Mutation Testing

Expose Your Tests' Hidden Weaknesses

mesutdurukal

MESUT DURUKAL

QA CAPABILITIES LEAD

Building and scaling the organization’s cross-team quality skills, advocating continuous delivery with built-in quality and improving efficiency of QA teams.

Experience

- Defense (7 years)
- Siemens (5 years)
- Robotics (1 year)
- Indeed (4 years)
- OMP (3 months)

Background

- Project Management, PMP
- Agile Practitioner, PSM & PSO
- Quality Assurance, ISTQB TM
- Test Automation, CI/CD
- Software development

Personal

- Previously lived in Turkey and Japan
- Currently living in Antwerp with wife and 1.5yo daughter
- Hobbies: Football, travel, food exploration
- Public speaking: approximately 200 appearances

Test Smells

https://test-smell-catalog.readthedocs.io/en/latest/

Test Smell Groups	Test Smell Types
Stability & Reliability Related	Flakiness
	Suite Dependency
	Fragility
Distortive	Assertions
	Mocks
Scope Related	Eager Tests
	Limited Scope
	Test Scope Overlap
Performance Related	Slow or Long Running
Structural	Code Duplication
	Long Tests
	Obscure Tests
	Bad Naming
	Exception Handling

- **Google** (Google Testing Blog)
  - Of ~4.2 million tests running in their CI system,
  - ~ 16 % of the tests have some level of flakiness
  - ~ 63K (≈1.5 %) had a flaky run during one week.
  - ~ 84 % of pass-to-fail transitions were caused by flaky tests.

- **Microsoft** (Microsoft for Developers)
  - “Flaky Test Management System” revealed that
  - across more than 100 teams.
  - over ~49,000 flaky tests had been identified
  - Their system also passed ~160,000 sessions that would otherwise have failed due to flaky tests.
  - Asynchronous calls were the leading cause of flakiness.

Test Reliability

Quality Gate :P

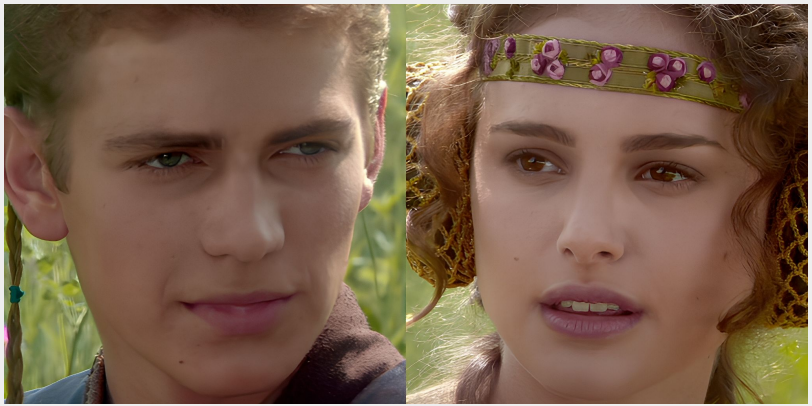
		Any Issues in the feature?	
		No	Yes
Test Report	PASS	No Problem	Silent Horror
	FAIL	False Alarm	Real Bug

Not covered

Not well covered

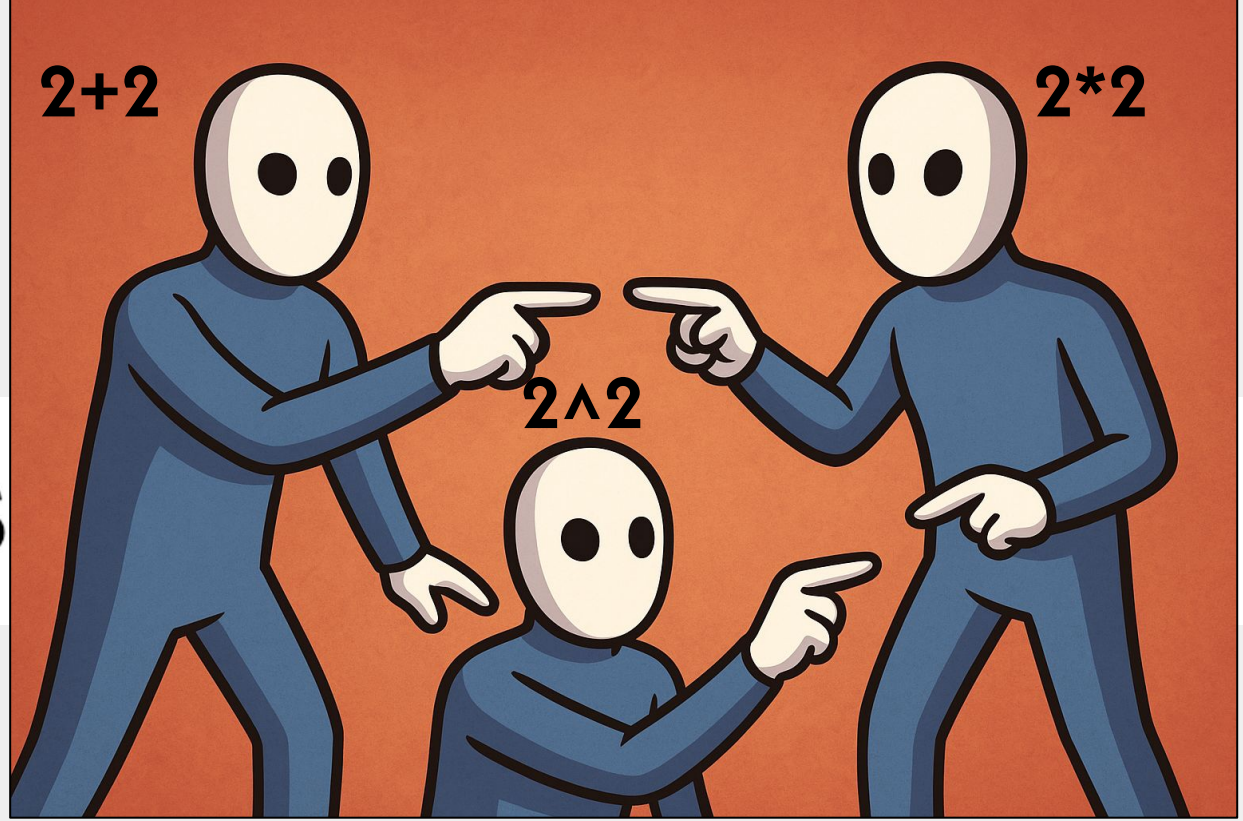
Quantity != Quality

I have 1000 tests running



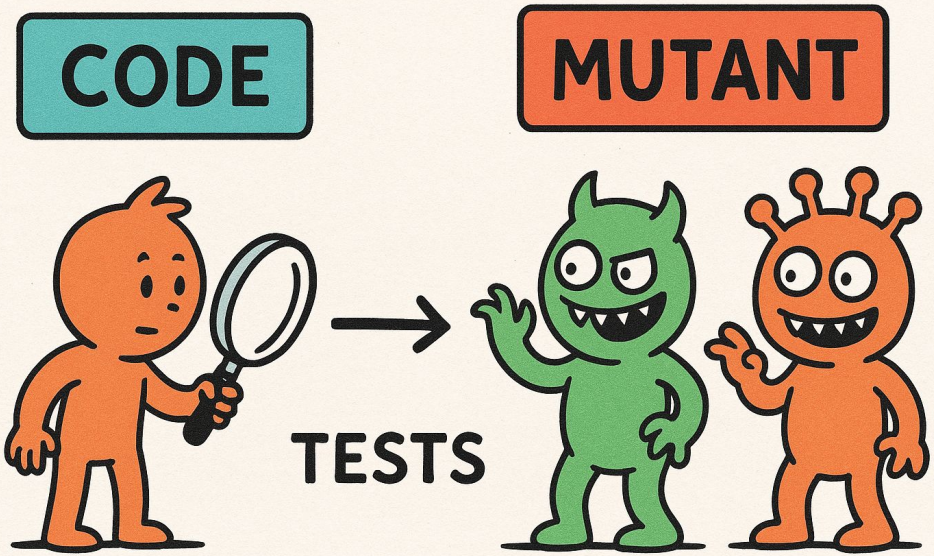
I forgot to add assertions

as



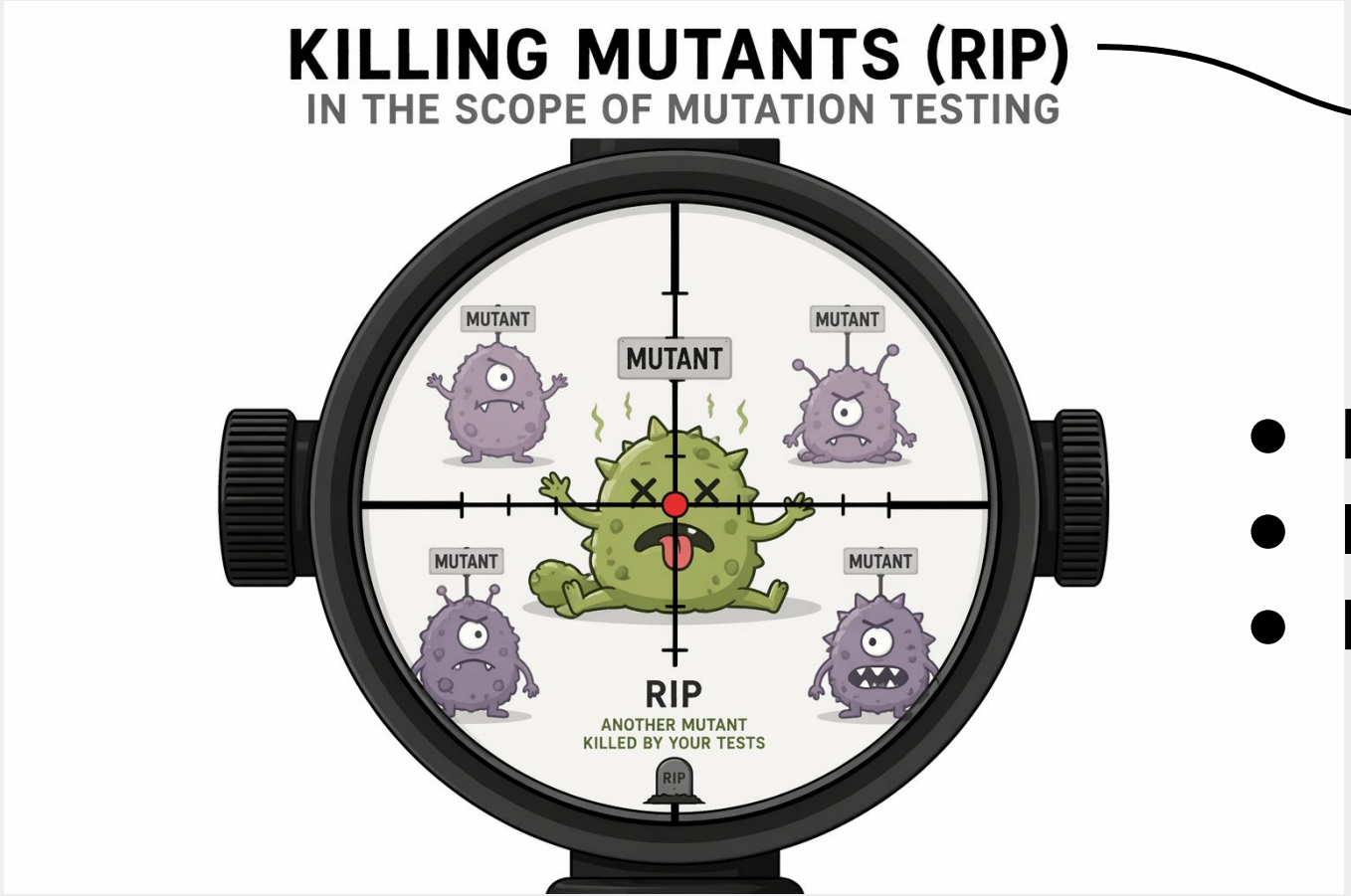
4

assert power (2,2) == 4



- Making mutations in source code
- White box testing
- introduced by **Richard Lipton** in 1971

After injected mutation, if test fails (blocking release) > Killed mutant



RIP

- Reached
- Infected
- Propagated

“

Mutation testing doesn’t check if your **code** works — it **checks** if your **tests** really know when it doesn’t.

Mutation Types

- Value mutation

Original Code (Java):

```java
int originalValue = 10;
      if (originalValue > 5) {
      System. out.println("Original code: Value is greater than 5.");
      }
```

Mutant Code (Value Mutated):

```java
int originalValue = 10;
     int mutantValue = 2; // Changed from 10 to 2
     if (mutantValue > 5) {
     System.out.println("Mutant code: Value is greater than 5.");
     }
```

Mutation Types

- Decision mutation

Original Code (Python):

```
a = 10
b = 5
if a > b:
   print("Original code: a is greater than b.")
```

Mutant Code (Decision Mutated):

```
a = 10
b = 5
if a < b:  # Changed from a > b
print("Mutant code: a is less than b.")  # Changed message
```

Mutation Types

- Statement mutation

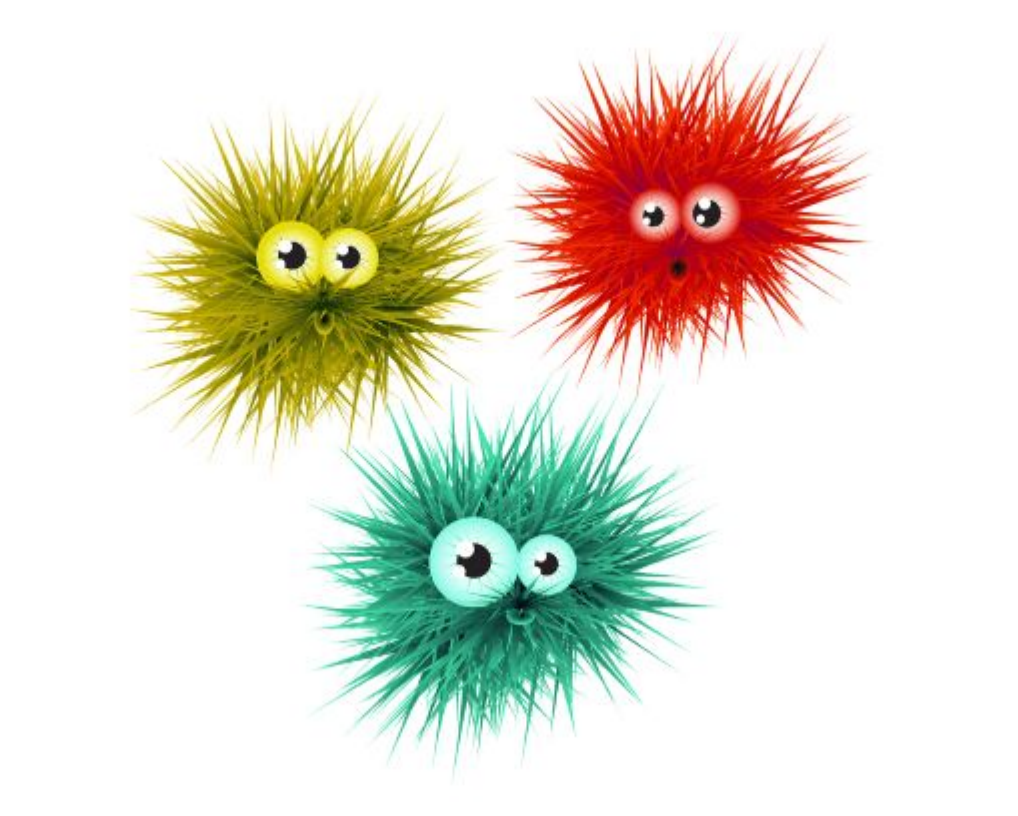
Original Code (C++):

```
int x = 5;
int y = 10;
int result = x + y;
```

Mutant Code (Statement Mutated):

```
int x = 5;
int y = 10; // Mutated: Statement removed
int result = x - y; // Changed operation from addition to subtraction
```

Demo



Value mutation

Mutator	What it does	Example
`EMPTY_RETURNS`	Replaces return value with default (0, false, null)	`return a + b;` → `return 0;`
`FALSE_RETURNS`	Forces boolean return to `false`	`return x > 0;` → `return false;`
`TRUE_RETURNS`	Forces boolean return to `true`	`return x > 0;` → `return true;`
`NULL_RETURNS`	Replaces object return with `null`	`return obj;` → `return null;`
`PRIMITIVE_RETURNS`	Replaces primitive return with default (0, 0L, etc.)	`return a * b;` → `return 0;`

Decision mutation

Mutator	What it does	Example
`NEGATE_CONDITIONALS`	Negates conditions	`if (b == 0) → if (b != 0)`
`INVERT_NEGS`	Removes/adds negation (`!`)	`!flag → flag`
`CONDITIONALS_BOUNDARY`	Changes boundary operators	`x > 0 → x >= 0`

Statement mutation

Mutator	What it does	Example
`MATH`	Replaces arithmetic operators	`a + b → a - b`, `a * b → a / b`
`INCREMENTS`	Changes increment/decrement	`i++ → i--`
`VOID_METHOD_CALLS`	Removes void method calls	`doSomething(); → (removed)`

mesutdurukal

Thank you!