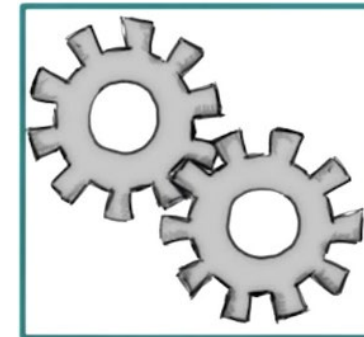
Works on a formal Model - No subjective decisions on level of abstraction needed



Comparable-coverage percentage as objective measure



2 broad classes:
Practical and
Theoretical



Fully Automatable

Industry Standard Today

- Junit
 - unit testing framework that supports test automation in Java Programming Language, provides the Test coverage report as well; licensed under Eclipse Public License
- [Nunit](#)
 - open source unit testing framework that supports all .NET languages
- [Fiddler](#)
 - Popular framework for web applications; logs and scrutinizes all HTTP(s) traffic between your system and the Internet.
- [Bugzilla](#)
 - popular defect tracking system; records the steps that lead up to reproduce the bug, so developers have all the information they need to fix it.
- [Parasoft Jtest](#)
 - Used to test and improve Java codebase on both development and production systems.maintain Junit tests
- Security vulnerabilities - [Wireshark](#) (network protocol analyzer), [ZAP](#), [Nmap](#)