Introduction to Mutation Testing

Can Vural

Can Vural

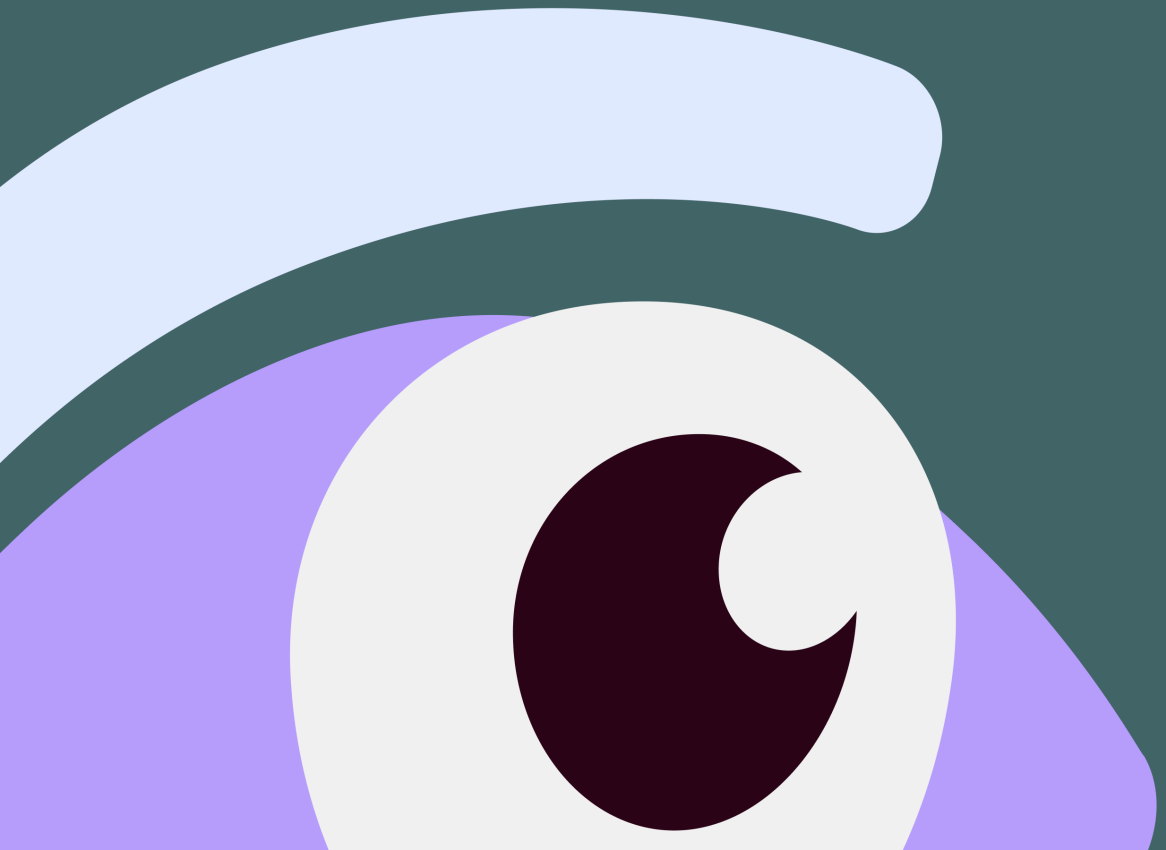
Senior Backend Developer @ StuDocu

GitHub canvural

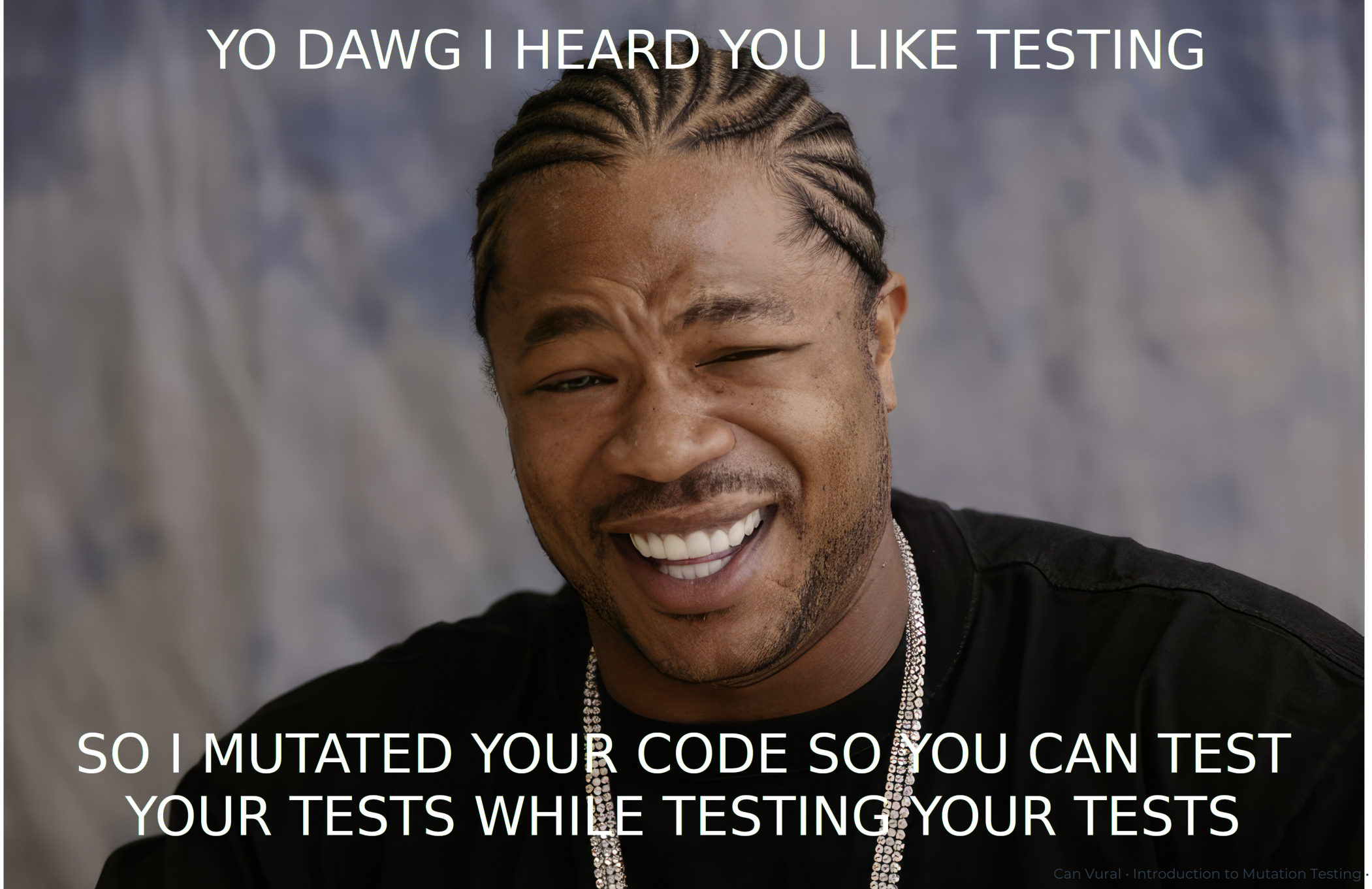
X can__vural

Selfie time!

What will we learn today?



- What is mutation testing?
- How does it work?
- Advantages and disadvantages
- How to get started?


YO DAWG I HEARD YOU LIKE TESTING
SO I MUTATED YOUR CODE SO YOU CAN TEST YOUR TESTS WHILE TESTING YOUR TESTS

What is mutation testing?

What is mutation testing?

Mutation testing is a software testing technique where deliberate **mutants are introduced into a program's source code**. The objective is to evaluate the efficacy of a test suite by observing how many of these mutants it can detect. A high detection rate implies a **robust test suite**, while many undetected mutants suggest **potential testing gaps**, guiding developers on areas to improve for greater software reliability.

Brief history

- Originated in the 1970s
- Mutation testing evolved through contributions from researchers and practitioners
- With the increase of computing power, it gained popularity

Basics of Mutation Testing

- Mutation - single change of code
- Mutant - mutated source code
- Mutation Operator

Original	Mutated
>	>=
===	!==
return true;	return false;

Basics of Mutation Testing

- MSI - Mutation Score Indicator
- Mutation Code Coverage
- Covered Code MSI

Why do we need it and how does it work?

Code Coverage

Types of code coverage

Three predominant types that are most commonly referenced are line coverage, branch coverage, and path coverage.

- Line Coverage
- Branch Coverage
- Path Coverage

```php
function isLeapYear($year): bool {
    return $year % 4 === 0 && (
        $year % 100 !== 0 ||
        $year % 400 === 0
    );
}

isLeapYear(2024); // true
```

Types of code coverage

Three predominant types that are most commonly referenced are line coverage, branch coverage, and path coverage.

- **Line Coverage**
- Branch Coverage
- Path Coverage

```php
function isLeapYear($year): bool {
    return $year % 4 === 0 && (
        $year % 100 !== 0 ||
        $year % 400 === 0
    );
}

isLeapYear(2024); // true
```

Types of code coverage

Three predominant types that are most commonly referenced are line coverage, branch coverage, and path coverage.

- Line Coverage
- **Branch Coverage**
- Path Coverage

```
function isLeapYear($year): bool {
    return $year % 4 === 0 && (
        $year % 100 !== 0 ||
        $year % 400 === 0
    );
}

isLeapYear(2024); // true
```

Types of code coverage

Three predominant types that are most commonly referenced are line coverage, branch coverage, and path coverage.

- Line Coverage
- Branch Coverage
- **Path Coverage**

```php
function isLeapYear($year): bool {
    return $year % 4 === 0 && (
        $year % 100 !== 0 ||
        $year % 400 === 0
    );
}

isLeapYear(2024); // true
```

Code coverage is not a quality target

> Test coverage is a useful tool for finding untested parts of a codebase. Test coverage is of little use as a numeric statement of how good your tests are.
>
> Martin Fowler

How does it work?

Source

```php
public function isLeapYear($year): bool {
      return $year % 4 === 0 && (
          $year % 100 !== 0 ||
          $year % 400 === 0
      );
  }
```

Mutations

```php
public function isLeapYear($year): bool {
      return $year * 4 === 0 && (
          $year % 100 !== 0 ||
          $year % 400 === 0
      );
  }
```

Modulo → Failing test(s) → ✓

```php
public function isLeapYear($year): bool {
      return $year % 4 === 0 && (
          $year % 100 !== 0 ||
          $year % 399 === 0
      );
  }
```

Decrement Integer → All tests pass → 

Escaped Mutant

```php
public function isLeapYear($year): bool {
      return $year % 4 === 0 || (
          $year % 100 !== 0 ||
          $year % 400 === 0
      );
  }
```

Logical And → Failing test(s) → ✓

Test suite

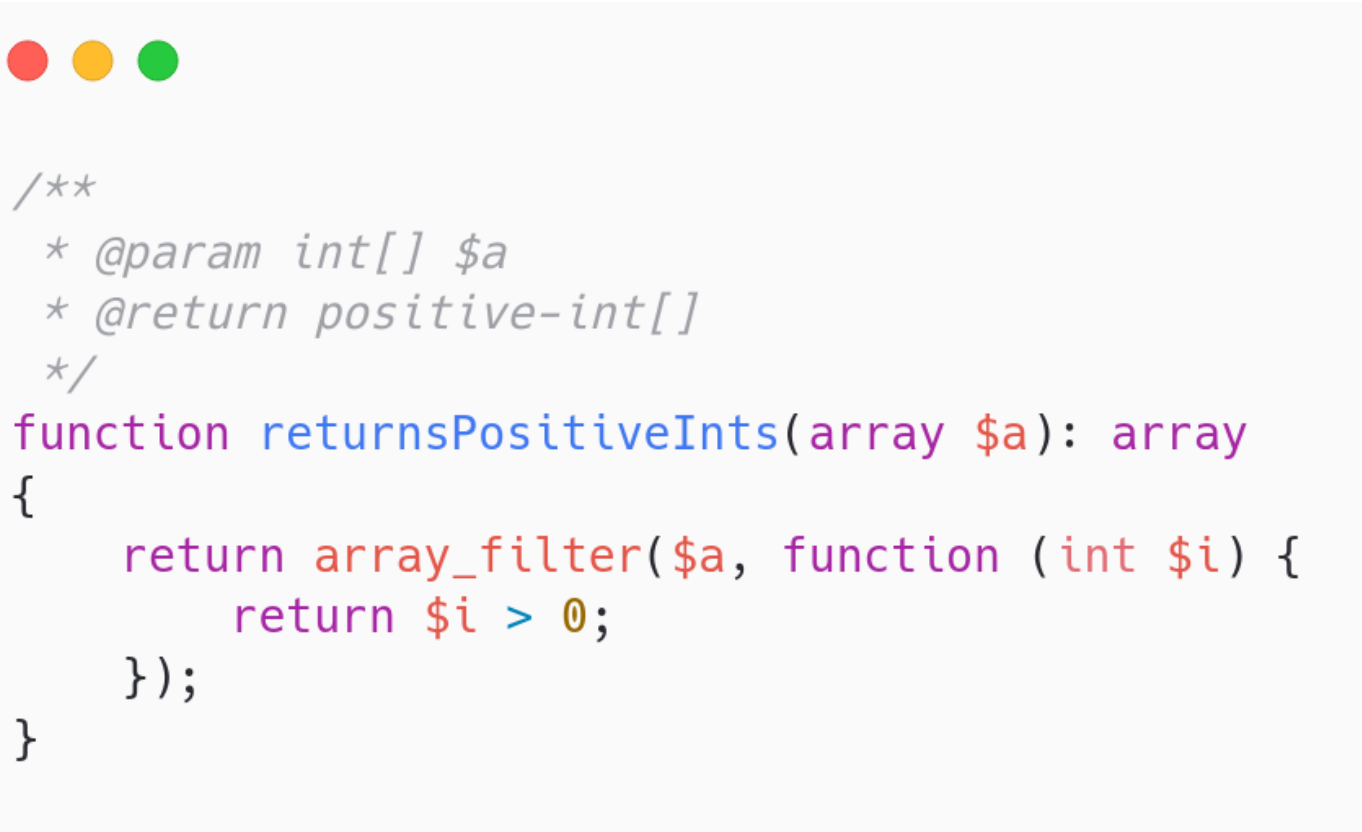
Problems

- **Speed**
- False positives
- Equivalent mutants

Speed = (N * T)

- **N** - Number of mutants
- **T** - Time it takes to run the test

- Speed
- **False positives**
- Equivalent mutants



```php
/**
 * @param int[] $a
 * @return positive-int[]
 */
function returnsPositiveInts(array $a): array
{
    return array_filter($a, function (int $i) {
        return $i > 0;
    });
}
```

- Speed
- **False positives**
- Equivalent mutants

```php
/**
 * @param int[] $a
 * @return positive-int[]
 */
function returnsPositiveInts(array $a): array
{
    return $a;
}
```

- Speed
- False positives
- **Equivalent mutants**

Equivalent mutants are the mutants that are identical logically to the original source code.

```
function foo(int $c)
{
    $b = -1;
    $a = $c * $b;
}
```

- Speed
- False positives
- **Equivalent mutants**

```
function foo(int $c)
{
    $b = -1;
    $a = $c / $b;
}
```

How to get started with Infection?

- **On PRs**
- On whole codebase
- On a new project

- Make sure your new feature is covered by tests.
- Make sure to use `@covers` or `#[CoversClass(Foo::class)]` annotations.
- Run Infection only for changed or newly added files in PR with `infection --git-diff-filter=AM --min-msi=100`

- On PRs
- **On whole codebase**
- On a new project

- If your codebase is large, consider running Infection daily or even weekly.
- Set a threshold for mutation score and fail the build if it is not met.
- Make use of the generated report.

- On PRs
- On whole codebase
- **On a new project**

- Start strict and set `--min-msi` to 100.
- ???
- Profit!

Demo Time!

What did we learn?

- **Code coverage alone is not a good metric**
- Mutation testing
- More reliable tests

100% code coverage does not say anything about the **quality of your tests**. It is easy to write simple tests that cover all lines of code, but it is hard to write tests that cover all possible scenarios. Mutation testing can help you to write **more reliable tests**.

What did we learn?

- Code coverage alone is not a good metric
- **Mutation testing**
- More reliable tests

- Mutation - single change of code
- Mutant - mutated source code
- Mutation Operator

What did we learn?

- Code coverage alone is not a good metric
- Mutation testing
- **More reliable tests**

If our tests successfully detect mutations, it means they're doing a great job at finding potential bugs

Thank you