

Objectives

- Coverage, Testing wrap up
- Design in the Small

1

Review

1. What is our git workflow when we're collaborating with teammates?
 - Both variations (why 2 variations?)
2. How should teams work together for success?
3. What is code coverage?
4. What is code coverage *criteria*?
 - Provide examples of code coverage criteria

2

Review:

Collaboration: Workflow – Seeking Feedback

1. Create a branch for your work from main
 - Commit periodically
 - Write descriptive comments so your team members know what you did and why
2. Push your branch
3. Open a **Pull Request** on your branch in GitHub
 - You can tag your teammates to let them know that you've completed your work
 - Team: discuss and review potential changes – can still update
4. Merge pull request into main branch (when ready)
5. Pull the main branch to get the latest code
 - May want to merge main into your branch

Nov 4, 2022

Sprenkle - CSCI209

3

3

Review: Collaboration: Workflow

1. Create a branch for your work
 - Commit periodically
 - Write descriptive comments so your team members know what you did and why
2. Switch to main
3. Pull main branch
4. Merge your branch into the main branch
 - Handle merge conflicts
 - Commit
5. Push main branch

Nov 4, 2022

Sprenkle - CSCI209

4

4

Culture Eats Strategy for Breakfast

- Your actions should match what your team says
are your squad goals

5

Review: Code Coverage

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

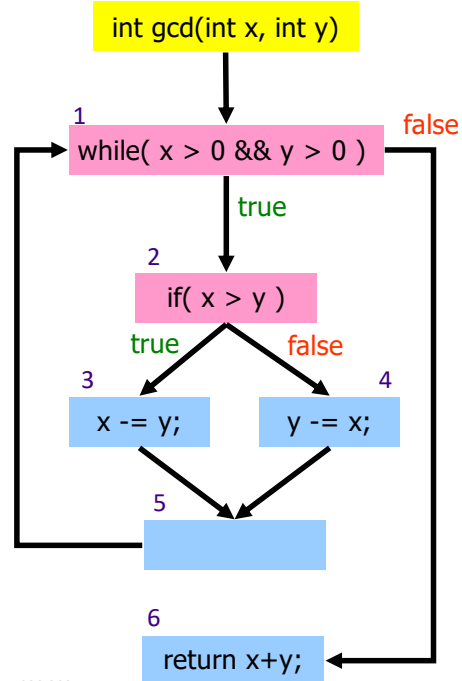
6

Path Coverage

- How many paths through this method?

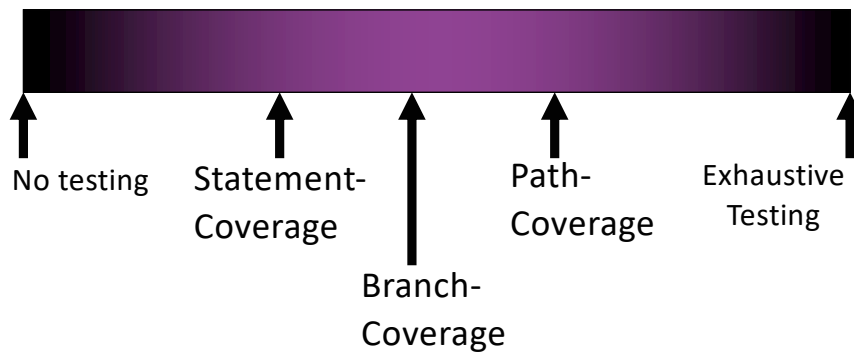
➤ Too many to count, test them all!

- 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



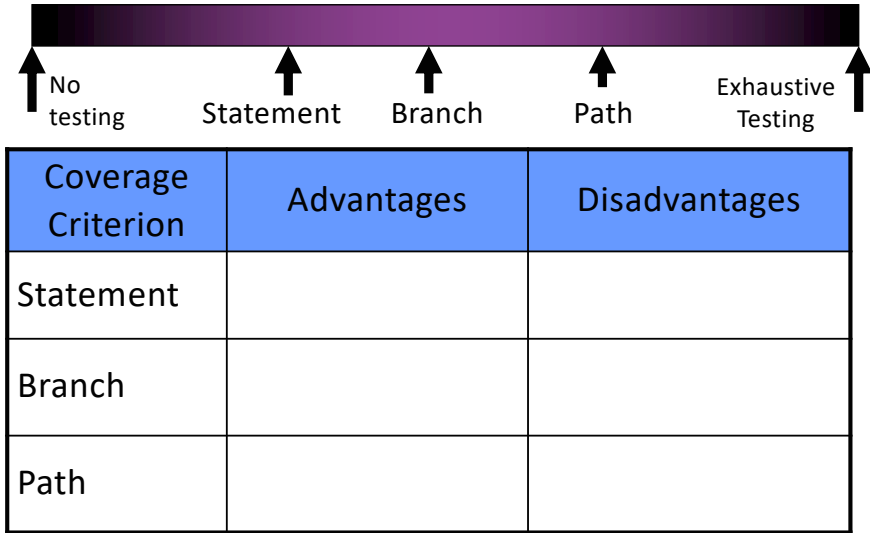
7

Testing Continuum



8

Comparison of Coverage Criteria



Nov 4, 2022

Sprenkle - CSCI209

9

9

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

Nov 4, 2022

Sprenkle - CSCI209

10

10

How Can We Use Coverage Criteria?

Nov 4, 2022

Sprenkle - CSCI209

11

11

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

Nov 4, 2022

Sprenkle - CSCI209

12

12

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
 - 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

Nov 4, 2022

Sprenkle - CSCI209

13

13

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

Nov 4, 2022

Sprenkle - CSCI209

14

14

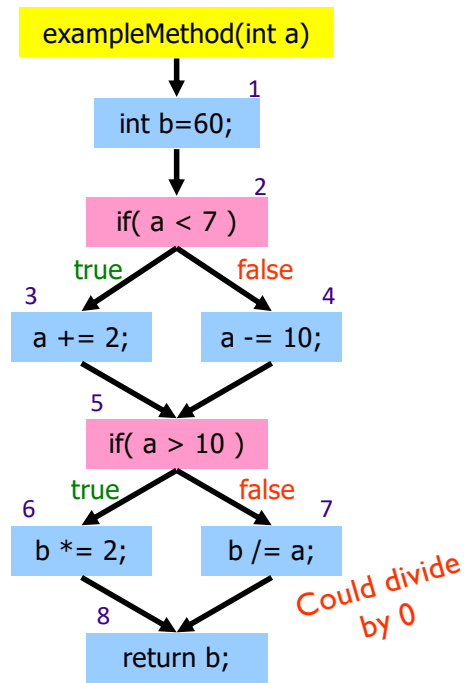
Example

Test Suite:

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

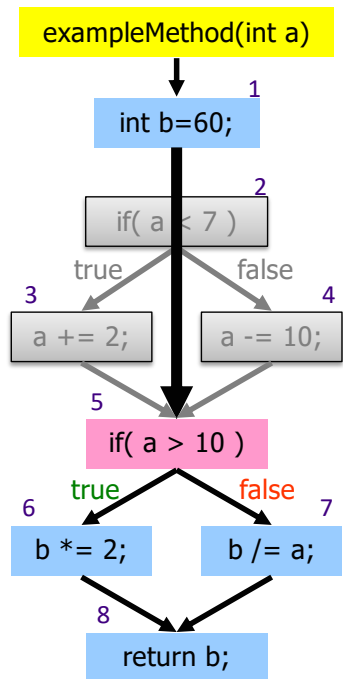
But, error shows up with

- 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.

Nov 4, 2022

Sprenkle - CSCI209

17

17

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

Nov 4, 2022

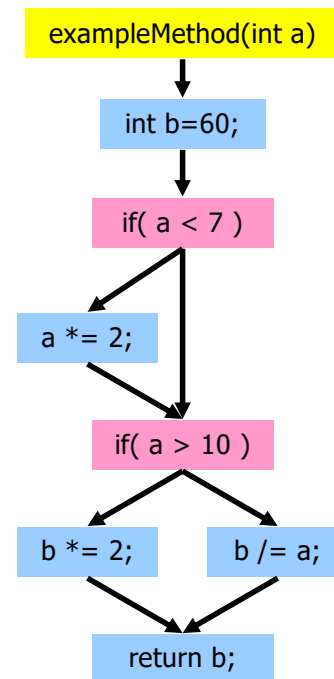
Sprenkle - CSCI209

18

18

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



Nov 4, 2022

Sprenkle - CSCI209

19

19

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

Nov 4, 2022

Sprenkle - CSCI209

20

20

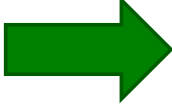
No Silver Bullet

- 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

OBJECT-ORIENTED DESIGN PRINCIPLES

Designing Systems

All systems **change** during their life cycle

- Requirements change
 - Misunderstandings in requirements
 - New functionality
- 
- Code must be **soft**
 - Flexible
 - Easy to change
 - New or revised circumstances
 - New contexts
 - Fix bugs

Nov 4, 2022

Sprenkle - CSCI209

23

23

Designing for Change Example

- July 2010, Oracle released Java 6 update 21
 - Generated java.dll replaced
 - COMPANY_NAME=Sun Microsystems, Inc. with
 - COMPANY_NAME=Oracle Corporation
- Change caused OutOfMemoryError during Eclipse launch
 - Eclipse versions 3.3-3.6 (widespread!)
 - Why? Eclipse used the company name in the DLL in startup (runtime parameters) on Windows
- Temporary Fix: Oracle changed name back
- Required changes to all Eclipse versions

Nov 4, 2022

Source: <http://www.infoq.com/news/2010/07/eclipse-java-6u21>

24

24

Designing Systems

All systems **change** during their life cycle

- Questions to consider:
 - How can we create designs that are stable in the face of change?
 - How do we know if our designs aren't maintainable?
 - What can we do if our code isn't maintainable?
- Answers will help us
 - Design our own code
 - Understand others' code

Nov 4, 2022


Sprenkle - CSCI209

25

25

Designing Systems

All systems **change** during their life cycle

- Questions to consider:
 -  **How can we create designs that are stable in the face of change?**
 - How do we know if our designs aren't maintainable?
 - What can we do if our code isn't maintainable?
- Answers will help us
 - Design our own code
 - Understand others' code

Nov 4, 2022

Sprenkle - CSCI209

26

26

Best Practices Overview

- (DRY): Don't repeat yourself
- Shy Code, Avoid Coupling
- Tell, Don't Ask
- Avoid code smells
- SOLID
 - Single Responsibility Principle
 - Open-closed principle
 - Liskov Substitution Principle
 - Interface Segregation Principle
 - Dependency Inversion Principle

A lot of related fundamental principles.
We have been using them/applying them,
just haven't named them.

Nov 4, 2022

Sprenkle - CSCI209

27

27

Don't Repeat Yourself (DRY): Knowledge Representation

Every piece of knowledge must have a single, unambiguous, and authoritative representation within a system

- **Intuition:** when need to change representation, make in only one place
- Requires planning
 - What data needed, how represented (e.g., type)
 - Consider documentation as well

Nov 4, 2022

Sprenkle - CSCI209

28

28

Don't Repeat Yourself (DRY): Knowledge Representation

Every piece of knowledge must have a single, unambiguous, and authoritative representation within a system

- Example:

- **Car** class defined constants for gears
- **CarTest** should refer to those constants
 - Not redefine those gears, nor just hardcode numbers
 - The values are likely to change, so refer to the variables.

Nov 4, 2022

Sprenkle - CSCI209

29

29

Don't Repeat Yourself (DRY): Knowledge Representation

Every piece of knowledge must have a single, unambiguous, and authoritative representation within a system

- Example:

- **Birthday** class had a month
 - Could be represented as a number and a String
- **Best:** represent as a number (only), i.e., only one instance variable to represent the month
 - Get month String from the number (e.g., MONTHS_OF_YEAR[month-1])
- **Why?**

Nov 4, 2022

Sprenkle - CSCI209

30

30

Don't Repeat Yourself (DRY): Knowledge Representation

Every piece of knowledge must have a single, unambiguous, and authoritative representation within a system

- Example:

- **Birthday** class had a month
 - Could be represented as a number and as a String
- Best: represent as a number (only), i.e., only one instance variable to represent the month
 - Get month String from the number (e.g., MONTHS_OF_YEAR[month-1])
- Why? If need to update the month, just one variable needs to be updated, not two, which *can get out of sync*

Nov 4, 2022

Sprenkle - CSCI209

31

31

Shy Code

- Goal: Won't reveal *too much* of itself
- Otherwise: get *coupling*
 - Coupling: dependence on other code
 - Static, dynamic, domain, temporal

What techniques have we discussed for how to keep our code shy?

- Coupling isn't always bad...
 - Can't be completely avoided...
 - We want *shy* code – not completely isolated code

Nov 4, 2022

Sprenkle - CSCI209

32

32

Achieving Shy Code

- Private instance variables
 - Especially mutable fields
- Make classes public only when need to be public
 - i.e., accessible by other classes → part of API
- Getter methods shouldn't return private, mutable state/objects
 - Use `clone()` before returning

How can you make any field immutable?

Nov 4, 2022

Sprenkle - CSCI209

33

33

Coupling Overview

- Interdependence of classes
 - Dependence makes class susceptible to breaking if other class changes
- Class A is *coupled* with class B if class A
 - Has an object of type B
 - Instance variable, Parameter, return type
 - Calls on methods of object B
 - Is a child class of or implements B
- Goal: *Loose* coupling
 - Non-goal: no coupling

Nov 4, 2022

Sprenkle - CSCI209

34

34

Static Coupling

- Code requires other code to compile
- Clearly, we need some static coupling!
 - Example: to display a line of text, we need the code for `System.out`
- Problem if you include more than you need

Nov 4, 2022

Sprenkle - CSCI209

35

35

Static Coupling

- Code requires other code to compile
- Problem if you include more than you need
 - Example: poor use of inheritance
 - Brings excess baggage
 - Inheritance is reserved for “is-a” relationships
 - Base class should not include optional behavior
 - Not “uses-a” or “has-a”
- Solution: use *composition* or *delegation* instead

Nov 4, 2022

Sprenkle - CSCI209

36

36

Static Coupling

- Code requires other code to compile
- Problem if you include more than you need
- Solution: use *composition* or *delegation* instead
 - Example: I created a class where I have keys associated with values. I shouldn't extend HashMap, but **use** a HashMap
 - Example: GamePiece class did not and *should not* include *chase* functionality
 - Only certain child classes need that functionality

Nov 4, 2022

Sprenkle - CSCI209

37

37

Tell, Don't Ask

- When designing methods, think of them as *sending a message*
 - Send a message
 - Get a response
- Method call: 1) sends a request to do something; 2) response is what is returned
 - Don't ask about details
 - Black-box, encapsulation, information hiding
- Example: hasSameBirthday(Birthday[] birthdays)
 - Input: the array of birthdays to the method
 - Output: true/false if two people had the same birthday
 - Don't need to know how it was determined; no printing of output

Nov 4, 2022

Sprenkle - CSCI209

40

40

Looking Ahead

- Today: Science Advisory Board!
- Tuesday, 11:45 a.m. – lunch and talk!
 - Andy Ramlatchan
 - Check your email from Carolyn
- Wednesday: Testing project
- Friday-Sunday: Exam 2