CSE 403

Software Engineering

Spring 2023

#16: Coverage-based Testing

Logistics

WEEK 6

05/01	L: Test Coverage	
05/02	T:	DUE: TCC!!!
05/03	L: Mutation Testing	Alpha Release (R1)
05/04	P:	
05/05	LX: Code Defenders	

This week: test efficacy and adequacy

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

Structural code coverage: motivating example

Structural code coverage: motivating example

```
[tool.hatch.envs.test]
dependencies = [
  "coverage[toml]",
  "pytest"
]


[tool.coverage.run]
source = ["litw"]
omit = ["**/test*"]



[tool.hatch.envs.test.scripts]
run-coverage = "coverage run -m pytest; coverage report"
run = "pytest"
```

Structural code coverage: motivating example

Structural code coverage: motivating example

Coverage report: 92%

coverage.py v7.2.5, created at 2023-04-30 14:16 -0700

Module	statements	missing	excluded	coverage
src/litw/__init__.py	0	0	0	100%
src/litw/api/__about__.py	1	1	0	0%
src/litw/api/__init__.py	0	0	0	100%
src/litw/api/api.py	45	4	0	91%
src/litw/api/data/__init__.py	0	0	0	100%
src/litw/api/data/model.py	28	0	0	100%
src/litw/api/data/mongo.py	90	0	0	100%
src/litw/api/security.py	92	15	0	84%
src/litw/api/tests/__init__.py	0	0	0	100%
src/litw/api/util.py	14	2	0	86%
src/litw/settings.py	11	0	0	100%
Total	**281**	**22**	**0**	**92%**

coverage.py v7.2.5, created at 2023-04-30 14:16 -0700

Structural code coverage: motivating example

```
31  @app.post("/studies")
32  async def post_study(study_name: str, user: dict = Depends(user_authorization)):
33      return {}
34
35
36  @app.get("/studies/{study_id}")
37  async def get_studies(study_id: UUID, study: dict = Depends(study_authorization)):
38      if study['id'] == str(study_id):
39          return study
40      else:
41          raise HTTPException(
42              status_code=status.HTTP_401_UNAUTHORIZED,
43              detail="You don't have access to the study: {}.".format(study_id)
44          )
45
46
47  @app.post("/studies/{study_id}/data")
48  async def post_study_data(study_id: UUID, study_data: dict, study: dict = Depends(stuc
49      if study['id'] == str(study_id):
50          data_access = DataAccessFactory()
51          study_data_access: StudyDataMongo = data_access.access_points[data_access.avai
52          result = study_data_access.add_data(str(study_id), study_data)
53          return result
54      else:
55          raise HTTPException(
56              status_code=status.HTTP_401_UNAUTHORIZED,
57              detail="You don't have access to the study: {}.".format(study_id)
58          )
59
```

Code coverage metrics

- Statement coverage
- Branch coverage
  - Condition coverage
  - Decision coverage
  - Modified Condition/Decision coverage

Structural code coverage: the basics

Average of the absolute values of an array of doubles

```java
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 coverag

Average of the absolute val

```
public double avgAbs(double ... numbers)

  // We expect the array to be non-null
  if (numbers == null || numbers.length
    throw new IllegalArgumentException('
  }

  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

Average of the absolute values of an array of doubles

```java
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;
}
```

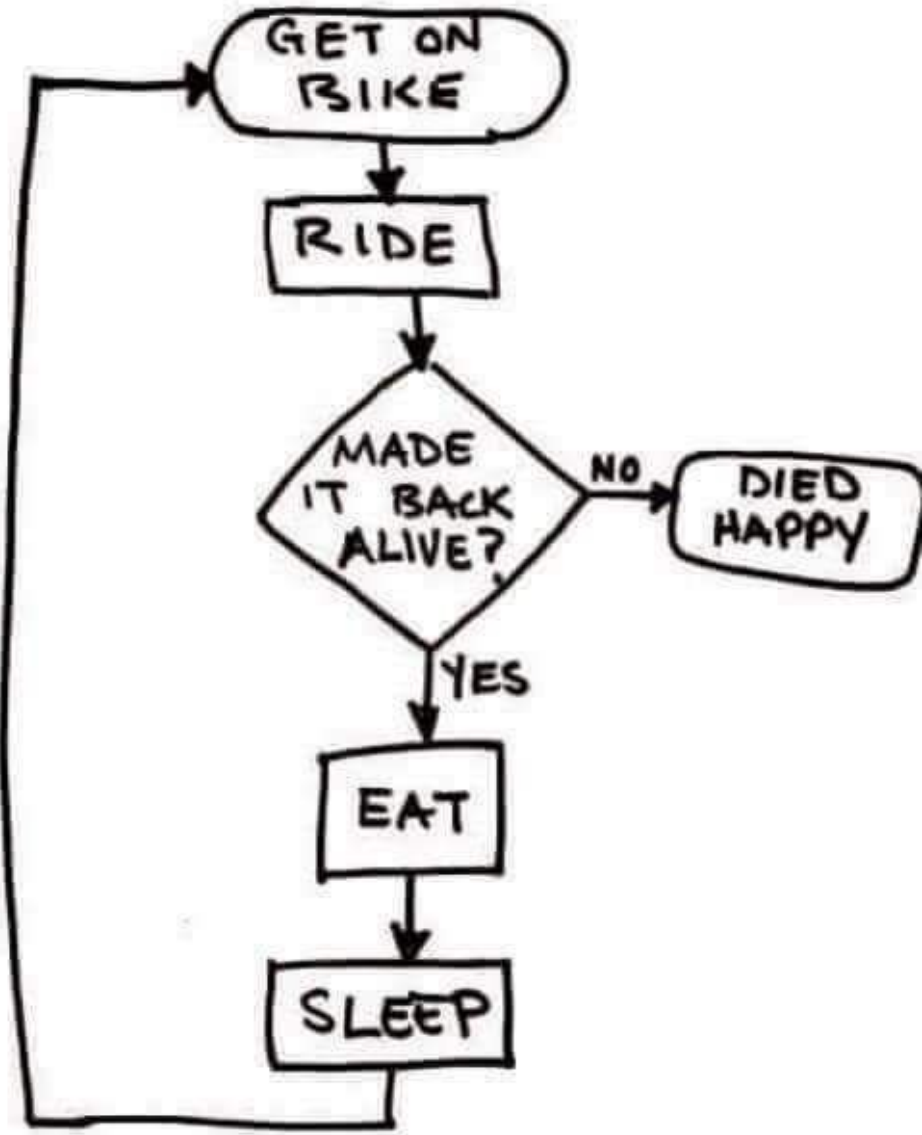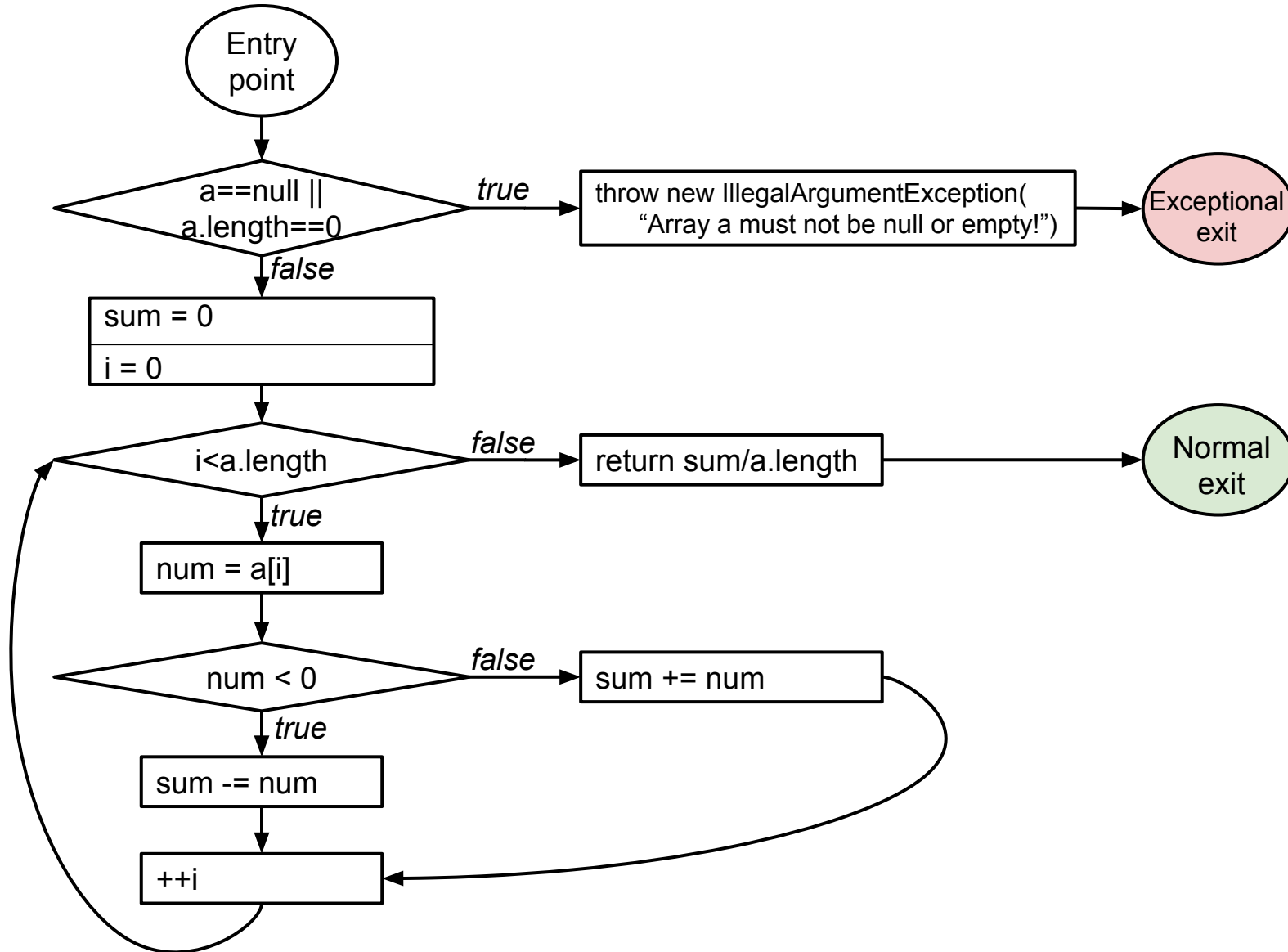**What's the control flow graph (CFG) for this method?**

Structural code coverage: the basics

Structural code coverage: the basics

Average of the absolute values of an array of doubles

```java
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;
}
```
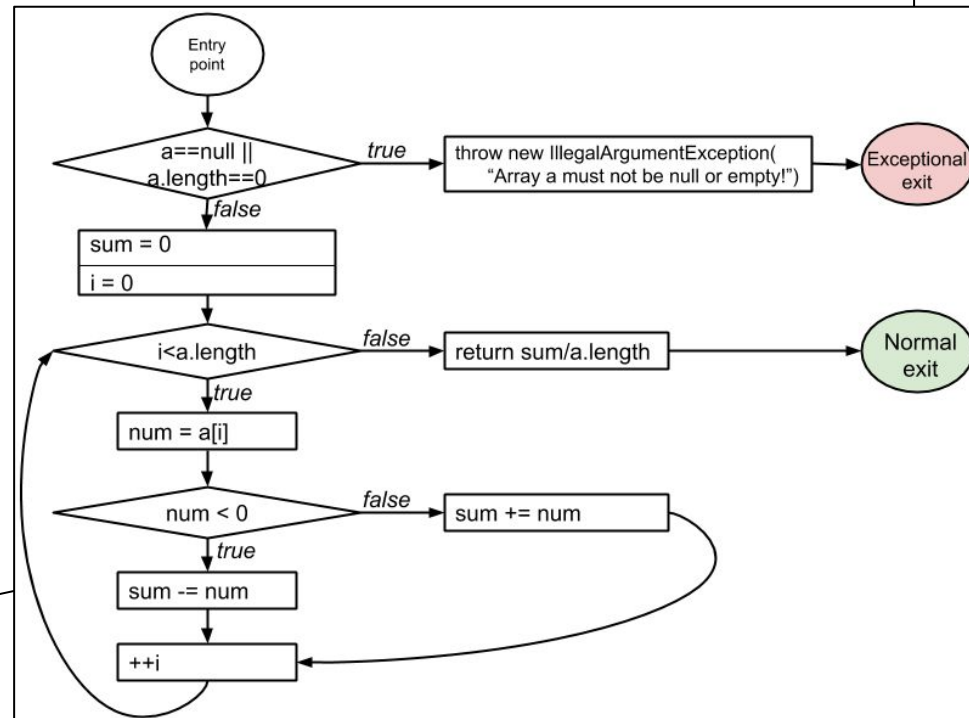
Statement coverage

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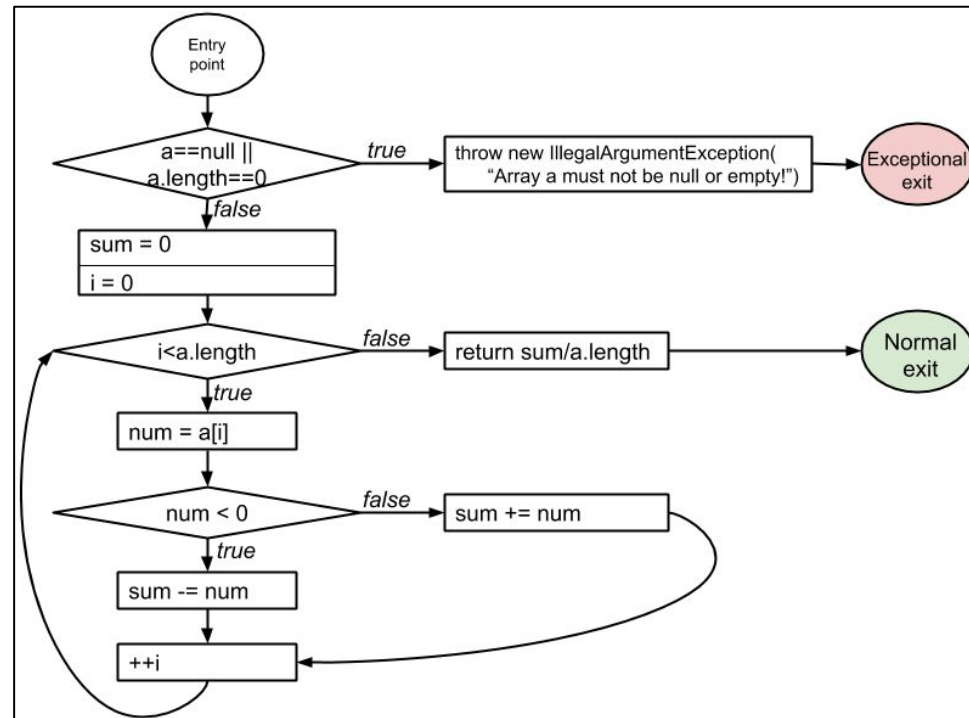
Statement coverage
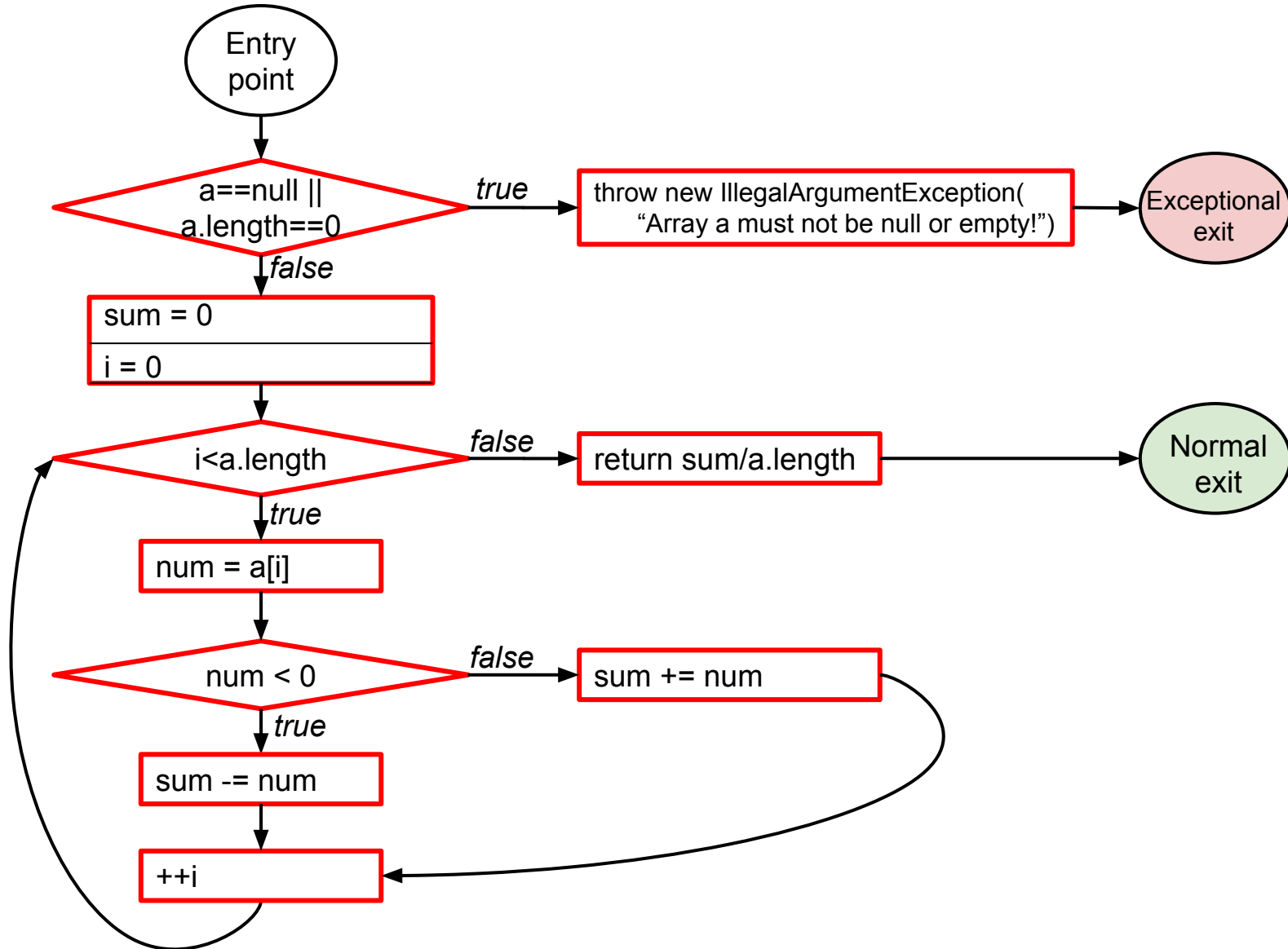
Statement coverage

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

- Given the control-flow graph (CFG), this is equivalent to node coverage.
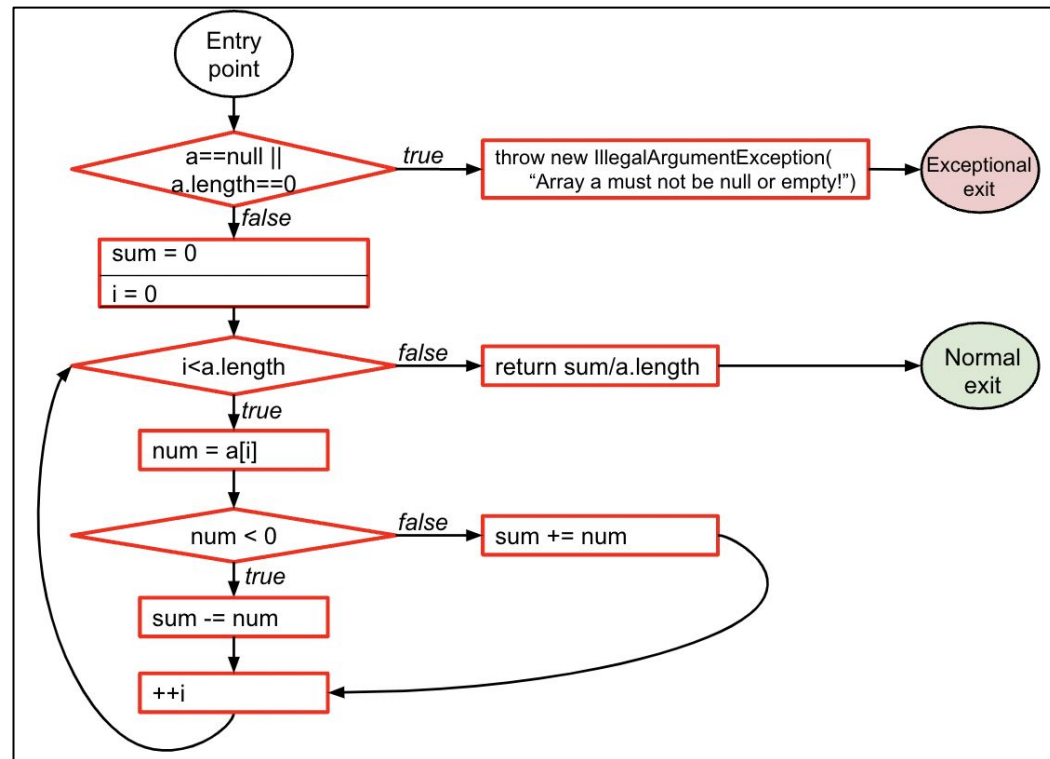
Branch coverage: Condition vs. Decision

Branch coverage: Condition vs. Decision

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.*

Branch coverage: Condition vs. Decision

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.**
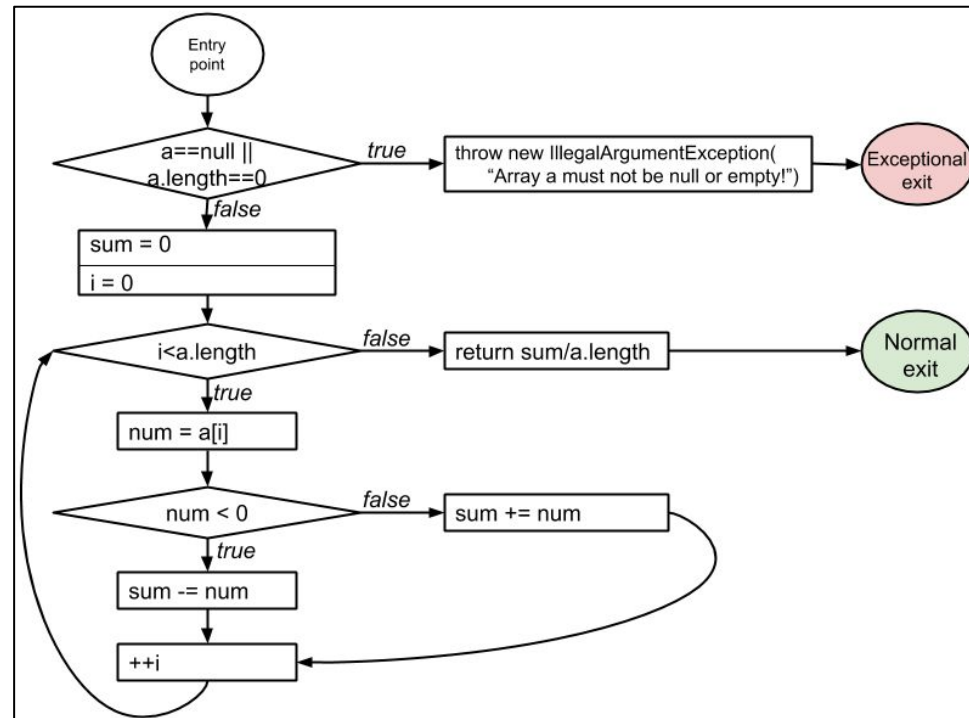
Decision coverage
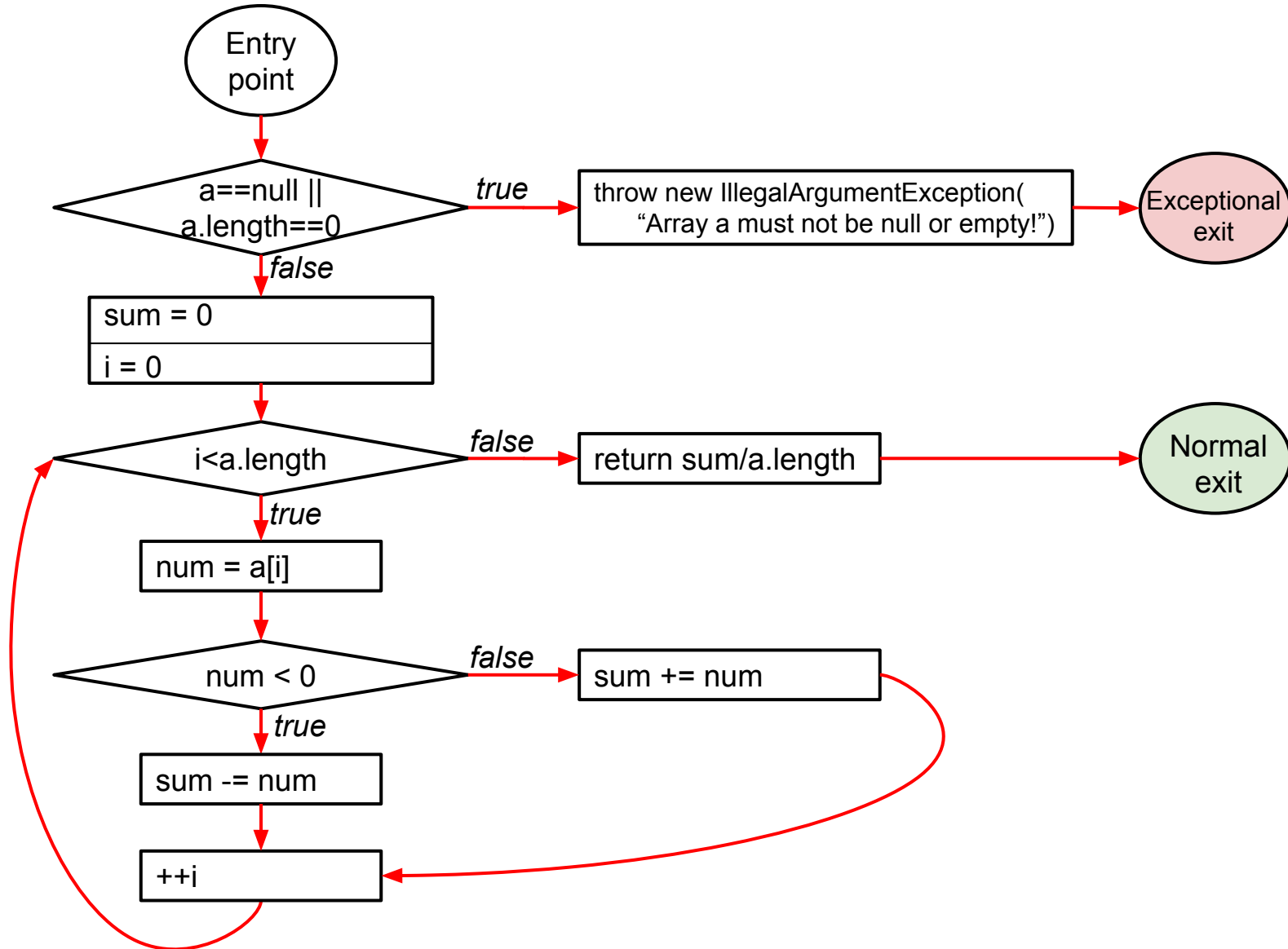
Branch coverage: **Condition** vs. Decision

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.
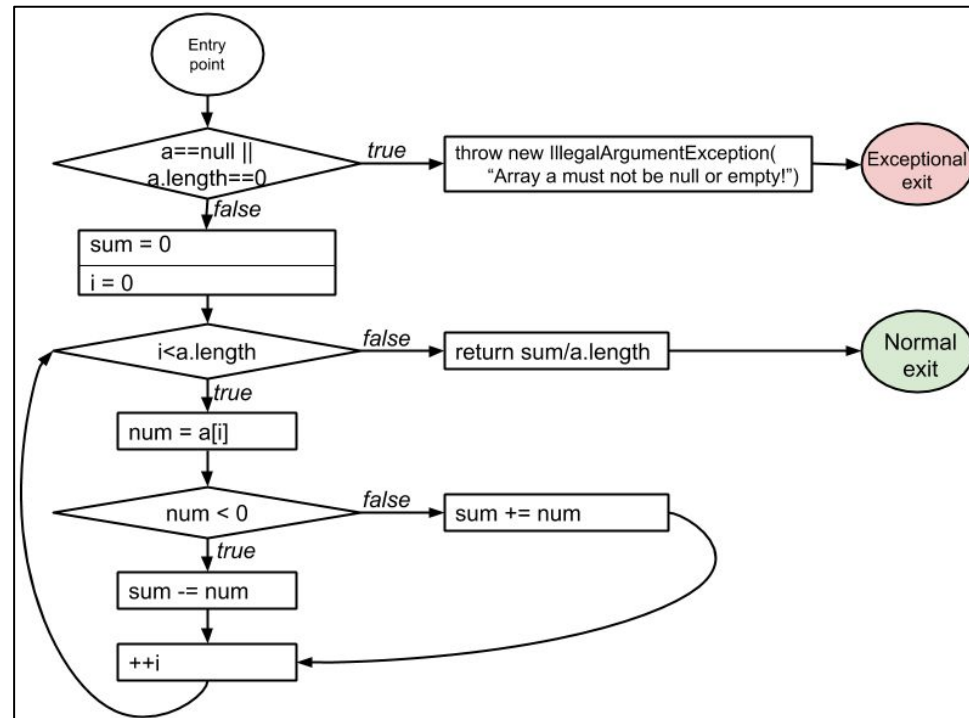
Condition coverage
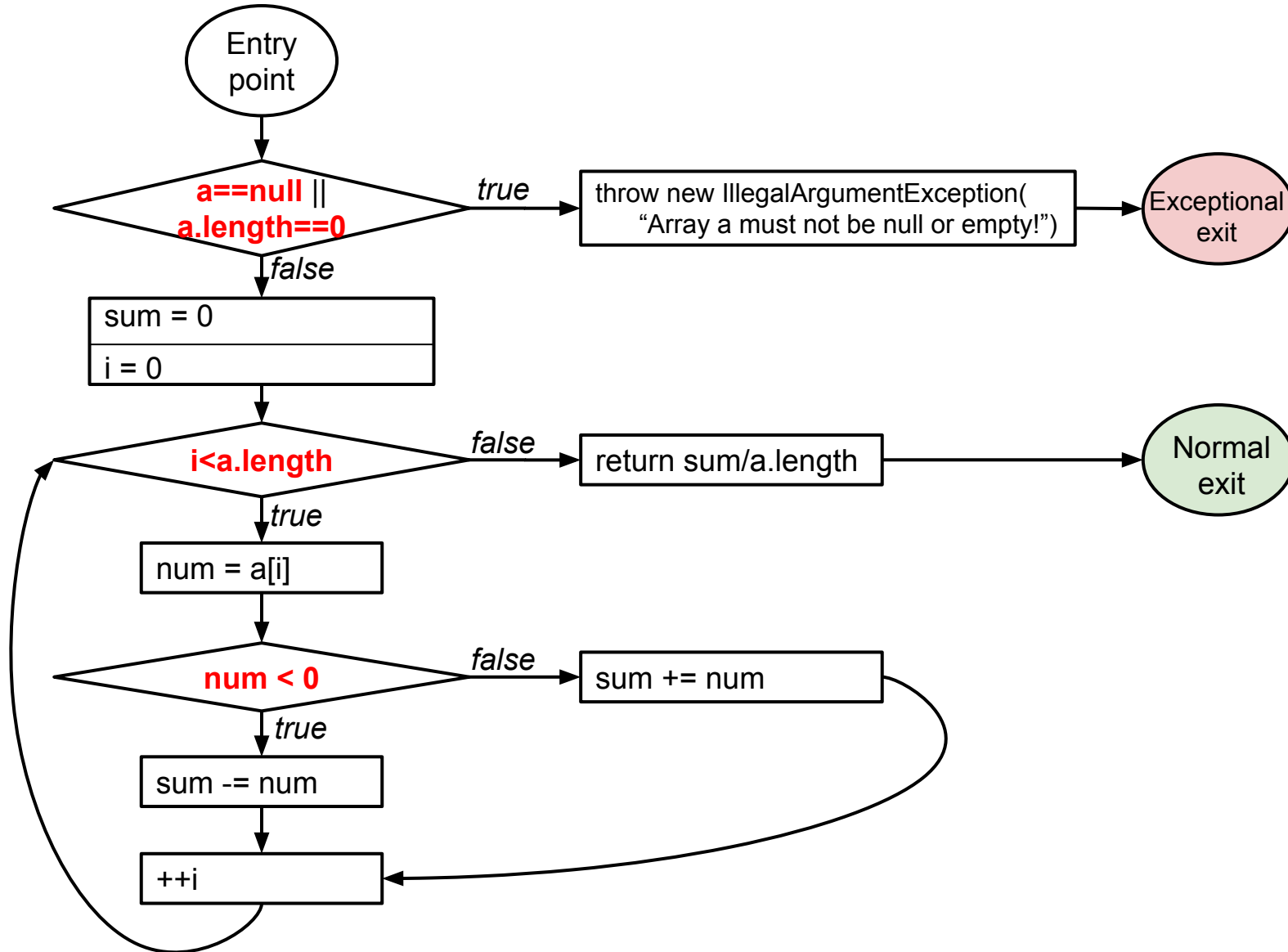
Code coverage metrics

- Statement coverage
- Branch coverage
  - Condition coverage
  - Decision coverage
  - Modified Condition/Decision coverage

Structural code coverage: subsumption

Given two coverage criteria A and B,

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

- Subsumption relationships (True or False):

  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://tinyurl.com/cse403-cov

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

Decision coverage vs. condition coverage

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

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

a	b	$a \mid b$
0	0	0
0	**1**	**1**
1	**0**	**1**
1	1	1

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

a	b	$a \mid b$
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)

Modified Condition/Decision Coverage (MC/DC)

Do not confuse… 🤘

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](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

```
46
47  @app.post("/studies/{study_id}/data")
48  async def post_study_data(study_id: UUID, study_data: dict, study: dict = Depends(stud
49      if study['id'] == str(study_id):
50          data_access = DataAccessFactory()
51          study_data_access: StudyDataMongo = data_access.access_points[data_access.avai
52          result = study_data_access.add_data(str(study_id), study_data)
53          return result
54      else:
55          raise HTTPException(
56              status_code=status.HTTP_401_UNAUTHORIZED,
57              detail="You don't have access to the study: {}.".format(study_id)
58          )
59
```

- 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!