

CS 520

Theory and Practice of Software Engineering
Fall 2026

Software testing

1

Today

Introduction to software testing

- Blackbox vs. whitebox testing
- Unit testing (vs. integration vs. system testing)
- Test adequacy
 - Structural code coverage
 - Statement coverage
 - Decision coverage
 - Condition coverage
 - Mutation analysis

2

Software testing

What can testing do, and what can't it do?

Software testing can **show** the **presence of defects**,
but never show their absence! (Edsger W. Dijkstra)

- A good test is one that fails because of a defect.

How do we come up with good tests?

3

Two strategies: black box vs. white box

Black box testing

- The system is a black box (can't see inside).
- No knowledge about the internals of a system.
- Create tests solely based on the specification (e.g., input/output behavior).

White box testing

- Knowledge about the internals of a system.
- Create tests based on these internals (e.g., exercise a particular part or path of the system).

4

Unit testing, integration testing, system testing

Unit testing

- Does each unit work as specified?

Integration testing

- Do the units work when put together?

System testing

- Does the system work as a whole?

Our focus: unit testing

5

Unit testing

- A **unit** is the **smallest testable part** of the software system.
- **Goal:** Verify that each software unit performs as specified.
- **Focus:**
 - Individual units (not the interactions between units).
 - Usually input/output relationships.

6

Software testing

Software testing can show the **presence of defects**, but never show their absence! (Edsger W. Dijkstra)

- A good test is one that fails because of a defect.

When should we stop testing if no (new) test fails?

7

Test effectiveness

Ratio of detected defects is the best effectiveness metric!

Problem

- The set of defects is unknowable

Solution

- Use a proxy metric, for example code coverage

8

Structural code coverage: live example

Average of the absolute values of an array of doubles

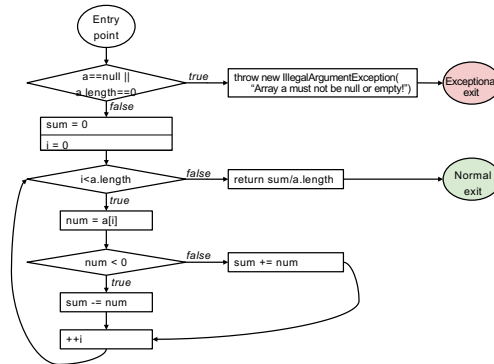
```
public double avgAbs(double ... numbers) {
    // We expect the array to be non-null and non-empty
    if (numbers == null || numbers.length == 0) {
        throw new IllegalArgumentException("Array numbers must not be null or empty!");
    }

    double sum = 0;
    for (int i=0; i<numbers.length; ++i) {
        double d = numbers[i];
        if (d < 0) {
            sum -= d;
        } else {
            sum += d;
        }
    }

    return sum/numbers.length;
}
```

9

Control Flow Graph (CFG)



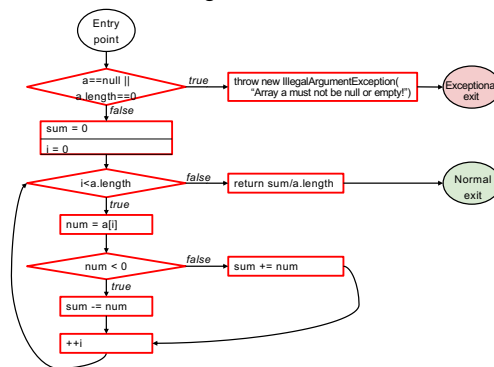
10

Statement coverage

- **Every statement** in the program must be **executed at least once**
- Given the control-flow graph (CFG), this is equivalent to node coverage

11

Statement coverage



12

Condition coverage vs. decision coverage

Terminology

- **Condition:** a boolean expression that cannot be decomposed into simpler boolean expressions.
- **Decision:** a boolean expression that is composed of conditions, using 0 or more logical connectors (a decision with 0 logical connectors is a condition).
- **Example:** if $((x < 5) \ \&\& \ (y > 7)) \{ \dots \}$
 - $(x < 5)$ and $(y > 7)$ are *conditions*.
 - The boolean expression $((x < 5) \ \&\& \ (y > 7))$ is a *decision*.

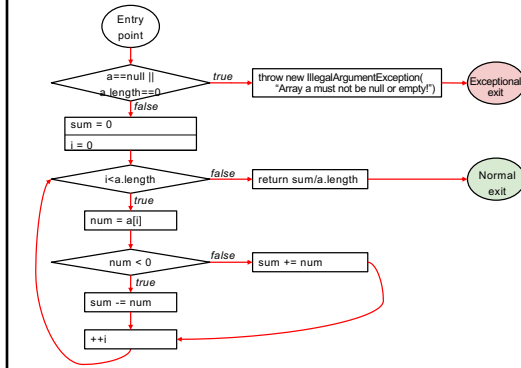
13

Decision coverage (a.k.a. branch coverage)

- **Every decision** in the program must take on **all possible outcomes** (true/false) **at least once**
- Given the CFG, this is equivalent to edge coverage
- **Example:** if $(a > 0 \ \&\& \ b > 0)$
 - $a=1, b=1$
 - $a=0, b=0$

14

decision coverage



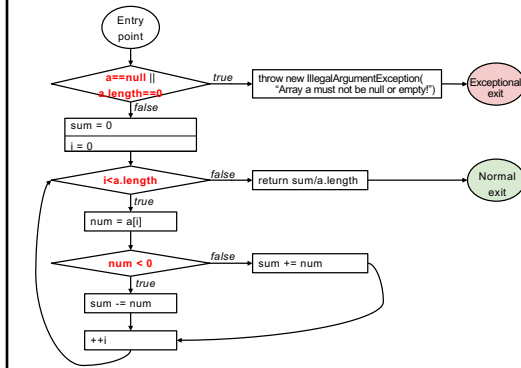
15

Condition coverage

- **Every condition** in the program must take on **all possible outcomes** (true/false) **at least once**
- **Example:** $(a > 0 \ \&\& \ b > 0)$
 - $a=1, b=0$
 - $a=0, b=1$

16

condition coverage



17

subsumption

Given two coverage criteria A and B,
A subsumes B iff **satisfying A implies satisfying B**

- **Subsumption relationships:**
 - Does decision coverage subsume statement coverage?
 - Does decision coverage subsume condition coverage?
 - Does condition coverage subsume decision coverage?

18

Decision coverage vs. condition coverage

4 possible tests for the decision $a \parallel b$:

1. $a = 0, b = 0$
2. $a = 0, b = 1$
3. $a = 1, b = 0$
4. $a = 1, b = 1$

a	b	$a \parallel b$
0	0	0
0	1	1
1	0	1
1	1	1

Satisfies **condition coverage**
but not **decision coverage**

a	b	$a \parallel b$
0	0	0
0	1	1
1	0	1
1	1	1

Satisfies **decision coverage**
but not **condition coverage**

Neither coverage criterion subsumes the other!

19

Structural code coverage: subsumption

Given two coverage criteria A and B,

A subsumes B iff **satisfying A implies satisfying B**

- Subsumption relationships:
 - Decision coverage **subsumes** statement coverage
 - Decision coverage **does not subsume** condition coverage
 - Condition coverage **does not subsume** decision coverage

20

Code coverage: advantages

Classes in this File	Line Coverage	Branch Coverage	Complexity
1 package org;	100%	100%	6
2 public class Avg {			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

- Code coverage is easy to compute.
- Code coverage has an intuitive interpretation.

But, does coverage ensure effective testing?

21

Code coverage: drawbacks

Classes in this File	Line Coverage	Branch Coverage	Complexity
1 package org;	100%	100%	6
2 public class Avg {			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

- Code coverage does not require test assertions.
- Not all statements etc. are equally important.
- Coverage is not the same as behavior.

Are there any alternatives?

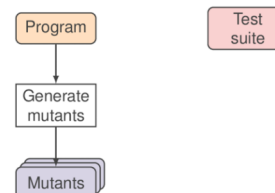
22

Mutation analysis: overview



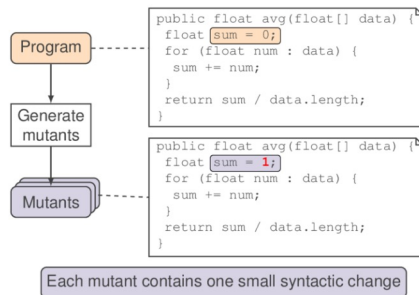
23

Mutation analysis: overview



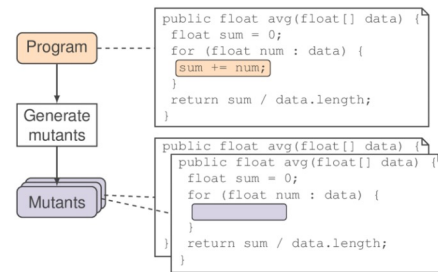
24

Mutation analysis: overview



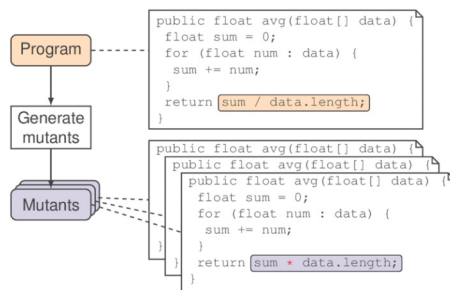
25

Mutation analysis: overview



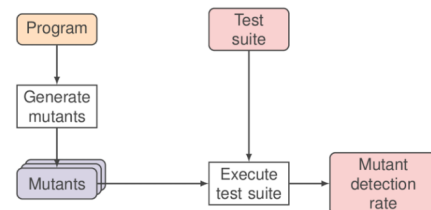
26

Mutation analysis: overview



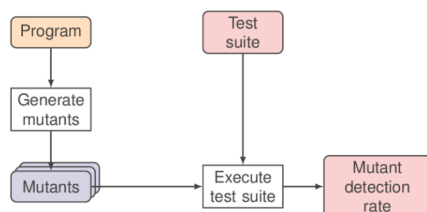
27

Mutation analysis: overview



28

Mutation analysis: overview



Assumption: Mutant detection rate is a good proxy for fault detection rate.

What does it mean for a test to fail on a mutant program?

29

Mutation analysis: example

Find a test case that detects the following mutant
(i.e., passes on the original program but fails on the mutant)

Original program:

```

public int min(int a, int b) {
    return a < b ? a : b;
}

```

a	b	Original	Mutant
1	2	1	1
1	1	1	1
2	1	1	2

Mutant:

```

public int min(int a, int b) {
    return a;
}

```

30

Mutation analysis: another example

Find a test case that detects the following mutant

(i.e., passes on the original program but fails on the mutant)

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

There is no such test that can detect the mutant...

The mutant is **undetectable**

Mutant:

```
public int min(int a, int b) {
    return a <= b ? a : b;
}
```

because it is **equivalent** to the **original program!**

31

Summary

- Testing is an important way to measure code quality
- Black-box testing
- White-box testing
- Coverage metrics
 - Statement
 - Condition
 - Decision
- Mutation-based metric

For more, read:

"Are mutants a valid substitute for real faults in software testing?" in FSE 2014

32