

# Objectives

- Coverage
- Testing wrap up

# Review

1. What is code coverage?

2. What is code coverage *criteria*?

- Provide examples of code coverage criteria

- Brainstorm:

- How can we leverage/use code coverage in our development process?

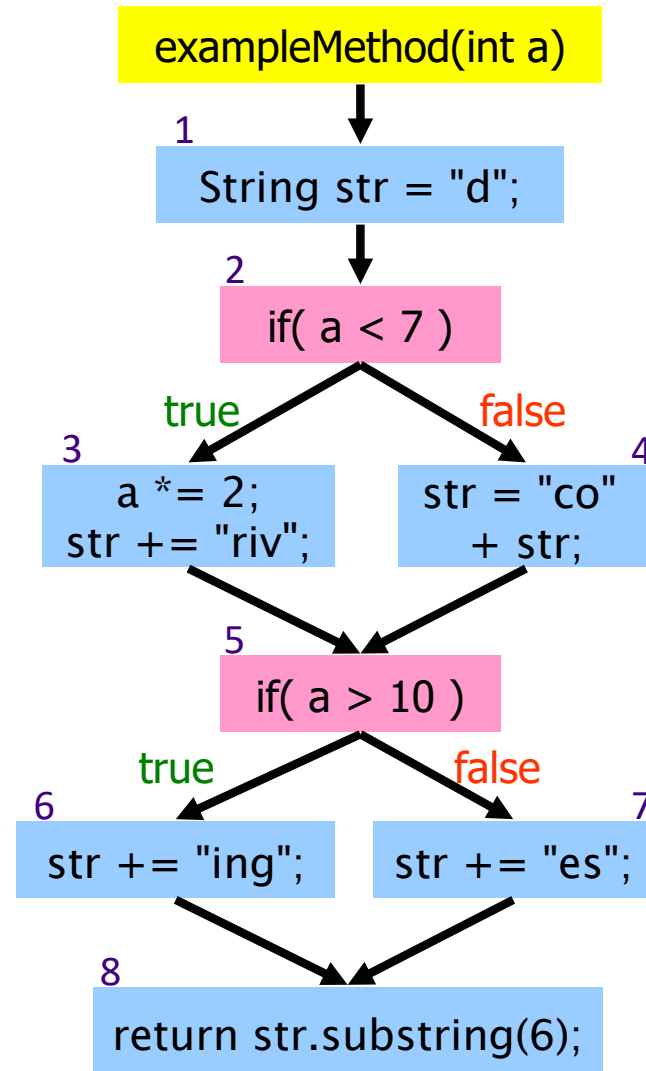
- What are the limitations of code coverage?

# Review: Code Coverage

- Code coverage: the amount of code that your tests execute
- Code coverage *criteria*: metric or measure used
  - Statement: number/% of statements executed
  - Branch: number/% of statements + branches (conditions, loops) executed
  - Path: number/% of paths executed

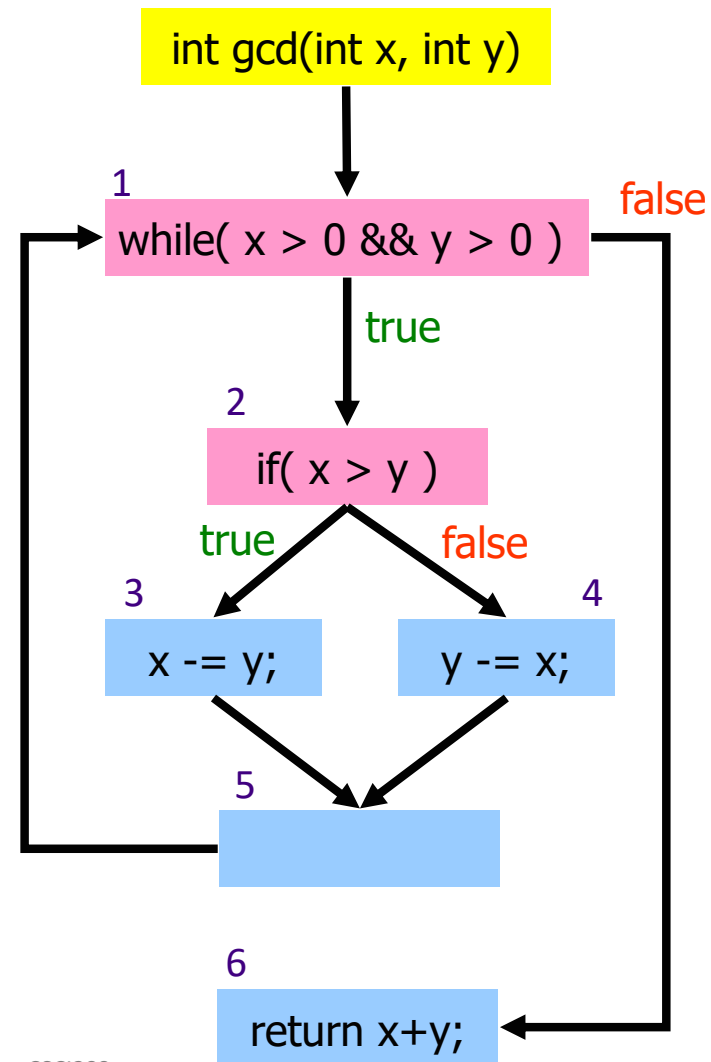
# Path Coverage

- Cover all **paths** in program's flow
- How many paths through this method? 4
  - 1-2-3-5-6-8
  - 1-2-3-5-7-8
  - 1-2-4-5-6-8
  - 1-2-4-5-7-8
- What test cases would give us path coverage?
  - One possibility:  $a = 3, 30, 6, 10$



## Example 3

```
/**
 * Euclid's algorithm to calculate
 * greatest common divisor
 */
public int gcd( int x, int y ) {
    while ( x > 0 && y > 0 ) {
        if( x > y ) {
            x -= y ;
        } else {
            y -= x;
        }
    }
    return x+y;
}
```



# Path Coverage

- How many paths through this method?

➤ Too many to count, test them all!

1-6

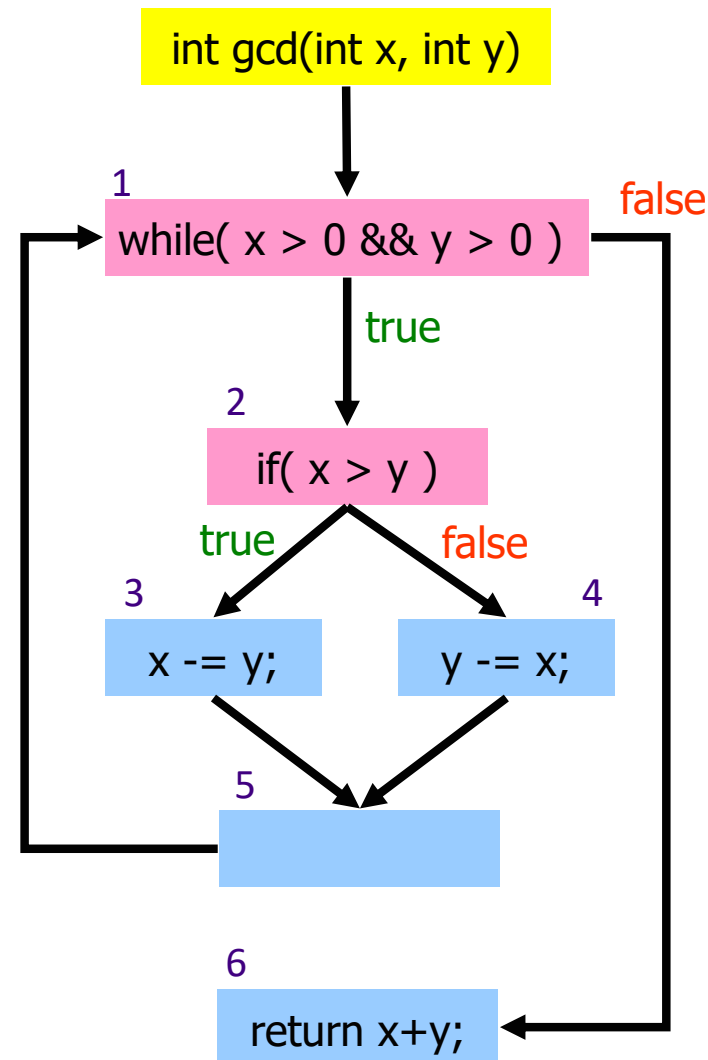
1-2-3-5-1-6

1-2-4-5-1-6

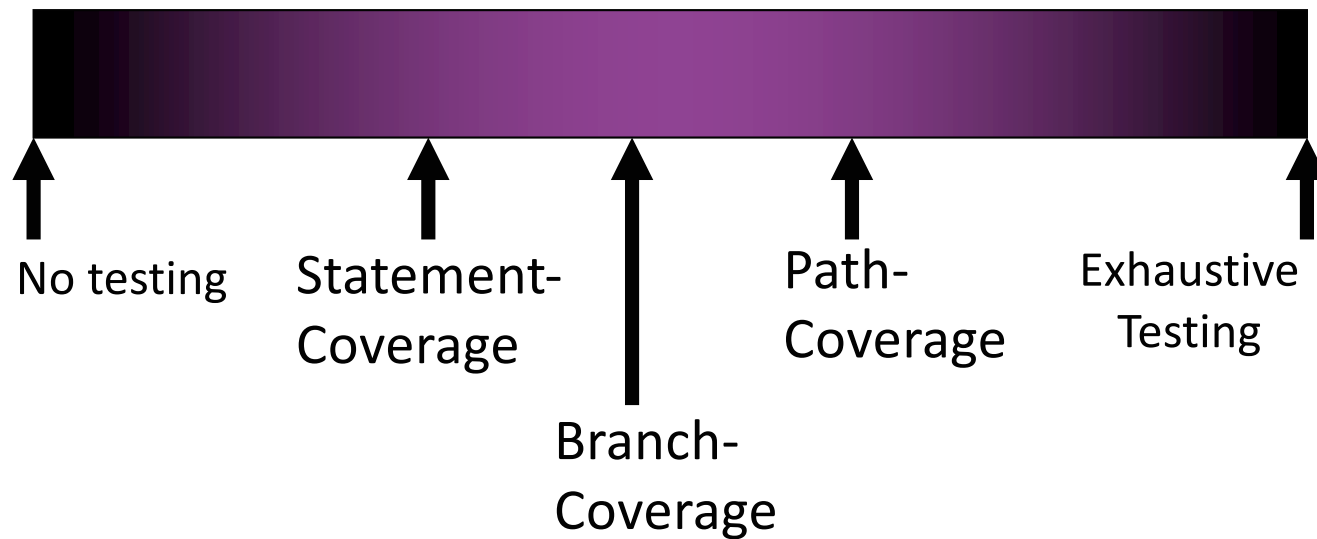
1-2-3-5-1-2-3-5-1-6

1-2-4-5-1-2-4-5-1-6

1-[2-(3|4)-5-1]\*-6



# Testing Continuum



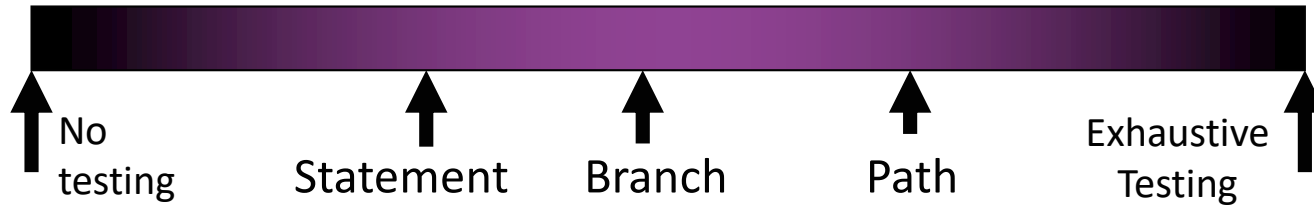
# Comparison of Coverage Criteria

Coverage Criterion	Advantages	Disadvantages
Statement		
Branch		
Path		

Consider how you would incorporate code coverage into your process

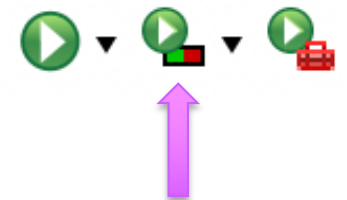


# Comparison of Coverage Criteria



Coverage Criterion	Advantages	Disadvantages
Statement	Practical	Weak, may miss many faults
Branch	Practical, Stronger than Statement	Weaker than Path
Path	Strongest	Infeasible, too many paths to be practical

# How Can We Use Coverage Criteria?



RevealingMutantsEvaluator (2) (Nov 6, 2023 10:53:54 AM)					
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions	
▼ CatchTheMutantsSource		1,941	201	2,142	
▼ src		1,941	201	2,142	
▼ mutants		1,260	155	1,415	
> Wolverine.java		113	68	181	
> Mutant1.java		73	23	96	
> Mutant10.java		74	11	85	
> Mutant11.java		76	7	83	
> Mutant12.java		74	7	81	
> Mutant3.java		72	7	79	
> Mutant4.java		66	7	73	
> Mutant8.java		72	7	79	
> Mutant9.java		72	7	79	
> Mutant5.java		65	5	70	
> Mutant14.java		74	2	76	
> Mutant15.java		113	2	115	
> Mutant7.java		47	2	49	
> Mutant13.java		113	0	113	
> Mutant2.java		75	0	75	
> Mutant6.java		81	0	81	
> testthetests		297	46	343	
> revealer		384	0	384	

# Measuring Code Coverage

- Code coverage tool built into Eclipse
  - EclEmma
- More on this in the final project

# Uses of Coverage Criteria

- “Stopping” rule → sufficient testing
  - Avoid unnecessary, redundant tests
- Measure test quality
  - Dependability estimate
  - Confidence in estimate
- Specify test cases
  - Describe additional test cases needed

# Coverage Criteria Discussion

- Is it always possible for a test suite to cover all the statements in a given program?
  - No. Could be infeasible statements
    - Unreachable code
    - Legacy code
    - Error handling code that should not typically happen
    - Configuration that is not on site
- Do we need the test suite to cover 100% of statements/branches to believe it is adequate?
  - 100% coverage does not mean correct program
  - But  $< 100\%$  coverage does mean testing inadequacy

# True/False Quiz

- A program that passes all test cases in a test suite with 100% path coverage is bug-free.

➤ **False.**

➤ **Examples:**

- The test suite may cover a faulty path with data values that don't expose the fault.
  - Towards Exhaustive Testing
- Errors of omission
  - Missing a whole if

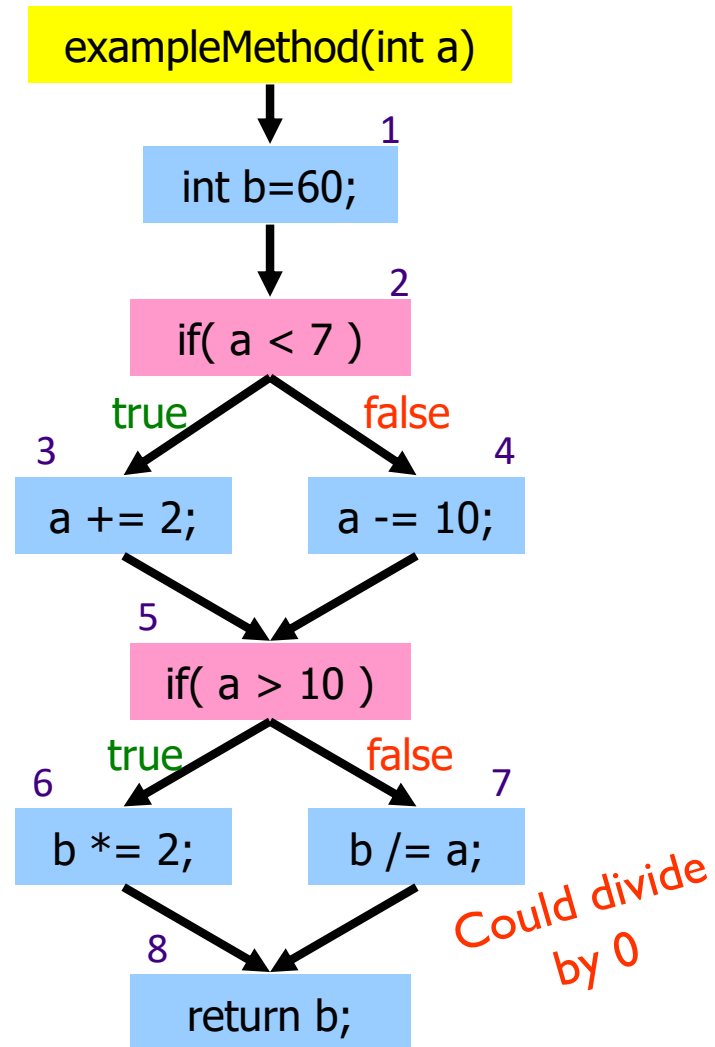
# Example

Test Suite:

3-7: a=3  
4-6: a=30  
3-6: a=6  
4-7: a=9

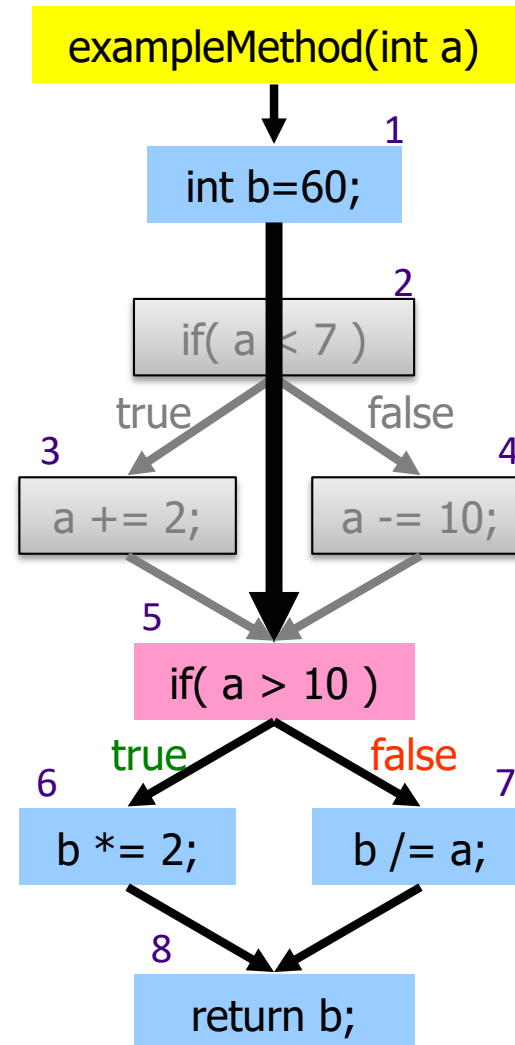
But, error shows up with

3-7: a=0  
4-7: a=10



# Omission Example

Consider if the first `if` block wasn't in the code.  
You could cover all the paths, but you're missing a crucial condition.





## True/False Quiz

- When you add test cases to a test suite that covers all statements so that it covers all branches, the new test suite is more likely to be better at exposing faults.

➤ **True.**

➤ You're adding test cases and covering new paths, which may have faults.

# Which Test Suite Is Better?

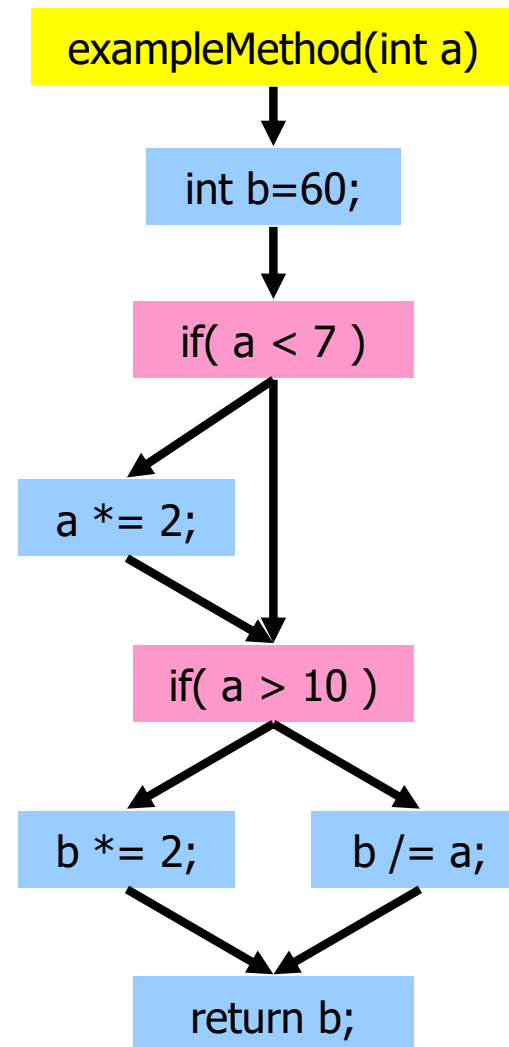
Statement-  
adequate  
Test Suite

Branch-  
adequate  
Test Suite

- Branch-adequate suite is not *necessarily* better than Statement-adequate suite
  - Statement-adequate suite could cover buggy paths and include input value tests that Branch-adequate suite doesn't

# Example

- TS1 (Statement-Adequate):
  - $a=0, 6$
- TS2 (Branch-Adequate):
  - $a=3, 30$
- Statement-adequate will find fault but branch-adequate won't
  - Covers the path that exposes the fault



# Software Testing: When is Enough Enough?

- Need to decide when tested enough
  - Balance goals of releasing application, high quality standards
- Can use program coverage as “stopping” rule
  - Also measure of confidence in test suite
  - Statement, Branch, Path and their tradeoffs
  - Use coverage tools to measure statement, branch coverage
- Still, need to use some other “smarts” besides program coverage for creating test cases

# No Silver Bullet

- Recall the Fred Brooks' quote:
  - “There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity.”
  - Known as “no silver bullet”
- Test coverage is one tool that will help us improve the quality of our code, but it will not solve everything

## Productive Use of Time that isn't Coding

- “Most programmers regard anything that doesn't generate code to be a waste of time. Thinking doesn't generate code, and writing code without thinking is a recipe for bad code. Before we start to write any piece of code, we should understand what that code is supposed to do. Understanding requires thinking, and thinking is hard.”
- In the words of the cartoonist Dick Guindon:  
“Writing is nature's way of letting you know how sloppy your thinking is.”

Source: <http://www.wired.com/opinion/2013/01/code-bugs-programming-why-we-need-specs>