CSE 403

Software Engineering

Winter 2023

Coverage-based Testing

This week: test efficacy and adequacy

- Coverage-based testing
- Mutation-based testing
- In-class exercise

Structural code coverage: motivating 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;
}
```

https://github.com/rjust/testing-ci-gradle

Structural code coverage: motivating example

Classes in this File	Line Coverage	Branch Coverage	Complexity
Avg	100% 10/10	100% 8/8	6

```
 1    package avg;
 2
 3 4  public class Avg {
 4
 5        /*
 6         * Compute the average of the absolute values of an array of doubles
 7         */
 8        public double avgAbs(double ... numbers) {
 9            // We expect the array to be non-null and non-empty
10 4          if (numbers == null || numbers.length == 0) {
11 2              throw new IllegalArgumentException("Array numbers must not be null or empty!");
12            }
13
14 2          double sum = 0;
15 8          for (int i=0; i<numbers.length; ++i) {
16 6              double d = numbers[i];
17 6              if (d < 0) {
18 2                  sum -= d;
19              } else {
20 4                  sum += d;
21              }
22            }
23 2          return sum/numbers.length;
24        }
25   }
```

(Cobertura's Code coverage report.)

https://github.com/rjust/testing-ci-gradle

Code coverage metrics

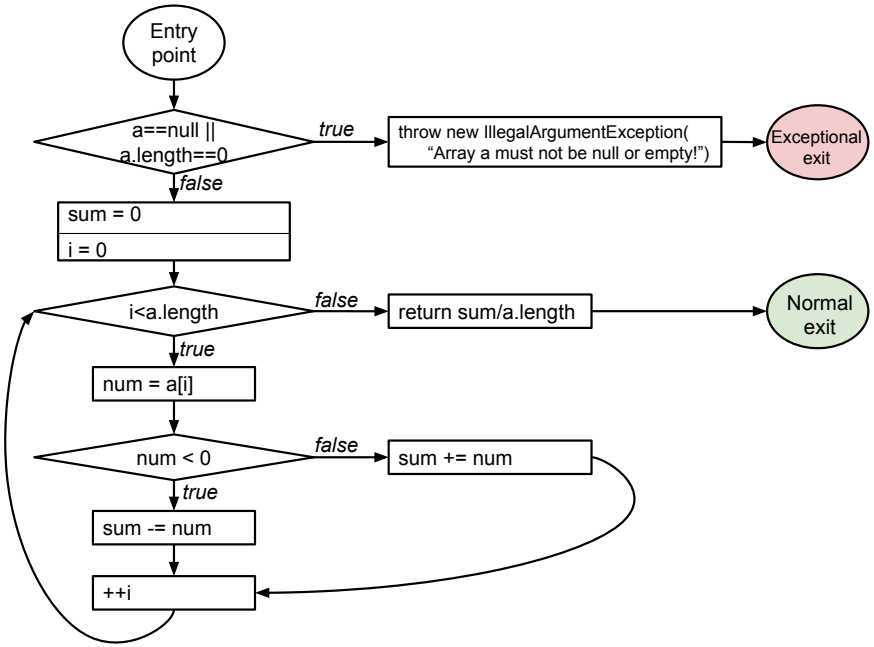
Structural code coverage: the basics

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;
}
```

What's the control flow graph (CFG) for this method?

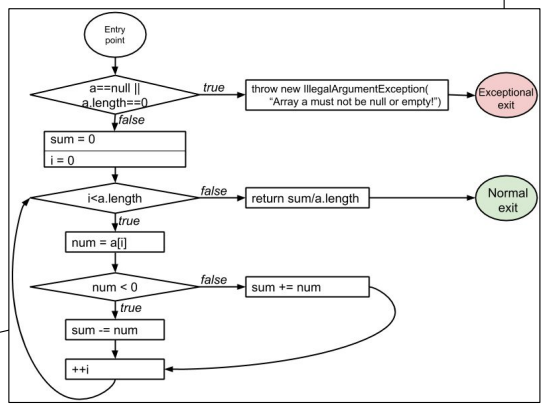
Structural code coverage: the basics



Structural code coverage: the basics

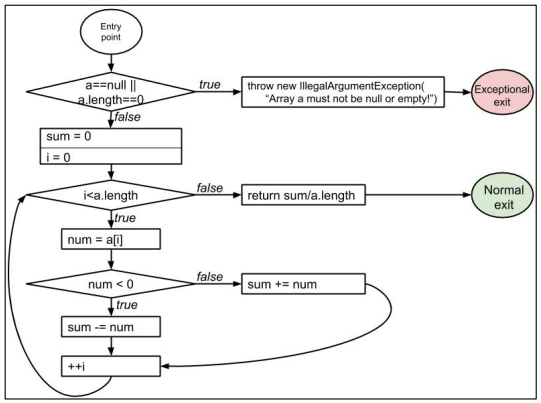
Average of the absolute values of an array of doubles

```
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;
}
```

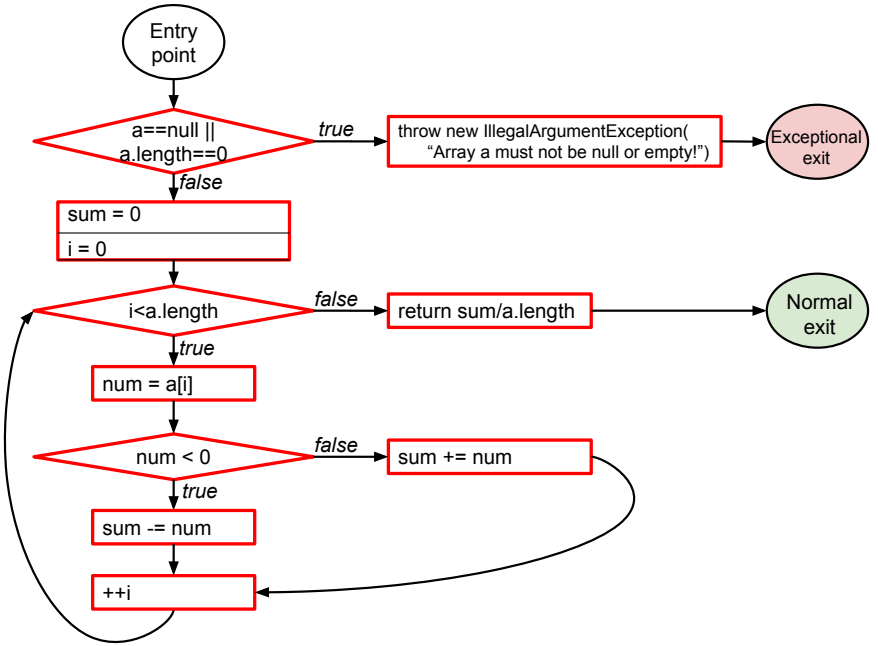


Statement coverage

- **Every statement** in the program must be **executed at least once.**

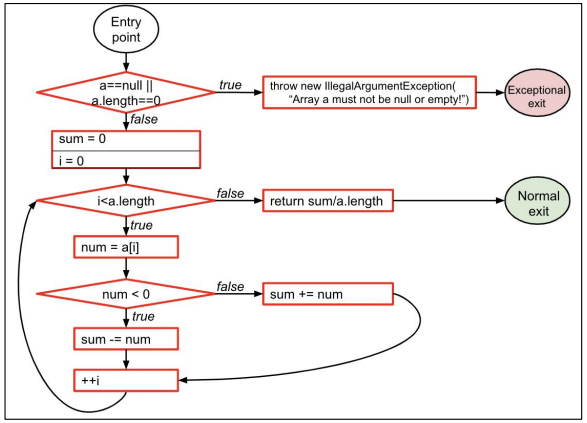


Statement coverage



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.



Condition coverage vs. decision coverage

Terminology

- **Condition**: a boolean expression that cannot be decomposed into simpler boolean expressions (atomic).
- **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 (*a* | *b*) { … }
  - *a* and *b* are *conditions.*
  - The boolean expression *a* | *b* is a *decision.*

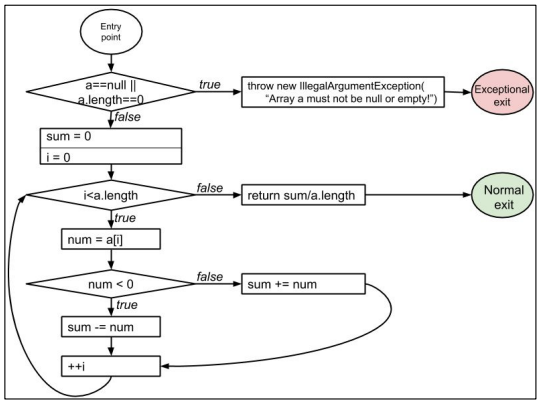
Condition coverage vs. decision coverage

Terminology

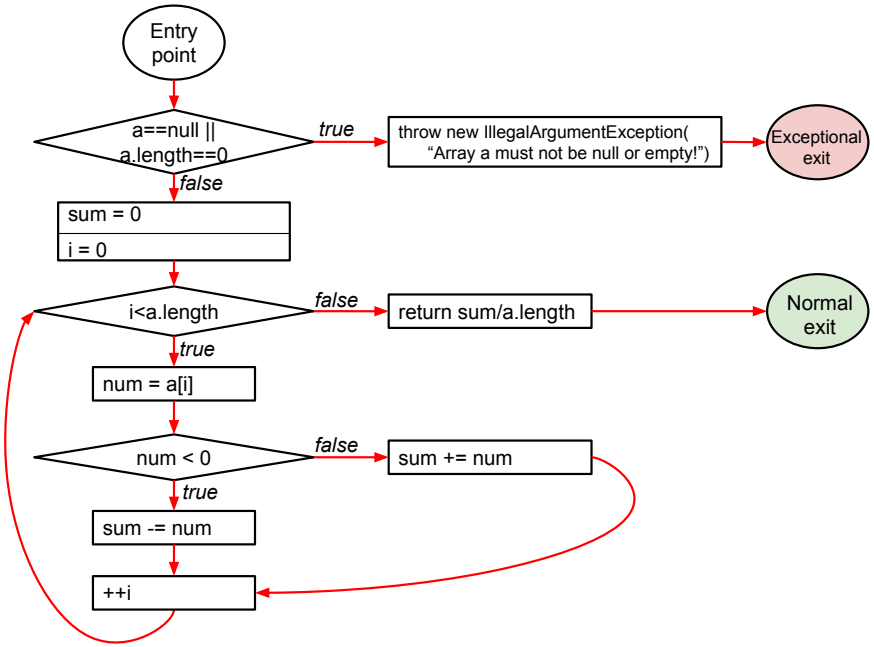
- **Condition**: a boolean expression that cannot be decomposed into simpler boolean expressions (atomic).
- **Decision**: a boolean expression that is composed of conditions, using 0 or more logical connectors (a decision with 0 logical connectors is a condition).
- **Example:** if (*a* | *b*) { … }
  - *a* and *b* are *conditions.*
  - The boolean expression *a* | *b* is a *decision.*

Decision coverage

- **Every decision** in the program must take on **all possible outcomes** (true/false) **at least once.**

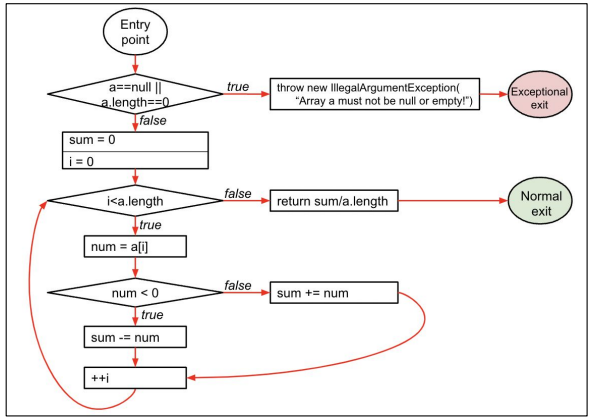


Decision coverage



Decision 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.



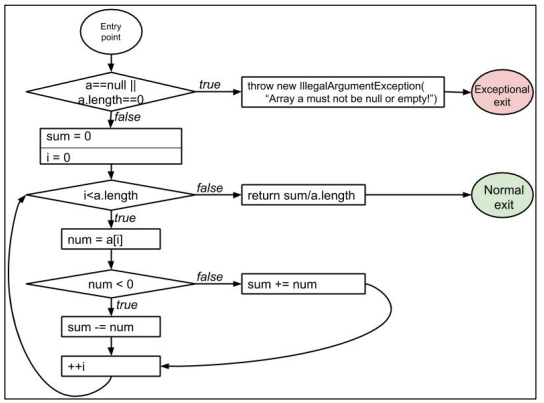
Condition coverage vs. decision coverage

Terminology

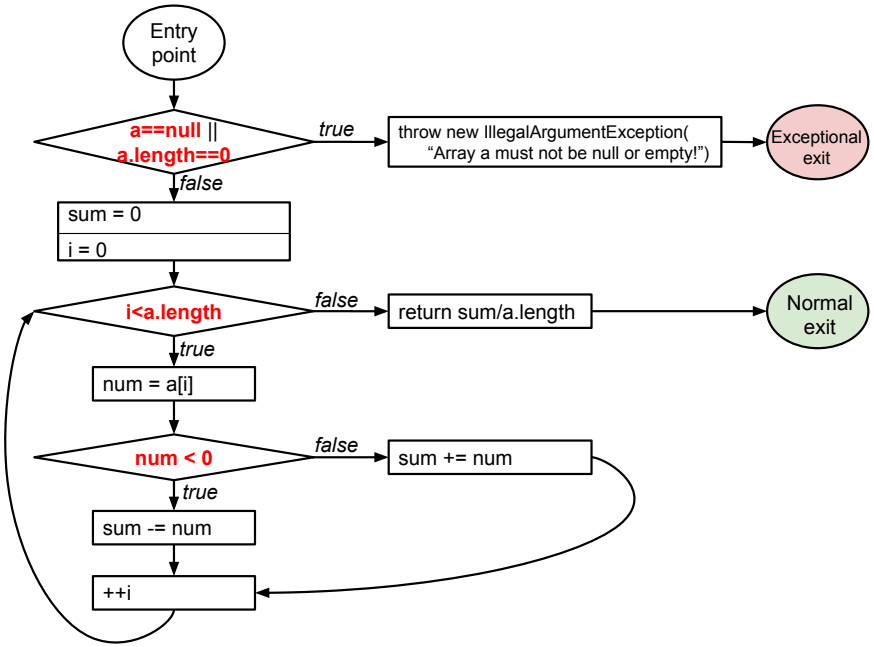
- **Condition**: a boolean expression that cannot be decomposed into simpler boolean expressions (atomic).
- **Decision**: a boolean expression that is composed of conditions, using 0 or more logical connectors (a decision with 0 logical connectors is a condition).
- **Example:** if (*a* | *b*) { … }
  - *a* and *b* are *conditions.*
  - The boolean expression *a* | *b* is a *decision.*

Condition coverage

- **Every condition** in the program must take on **all possible outcomes** (true/false) **at least once.**

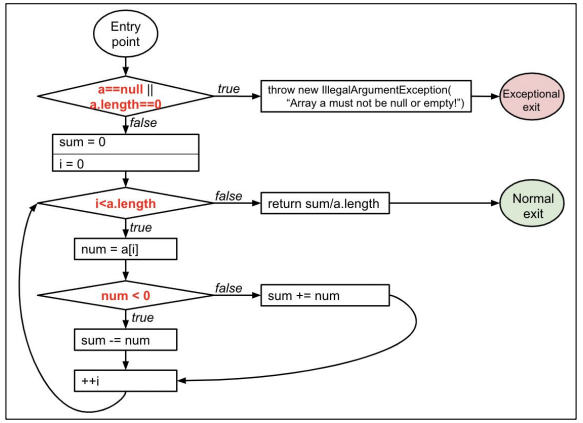


Condition coverage



Condition coverage

- **Every condition** in the program must take on **all possible outcomes** (true/false) **at least once.**



Structural code coverage: subsumption

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

- Subsumption relationships:
  1. Does statement coverage subsume decision coverage?
  2. Does decision coverage subsume statement coverage?
  3. Does decision coverage subsume condition coverage?
  4. Does condition coverage subsume decision coverage?

https://pollev.com/renejust859

Structural code coverage: subsumption

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

- Subsumption relationships:
  1. **Statement** coverage **does not subsume decision** coverage
  2. **Decision** coverage **subsumes statement** coverage
  3. **Decision** coverage **does not subsume condition** coverage
  4. **Condition** coverage **does not subsume decision** coverage

There are more coverage criteria, including MC/DC.
(MC/DC is required for safety-critical systems -- DO-178B/C.)

Decision coverage vs. condition coverage

4 possible tests for the decision *a* | *b*:

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

a	*b*	*a* | *b*
0	0	0
0	**1**	**1**
1	**0**	**1**
1	1	1

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

a	*b*	*a* | *b*
0	**0**	**0**
0	**1**	**1**
1	0	1
1	1	1

Does **not** satisfy **condition coverage** but **decision coverage**

Neither coverage criterion subsumes the other!

Modified Condition/Decision Coverage (MC/DC)

MCDC: Modified condition and decision coverage

- **Every decision** in the program must take on **all possible outcomes** (true/false) **at least once**
- **Every condition** in the program must take on **all possible outcomes** (true/false) **at least once**
- **Each condition** in a decision has been shown to **independently affect** that decision's **outcome**.
  (A condition is shown to independently affect a decision's outcome by: varying just that condition while holding fixed all other possible conditions.)

Required for safety critical systems (DO-178B/C)

MC/DC: an example

```
if (a | b)
```

a	b	Outcome
0	0	0
0	1	1
1	0	1
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Which tests (combinations of a and b) satisfy MCDC?

MC/DC: an example

```
if (a | b)
```

a	b	Outcome
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

MCDC is still cheaper than testing all possible combinations.

MC/DC: another example

```
if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
1	0	1
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Why is this example different?

MC/DC: another example

```
if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
1	--	1
1	--	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Short-circuiting operators may not evaluate all conditions.

MC/DC: yet another example

```
if (!a) ... if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
1	0	1
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

What about this example?

MC/DC: another example

```
if (!a) ... if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
X	X	X
X	X	X

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Not all combinations of conditions may be possible.

MCDC: complex expressions

Provide an MCDC-adequate test suite for:

1. `a | b | c`
2. `a & b & c`

a | b | c

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

a & b & c

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Structural code coverage: summary

Classes in this File	Line Coverage	Branch Coverage	Complexity
Avg	100% 10/10	100% 8/8	6

```
1  package avg;
2
3  public class Avg {
4
5      /*
6       * Compute the average of the absolute values of an array of doubles
7       */
8      public double avgAbs(double ... numbers) {
9          // We expect the array to be non-null and non-empty
10         if (numbers == null || numbers.length == 0) {
11             throw new IllegalArgumentException("Array numbers must not be null or empty!");
12         }
13
14         double sum = 0;
15         for (int i=0; i<numbers.length; ++i) {
16             double d = numbers[i];
17             if (d < 0) {
18                 sum -= d;
19             } else {
20                 sum += d;
21             }
22         }
23         return sum/numbers.length;
24     }
25 }
```

- Code coverage is easy to compute.
- Code coverage has an intuitive interpretation.
- Code coverage in industry: Code coverage at Google
- Code coverage itself is not sufficient!