

SE202

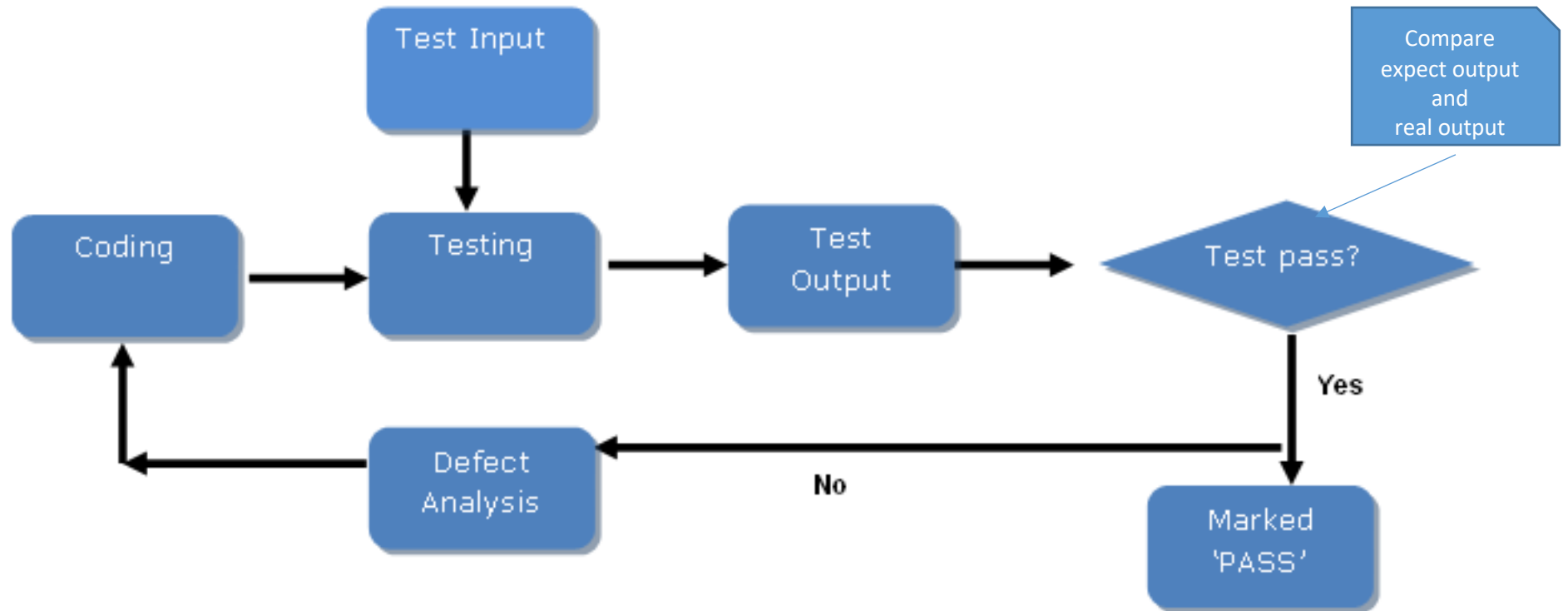
**Introduction to Software
Engineering**

Lecture 7.3
Software testing techniques

Last class

- Black box testing
 - Can be applied to every level of software testing
 - Attempts to find errors:
 - Incorrect or missing functions
 - Interface errors
 - Errors in data structures or external database access
 - Behavior or performance errors
 - Initialization and termination errors

Testing process



White-box testing



- Based on the code (know how the method does its work)
- Allows for covering the coded behavior
 - Statement coverage → branch coverage → and condition coverage → All-Paths coverage (exhaustive testing)
- Don't skip tests that you “know” the method can handle

Coverage criteria

Defined in term of



test requirements

Result in

test specifications (test condition)

test cases

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

```
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. }
```

Test req#1

Test spec#1: $a+b > 0$

Test req#2

Test spec#2: $a+b < 0$

Test cases

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

Statement coverage

Test requirements

Statements in the program

Coverage
measure

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

Most used in industry

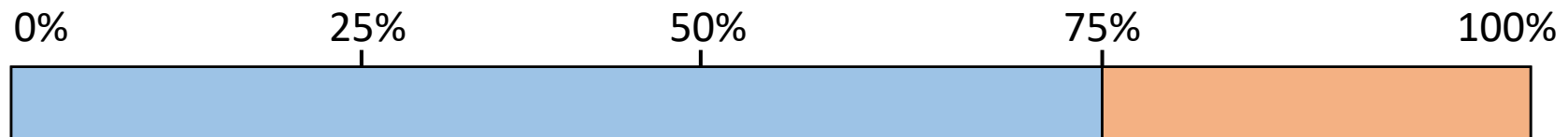
“Typical coverage target is 80-90%”

printSum: statement 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. }
```

TC #1
a = 3
b = 9

TC #2
a = -5
b = -8



Branch coverage

Test requirements

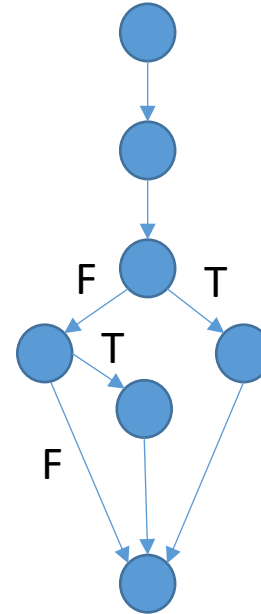
branches in the program

Coverage
measure

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

Branch coverage using control flow graphs (CFG)

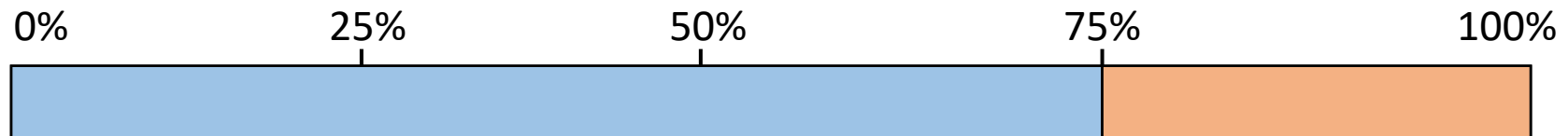
```
1. printSum (int a, int b) {  
2.     int result = a + b;  
3.     if (result > 0)  
4.         printcol("red", result);  
5.     else if (result < 0)  
6.         printcol("blue", result);  
7.     [else do nothing]  
8. }
```



TC #1
a = 3
b = 9

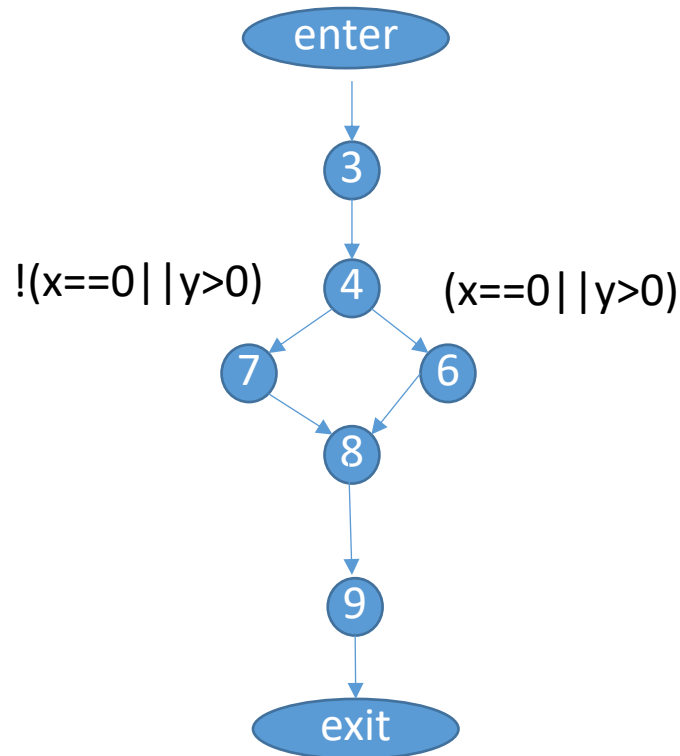
TC #2
a = -5
b = -8

TC #3
a = 0
b = 0



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)

Branch coverage 100%

Condition coverage

Test requirements

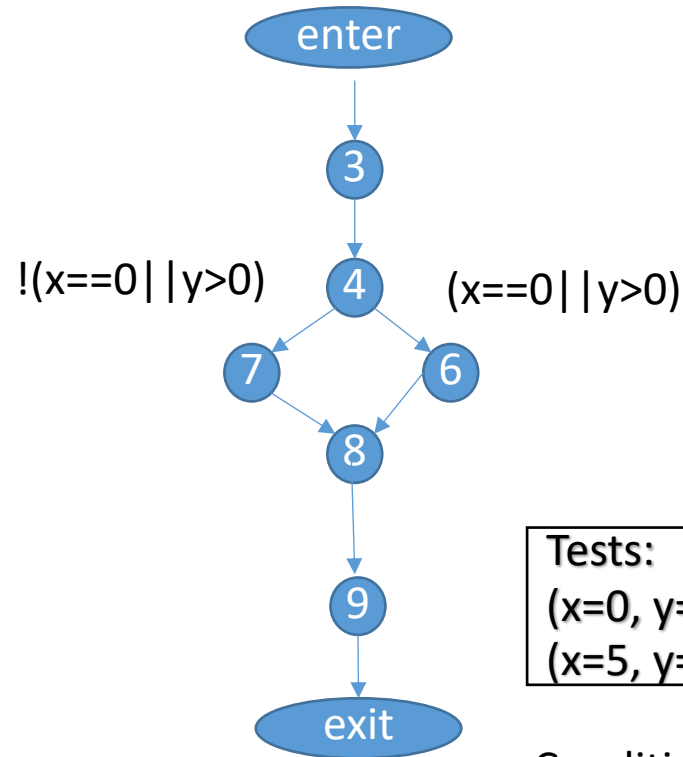
Individual conditions in the program

Coverage
measure

$$\frac{\text{number of conditions that are both T and F}}{\text{total number of conditions}}$$

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

Condition coverage 100%

What about branch coverage?
50%

Branch and Condition coverage

Test requirements

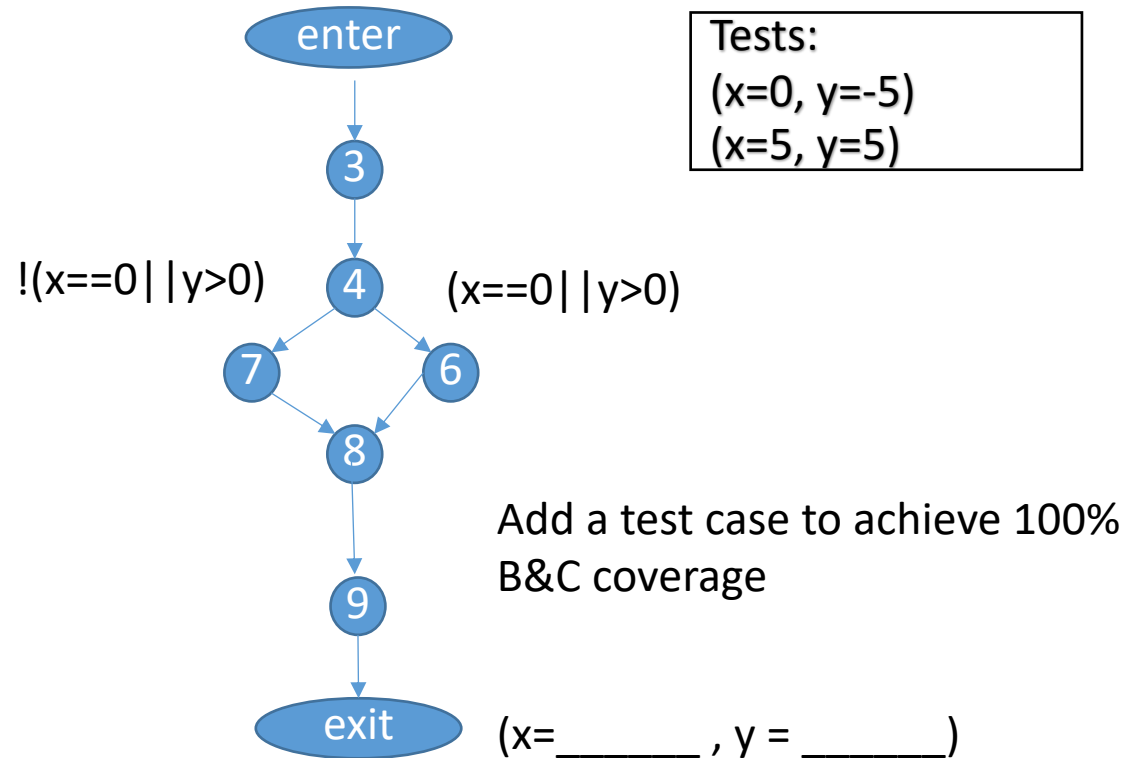
Branches and Individual conditions in the program

Coverage
measure

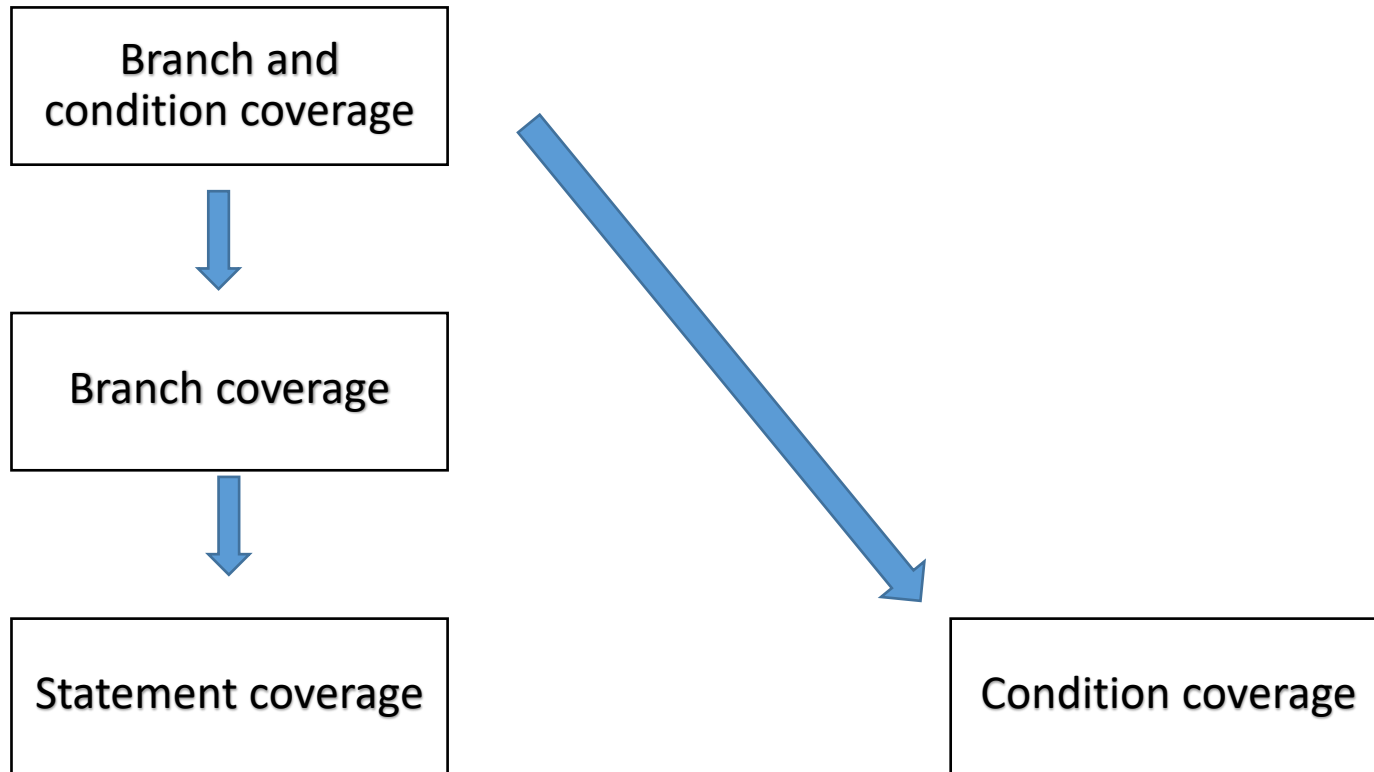
Computed considering both coverage measures

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 criteria subsumption





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

Every program contains DEAD or
UNREACHABLE code

Can you create a test suit to achieve statement coverage?

☐ yes

☐ no

Multiple Condition coverage

Test requirements

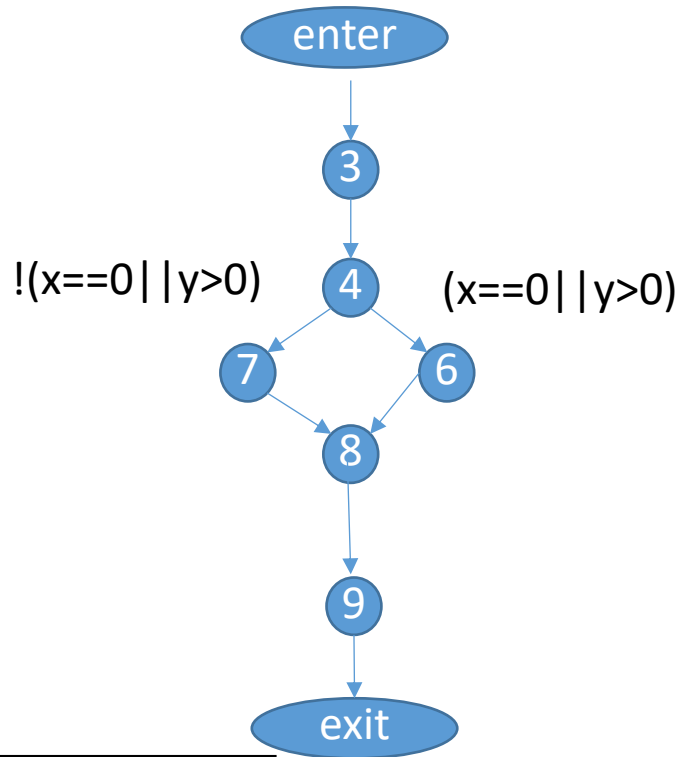
Invoke each point of entry at least once
Test cases can be arrived at by using truth table of the conditions

Coverage
measure

Computed considering every possible combination of Boolean sub expression occurs at least once

Multiple condition 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	Y	X Y
T	T	T
T	F	T
F	T	T
F	F	F

Tests:

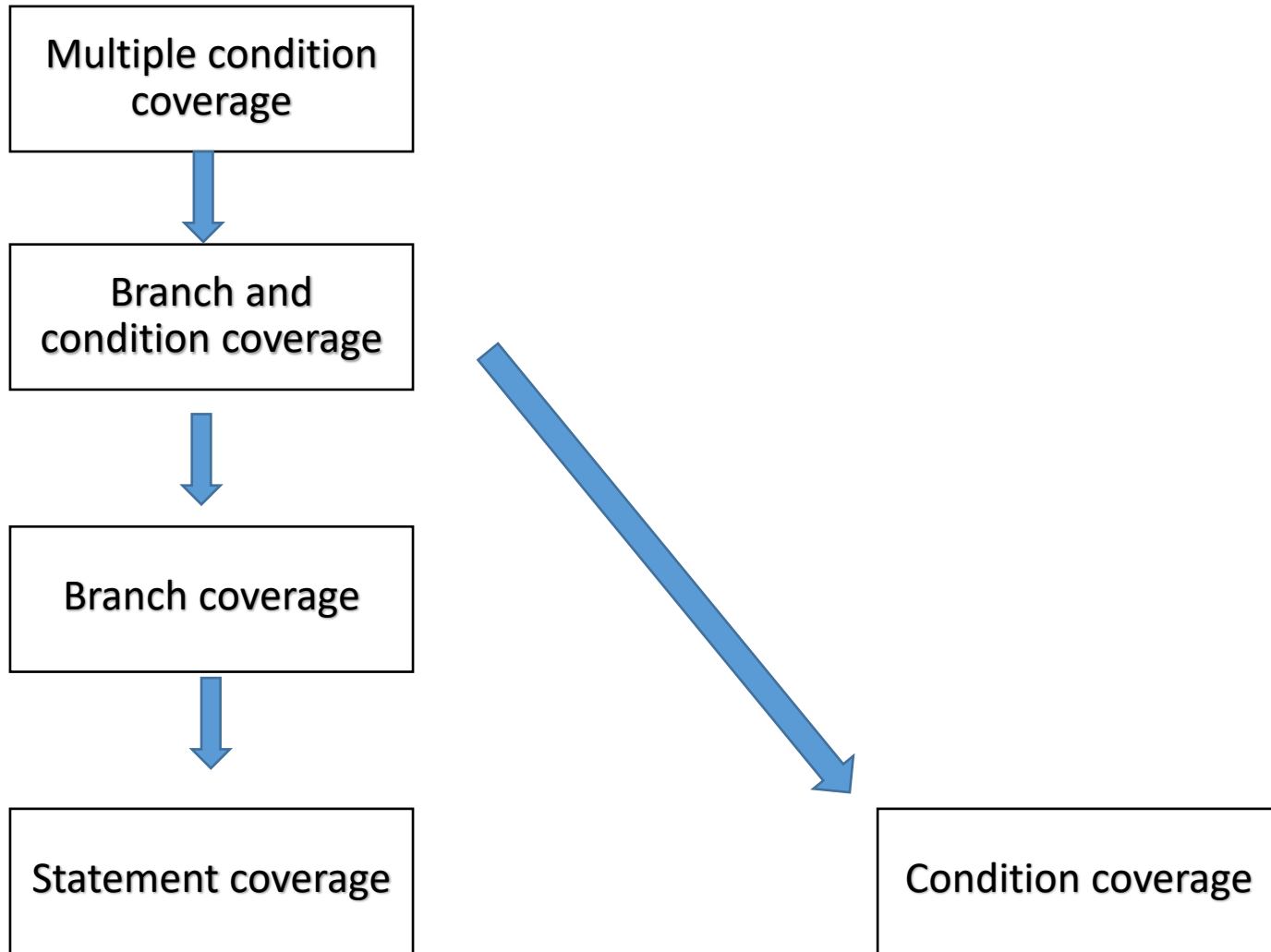
(x=0, y=5)

(x=0, y=-5)

(x=5, y=5)

(x=5, y=-5)

Test criteria subsumption



Grey-box testing

- A combination of white-box and black-box testing.
 - Use when some but not all the internals of the method are known by testers
- For example, suppose a method examines test score data to find students that might need extra tutoring help.
 - You don't know all the details, but you do know that it uses the quicksort algorithm to sort the students by their grades.
 - In that case, you might want to see what the method does if every student has the same grade because that might mess up quicksort. (Because you don't know what else is going on inside the method, you also need to write a bunch of black-box styles tests.)

Key points:

- Black-box testing and White-box testing are complementary techniques!!
 - The tester knows some but not all for the internals of the method
 - Use two programmers
 - Tester who didn't write the code → write black-box test
 - Tester who write the code → write white-box test
 - Timing
 - Before writing the code → write black-box test
 - After writing the code → write white-box test

Software testing terms

- Test plan
 - A high level document describing scope to testing, people responsible, testing strategies, criteria, testing environment, testing activities and more!
- Test Design Specification
 - Test description (Test condition)
 - Specify execution conditions (e.g. set of input) for testing
 - Test procedure (Test script)
 - Steps to run the test
 - Test case
 - A single test consisting of the inputs and expected results to be executed
- Test result
 - PASS or FAIL
 - A testing actual output comparing with an expected output

Example: test adding number feature of My Calculator program

- Test description (test condition)# 1:
 - Test adding number feature of My Calculator program by inputting 2 integer numbers on which the length of each number is in a range of the program.
- Test procedure#1
 - Step 1: Open the program by double click on “My Calculator” icon
 - Step 2: My Calculator program is started on the monitor
 - Step 3: Click mouse on number pad and the monitor displays “Input1 value + Input2 value” and click “=” button
 - Step 4: The monitor displays “Expected result value”
- Test cases:

Case	Description	Input	Expected result
1	Test adding 2 positive integer numbers	Input1 = 15, Input2 = 30	Expected result = 45
2	Test adding 2 negative integer numbers	Input1 = -15, Input2 = -30	Expected result = -45

Example: Test outbound links

- Test description#1
 - Test all outbound links of website
- Test procedure#1
 - Step 1: Locate the html file to be tested on tested machine
 - Step 2: Double click the file or right click to open the file
 - Step 3: Click outbound link
- Test cases

Case	Page	Description	Expected result
1	My Portfolio	Check outbound link of Floating mountain image	Link to http://www.skiresort.info
2	My Portfolio	Check outbound link of Fading sunset image	Link to https://thebeachseatrips.com

Test record

Case	Description	Expected result	Actual result	Pass/Fail
1	Check outbound link of Floating mountain image	Link to http://www.skiresort.info	Link to http://www.skiresort.info	Pass
2	Check outbound link of Fading sunset image	Link to https://thebeachseatrips.com	Link to http://www.devonbeachcompany.com/sea/	Fail

Bug tracking sheet

Date	Reviewer name	Page title	Issue type	Issue/comment	Change to	Status
01/04/18	Toey	My Portfolio	Outbound link	Wrong link when clicking fading sunset image	Change link to https://thebeachseatrips.com	Not resolved

Usability testing

- Usability is a quality attribute that assesses how easy user interfaces are to use.
- Usability testing is a technique used in user-centered interaction design to evaluate a product by testing it on users.
- Keys to usability testing is to observe the real user's behaviors and reactions to the user interfaces of website, app, or other product.

How the usability testing works

How

A participant uses the application to carry out pre-defined tasks.

The participant thinks aloud as he or she works.

Participants

5+ representative users (each tested individually).

Test moderator.

Observer.

Materials

Early prototype. Screenshots and paper mock-ups are fine.

Task scenarios.

Pen and paper.

Good usability test scenarios

- Are expressed in the user's language
- Realistic
 - i.e. they describe things that the user would normally expect to do.
- Don't hint at the correct path
 - a usability test is not a user acceptance test.
- Have a correct solution

Usability Testing Videos

- <https://usabilla.com/blog/7-must-see-usability-testing-videos/>