All Tests green? Oh no!!!

Why it is sometimes good, when a test fails.

OOP München 2024, 30.01.2024, Birgit Kratz

About me

Birgit Kratz

- Freelancing IT Consultant
- Java-Backend
- More than 25 years experience
- Co-Organizer of Softwerkskammer in Düsseldorf and Köln (Cologne)
- Co-Organizer of SoCraTes-Conf Germany
- Email: mail@birgitkratz.de
- Mastodon: @birgitkratz@jvm.social
- Github: https://github.com/bkratz
- Web: https://www.birgitkratz.de

Agenda

What is Mutation Testing and how does it work

What kind of problems can be solved with it

Disadvantages

Tipps

First some questions

Even with 100% code coverage…

… can you tell how good and reliable your tests are?

Goodhart’s Law

When a measure becomes a target, it ceases to be a good measure.

How to assess the quality of a test suite?

Possible Answers

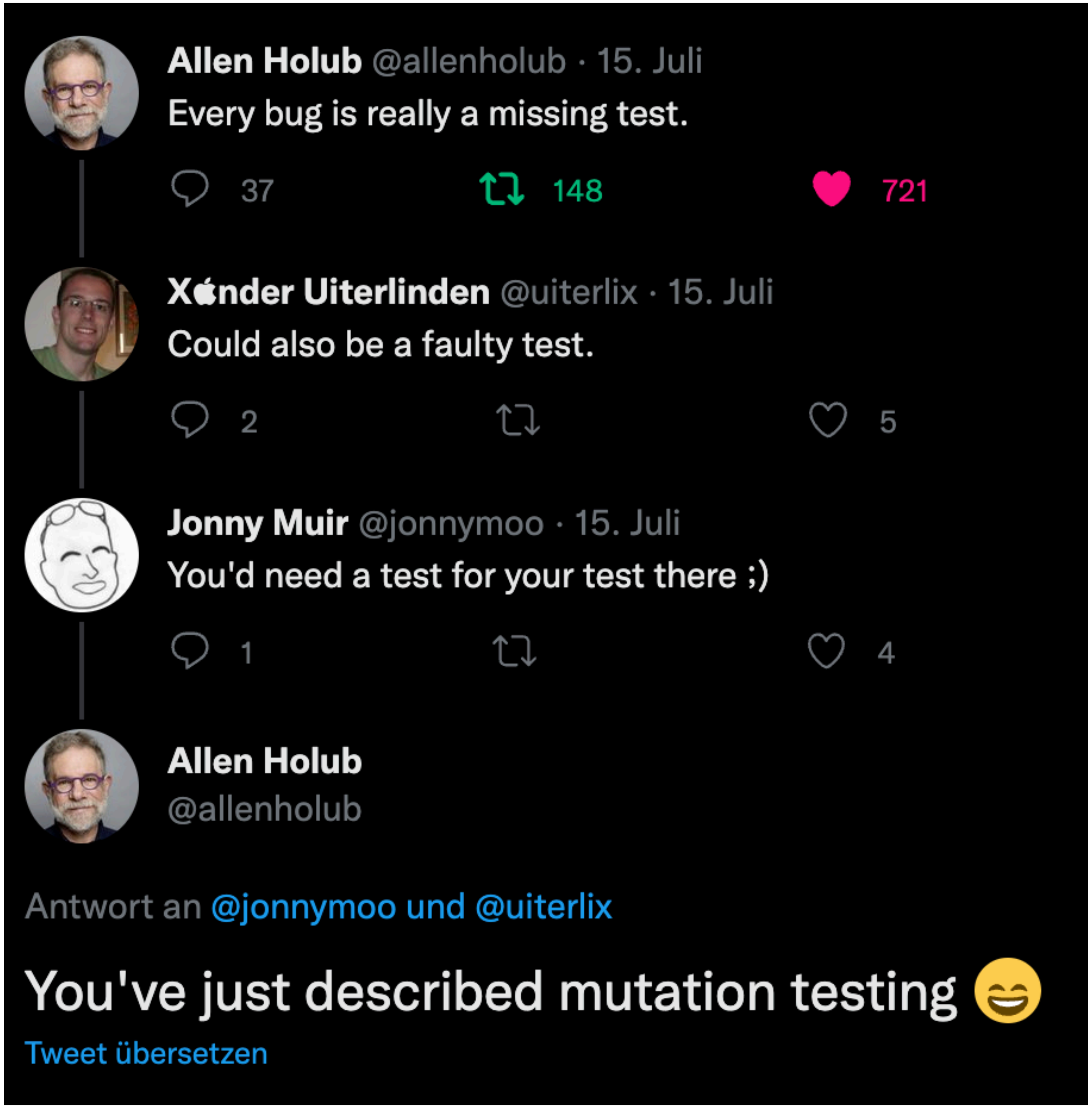
✓we do TDD

✓we do code reviews

✓we have a Quality department

> “Program testing can be used to show the presence of bugs, but never to show their absence!”

— Edsger W. Dijkstra in “Notes On Structured Programming”



Allen Holub @allenholub · 15. Juli

Every bug is really a missing test.

X🍎nder Uiterlinden @uiterlix · 15. Juli

Could also be a faulty test.

Jonny Muir @jonnymoo · 15. Juli

You'd need a test for your test there ;)

Allen Holub
@allenholub

Antwort an @jonnymoo und @uiterlix

You've just described mutation testing 😄

1971: Richard Lipton

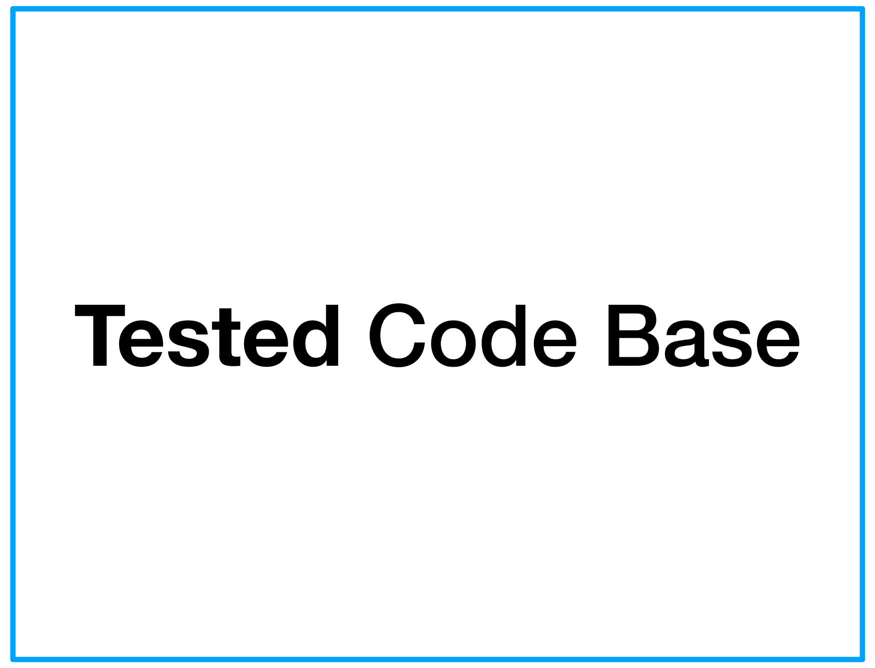
Paper: “Fault diagnosis of computer programs”

If you want to know, whether your test suite properly checks your code, introduce a bug and then see if the test suite can find the bug.

Mutation Testing

How it works

How it works

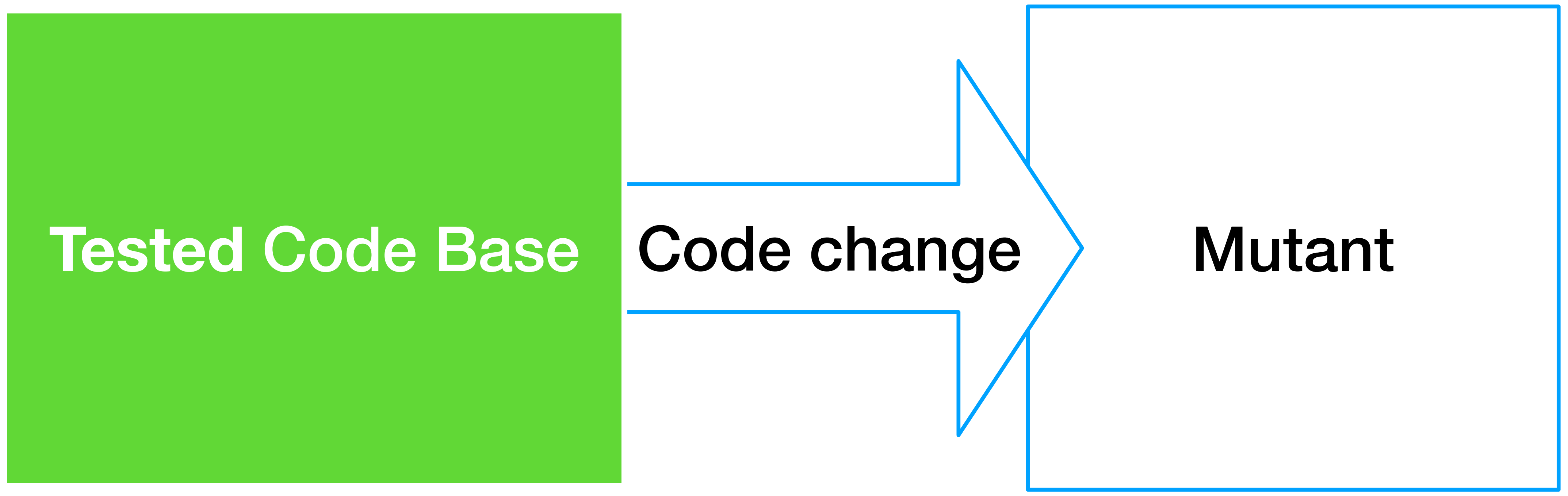


Tested Code Base

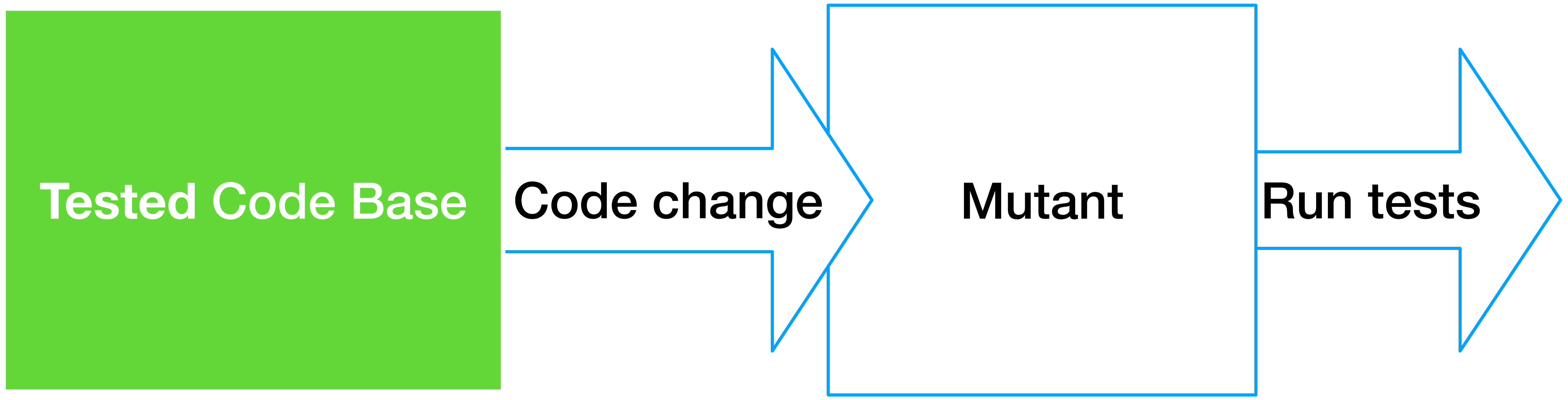
How it works

Tested Code Base

How it works



How it works



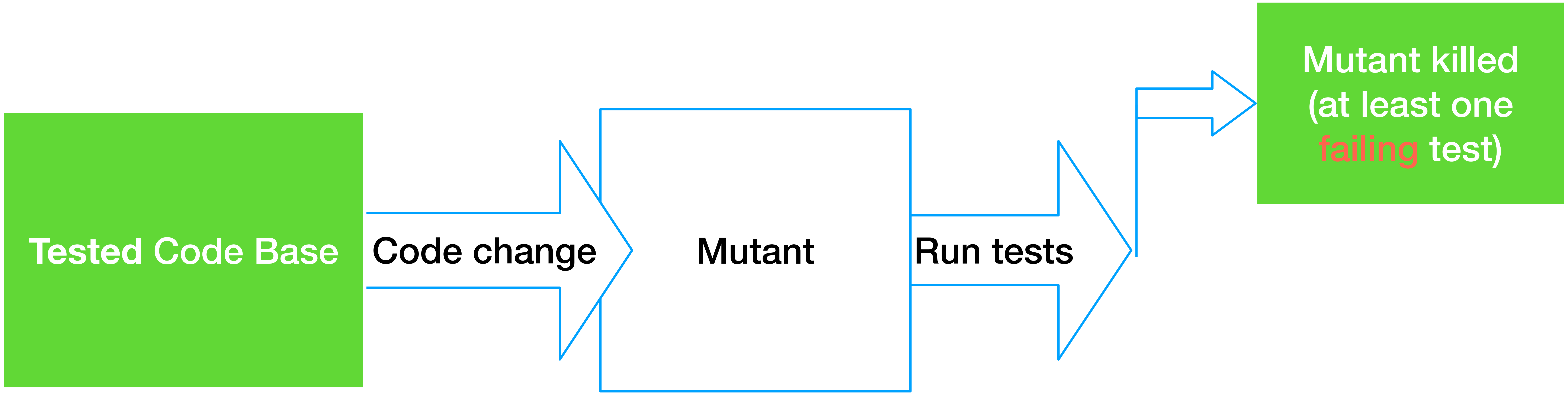
Tested Code Base

Code change

Mutant

Run tests

How it works



Tested Code Base

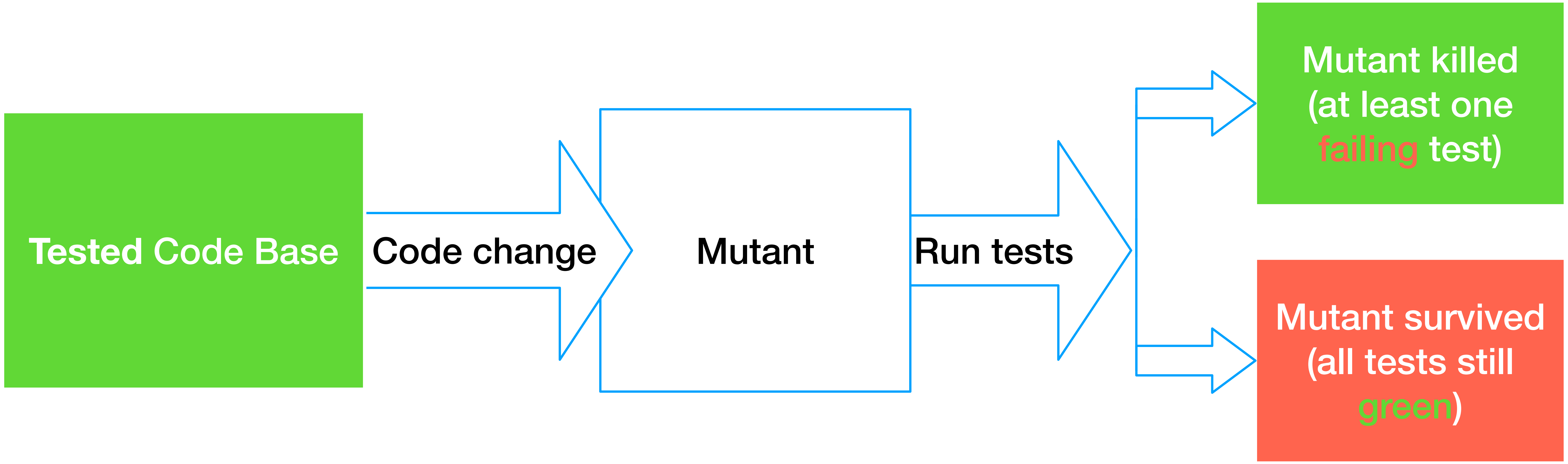
Code change

Mutant

Run tests

Mutant killed (at least one failing test)

How it works



Tested Code Base

Code change

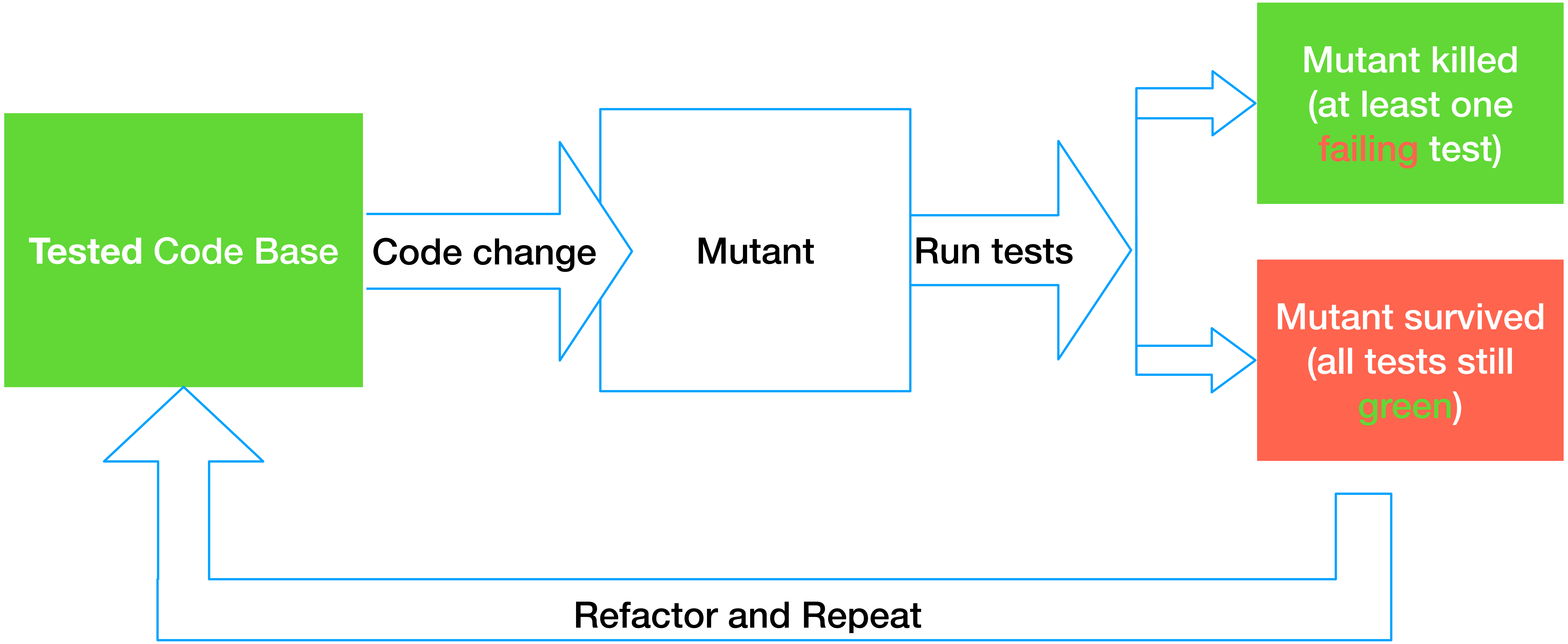
Mutant

Run tests

Mutant killed (at least one failing test)

Mutant survived (all tests still green)

How it works


Tested Code Base
Code change
Mutant
Run tests
Mutant killed (at least one failing test)
Mutant survived (all tests still green)
Refactor and Repeat

Which kind of Mutants are we talking about?

Conditional Boundary Mutator

Original	Mutant
<	<=
<=	<
>	>=
>=	>

Negate Conditionals Mutator

Original	Mutant
==	!=
!=	==
>	<=
>=	<
<=	>
<	>=

Increment Mutator

Original	Mutant
i++	i—
i—	i++

Invert Negatives Mutator

inverts negation of integer and floating point numbers

Original	Mutant
return -i	return i

Math Mutator

Original	Mutant
+	-
*	/
&	|
>>	<<
…	…

Many More

Void Method Call Mutator - removes calls to void methods

Empty Returns Mutator - replaces return values with an ‘empty’ value

False Returns Mutator - always returns false for a primitive boolean return value

True Returns Mutator - always returns true for a primitive boolean return value

Null Returns Mutator - replaces return values with null

Primitive Returns Mutator - replaces int, short, long, char, float and double return values with 0

Constructor Call Mutator - replaces constructor calls with null values

still more…

What kind of problems can be detected / can it help you with?

Detect poorly chosen or missing test data

Detect Ambiguities in code base or Logical errors

Detect missing test cases

Highlighting redundant code and code smells

Finding buggy test cases

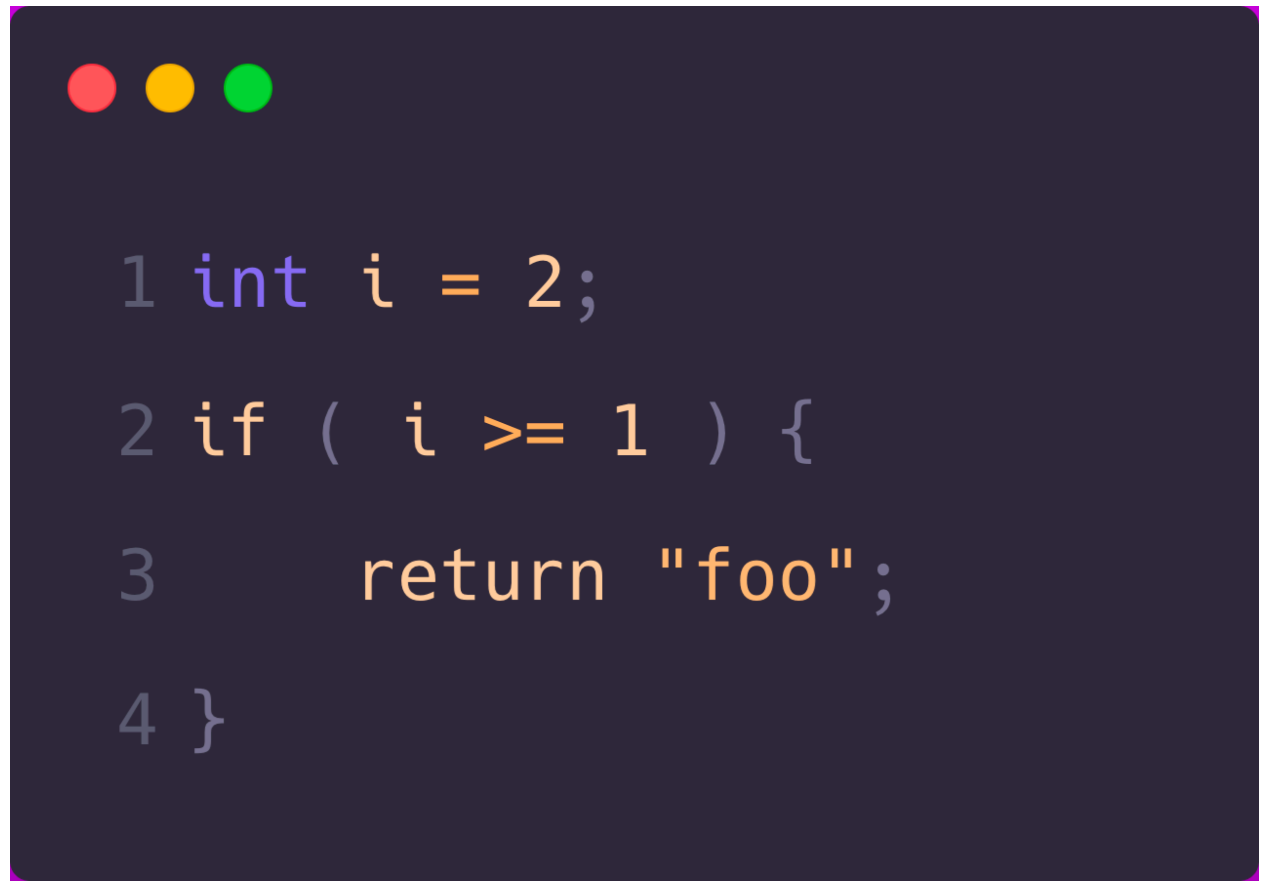
Provide a safety net when refactoring your tests

What kind of problems can not be solved?

Equivalent Mutation

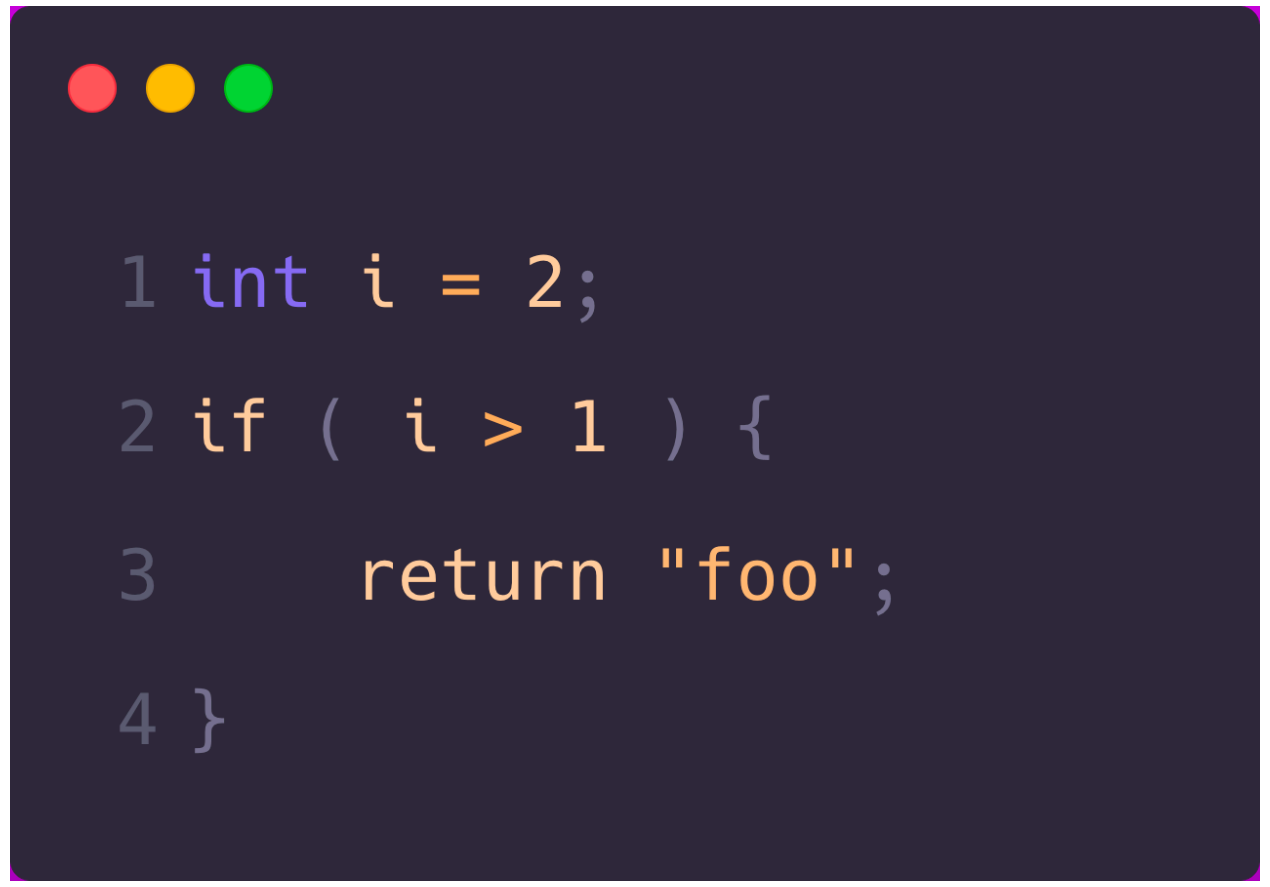
The mutants in this set cannot be killed because they are equivalent to the original program. No possible test input exists that can distinguish their behaviour from that of the original program.

Original



```
int i = 2;
if ( i >= 1 ) {
    return "foo";
}
```

Mutant



```
int i = 2;
if ( i > 1 ) {
    return "foo";
}
```

DEMO
with Java and PIT

(https://pitest.org/)

Disadvantages of Mutation testing

- Can be **very** time consuming
- Cannot detect/avoid equivalent mutations, since the resulting mutant behaves in exactly the same way as the original
- Not suitable for BlackBox Testing, i.e when focusing on frontend tests or E2E tests.

Cost of Mutation Testing

Let’s assume we have:

- a code base with 300 Java classes
- 10 test cases for each class
- on average, each test case requires 0.2 seconds for its execution
- the total test suite execution costs 300 * 10 * 0,2 = **600 seconds** (10 minutes)

Let’s assume we have, on average, 20 mutants per each class.

The total cost of mutation analysis is 300 * 10 * 0,2 * 20 = **12000 seconds** (3h 20 min)

How to reduce this cost?

Run tests in parallel for speed

Do not produce Mutants for code that is not covered by tests

Reduce number of used Mutations

Reduce number of Classes to apply Mutation Testing to

Incremental Analysis

Try it!

- ✓ Try it again
- ✓ Start small
- ✓ Use it as a TOOL to give you feedback as you work
- ✓ Write more tests
- ✓ Get familiar with reported issues and how to solve them
- ✓ Configure it to your needs
- ✓ Start with critical components
- ✓ Don’t use all Mutators all the time

Mutation Test Tools

https://github.com/theofidry/awesome-mutation-testing

Youtube Video with Henry Coles

https://www.youtube.com/watch?v=LoFJajoJQ2g

Questions?

Thank you

Slides: https://www.birgitkratz.de/uploads/OOP_2024_MutationTesting.pdf

Sample code: https://github.com/bkratz/robobar

- Email: mail@birgitkratz.de
- Twitter: @bikratz
- Mastodon: @birgitkratz@jvm.social
- Github: https://github.com/bkratz
- Web: https://www.birgitkratz.de