

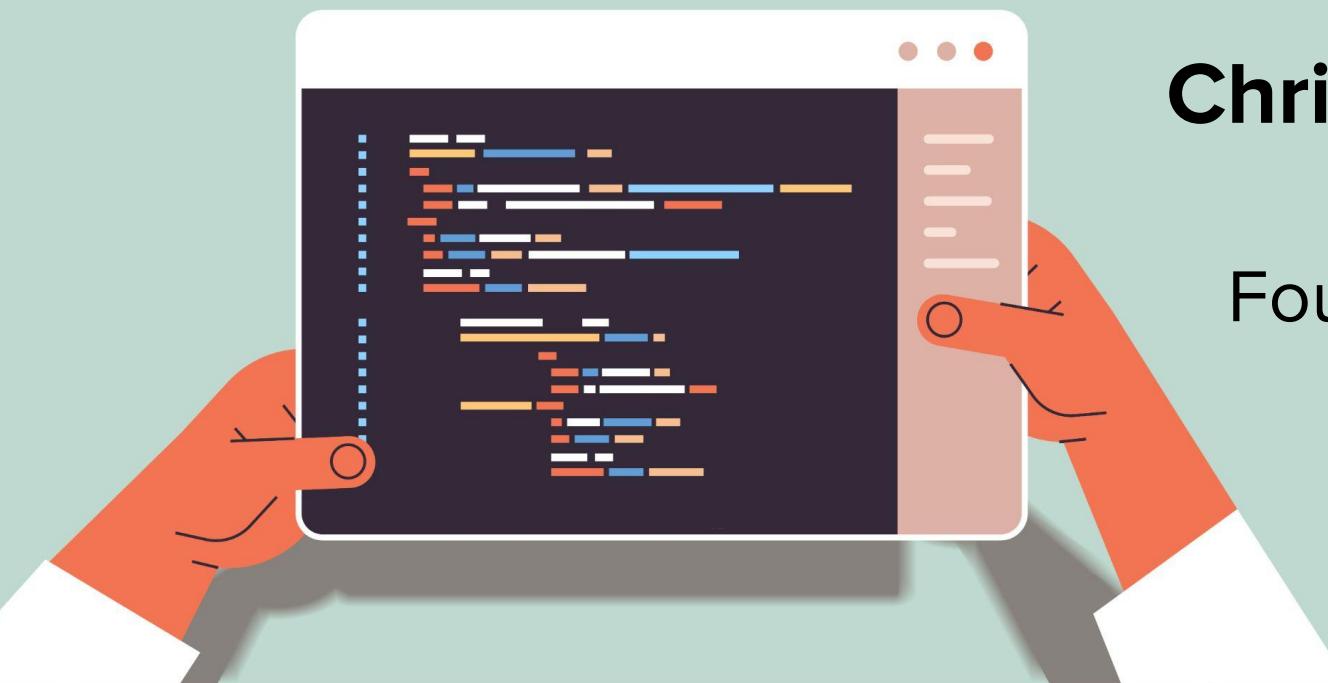
goto;

GOTO

Copenhagen 2022

#GOTOcph

# Five Lines of Code



**Christian Clausen**

Founder, mist-cloud

@theDrLambda

# Technical Agile Coach



A faded, sepia-toned photograph of a group of approximately ten people gathered around a large conference table in what appears to be a meeting room. They are all dressed in dark, professional attire. The scene is somewhat dimly lit, creating a focused and serious atmosphere.

# Technical Agile Coach

# Technical Agile Coach

Test Automation

Clean Code

Continuous attention to **technical excellence**

and **good design** enhances agility.

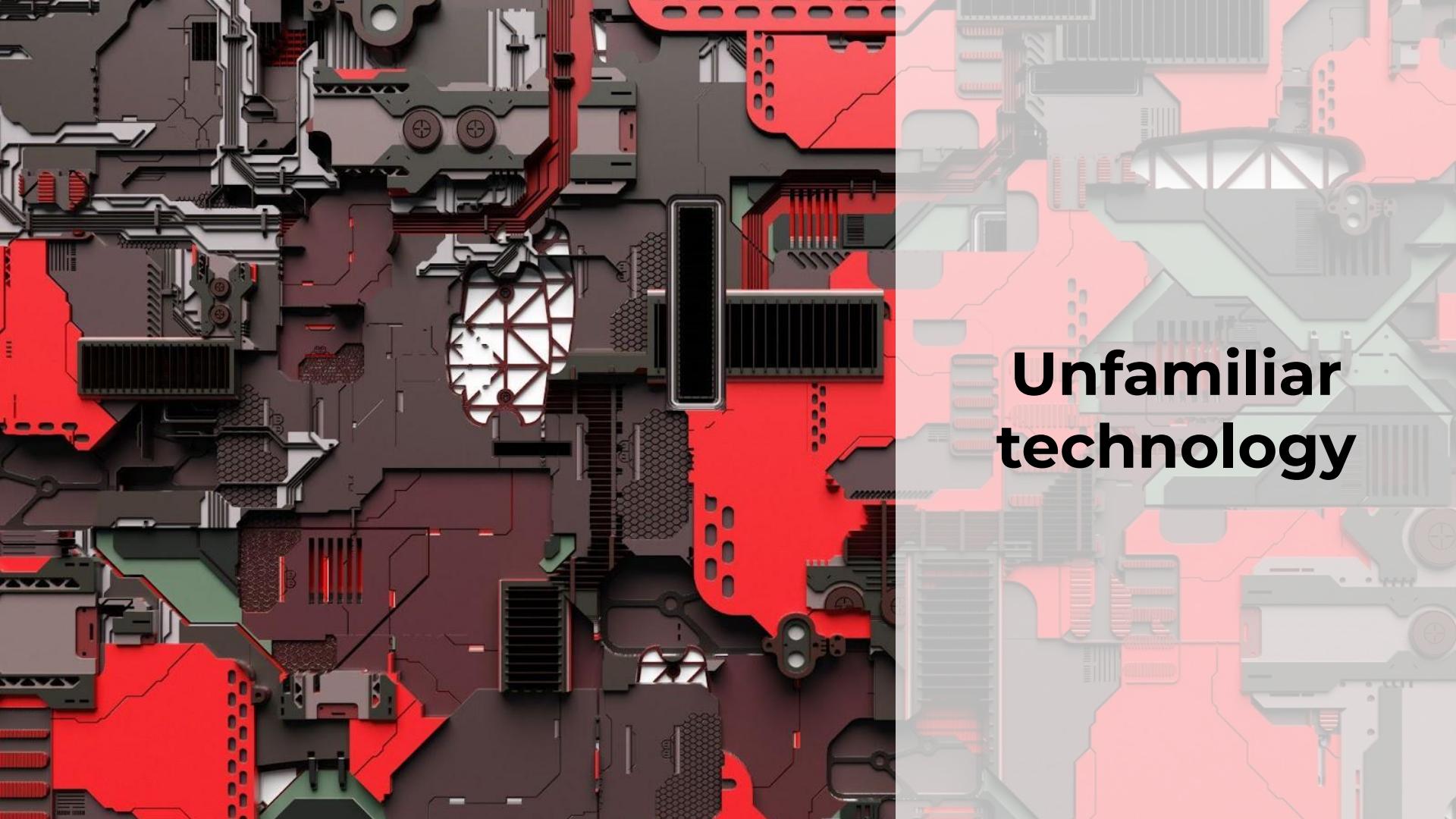
Pair Programming

Branching Strategy

Software Architecture

# A New Project





**Unfamiliar  
technology**



# Huge codebase

**Constantly  
breaking**



# Busfactor



**Lots of red  
tests**





**Architecture was a mess**

**Scared to  
change the  
code**





**I had no  
power**

A photograph of a man with long brown hair and a full brown beard, wearing a light pink button-down shirt. He is standing against a solid red background. His right arm is raised, holding a large, clear incandescent lightbulb by its base. He is looking upwards and slightly to his left with a thoughtful expression, his left hand resting near his chin. The lighting is bright and even.

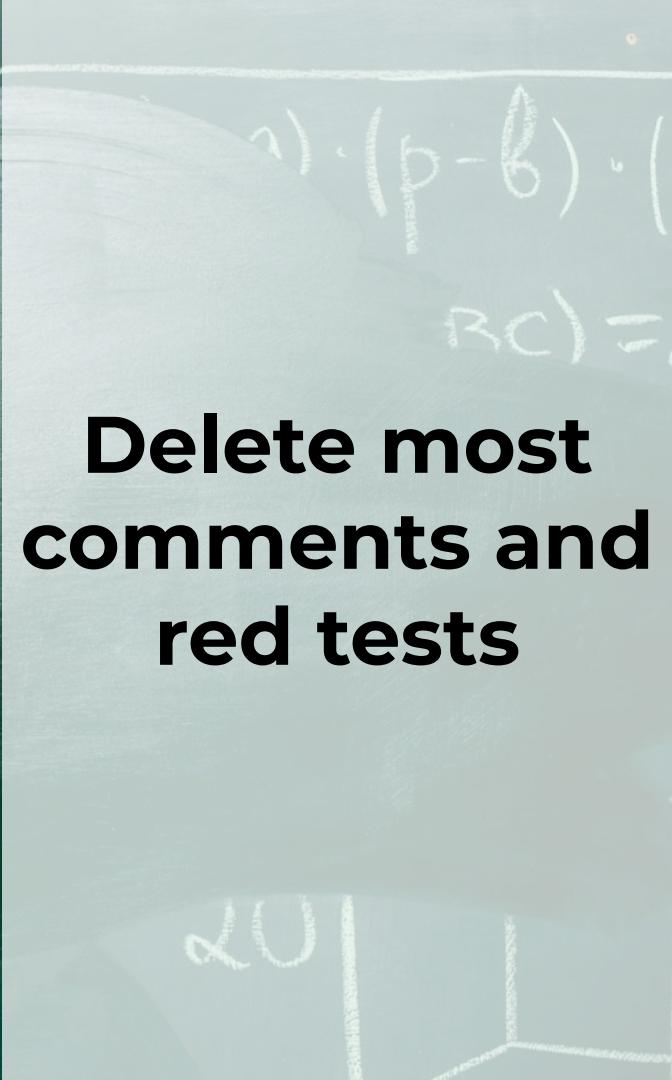
I had a vision



**The team agreed to humor me**

0

**Dead Weight**



**Delete most  
comments and  
red tests**



**Disruptions must go through me**



**Remove the busfactor**

“From head into code  
An idea must go through  
Someone else's hands”

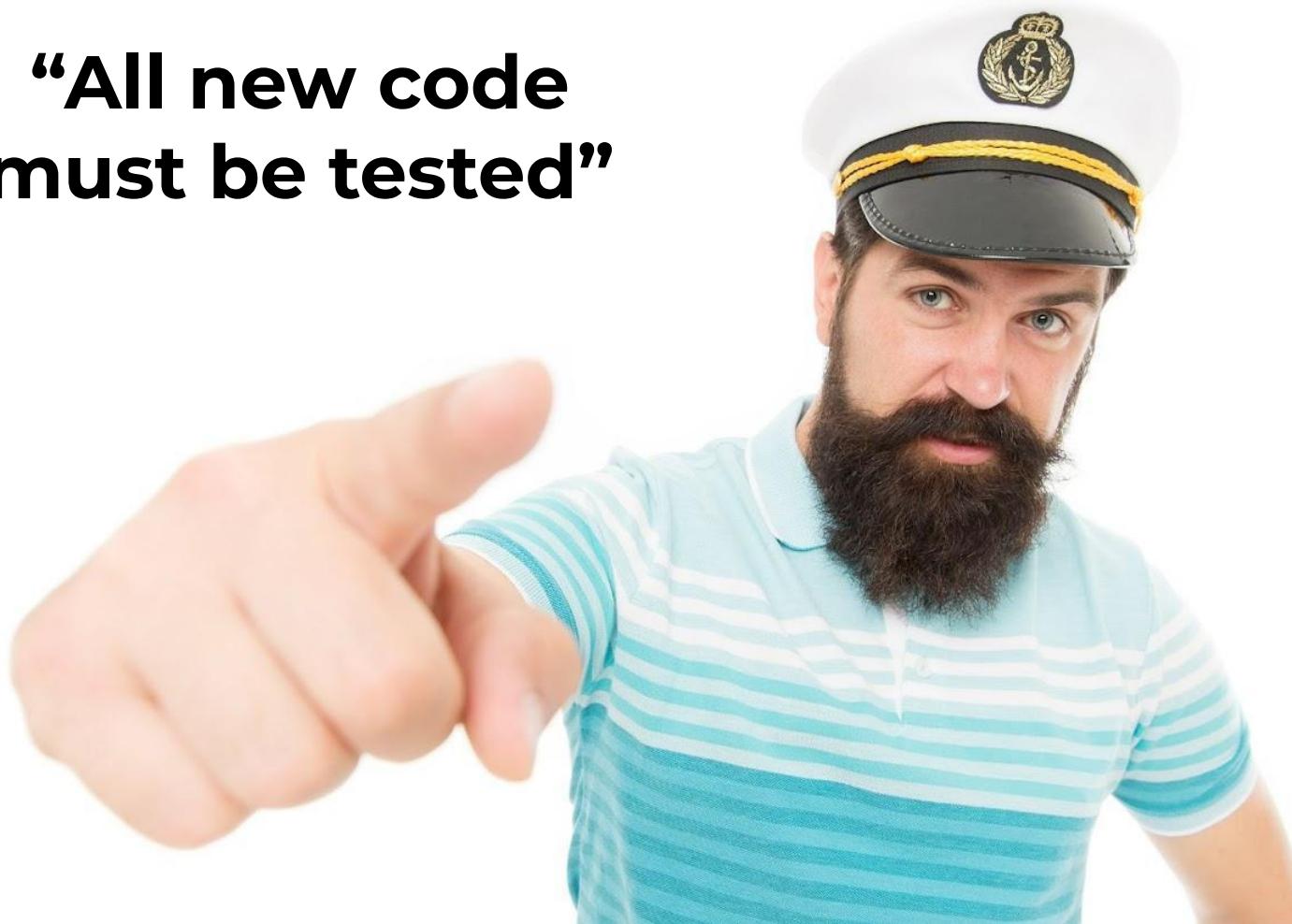
– *Llewellyn's strong-style pairing*



1

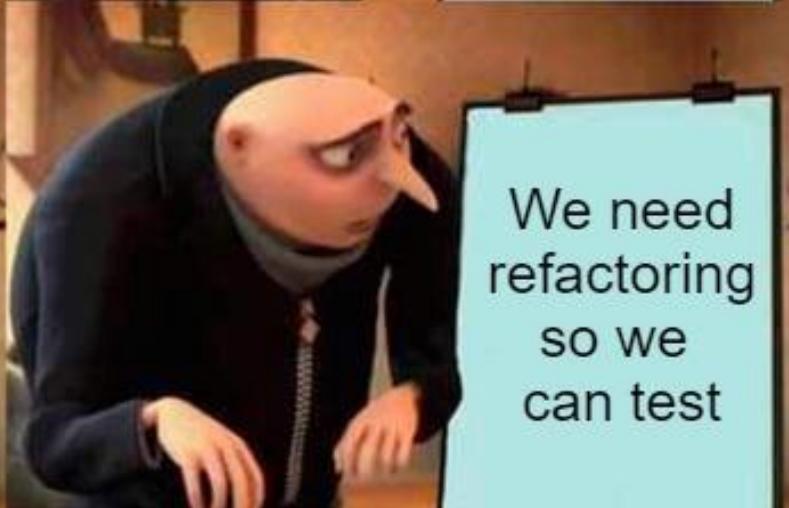
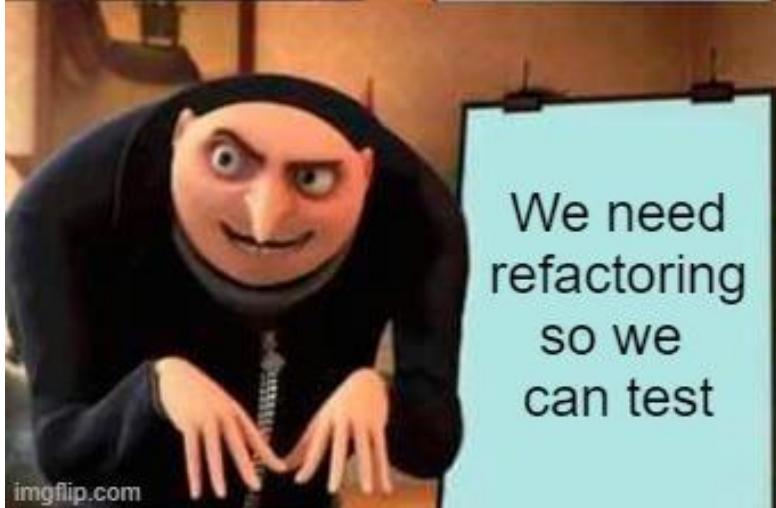
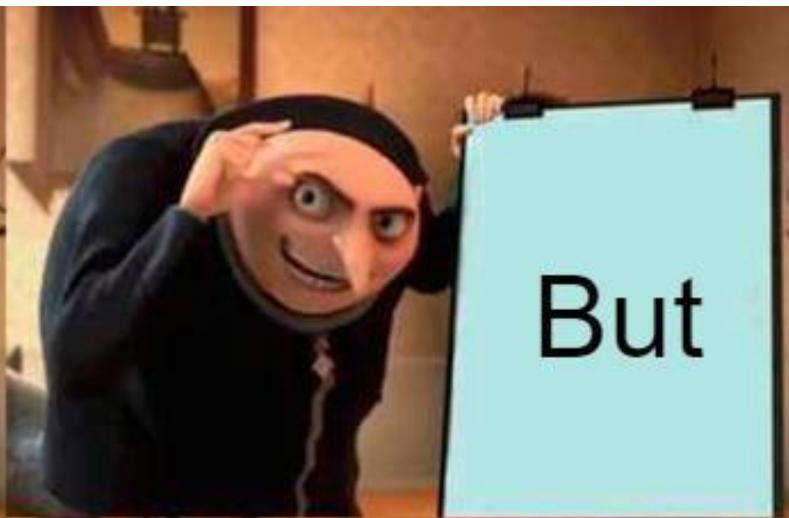
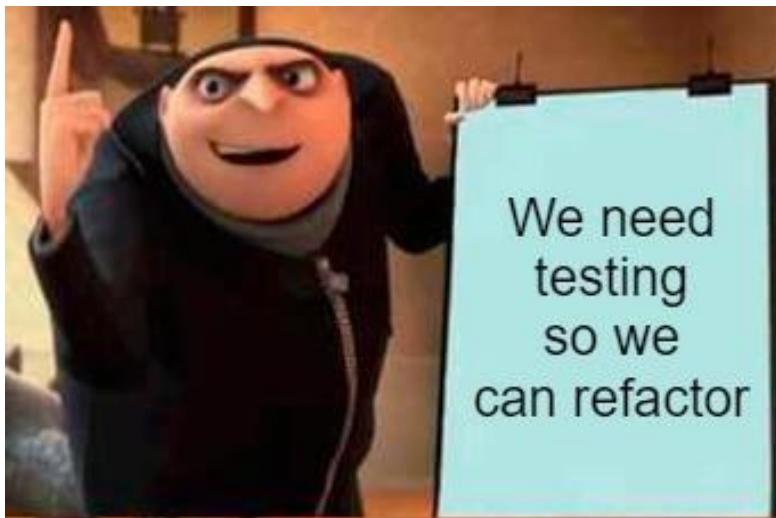
# Safety

**“All new code  
must be tested”**



# Nothing happened





2

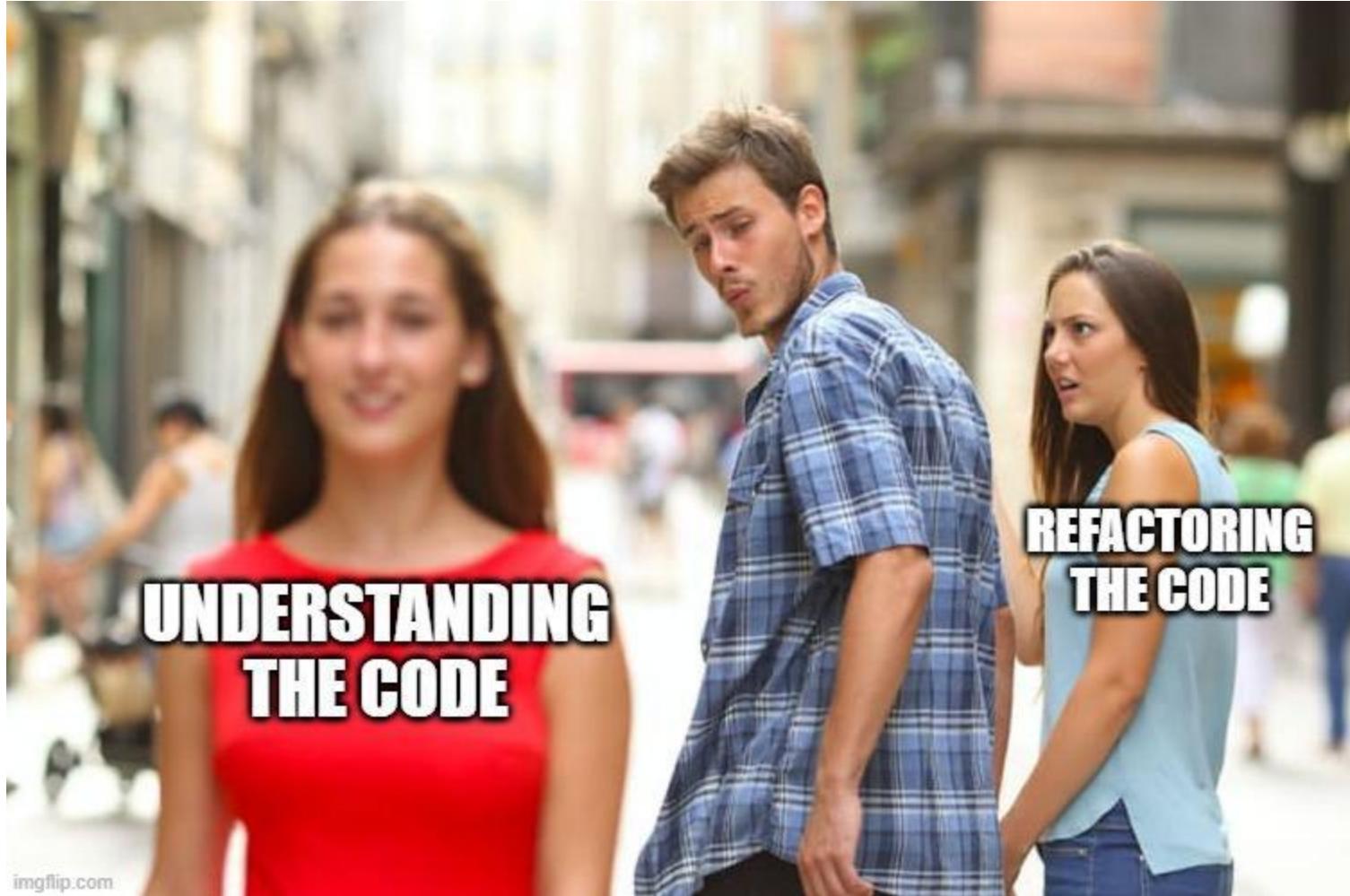
# The Code

**Each two weeks  
introduce a new  
code smell**



# **Still nothing happened**





3

# The Environment

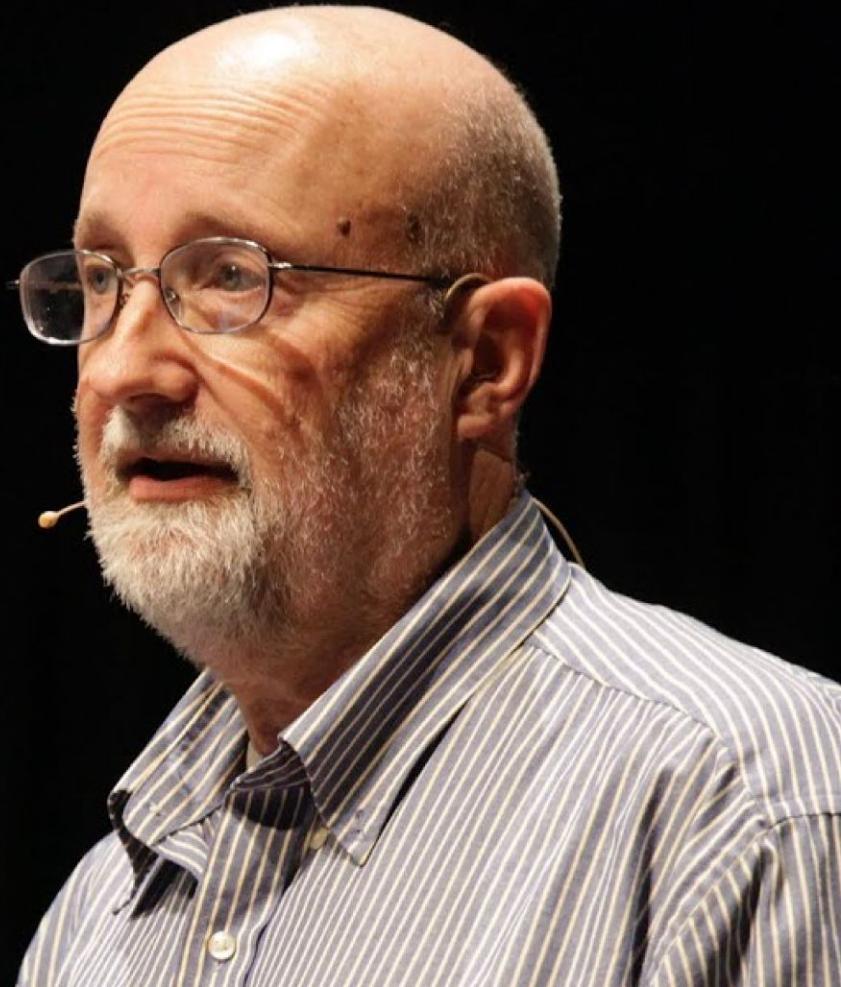


The object is *not* to  
create great software,  
but to create an **environment**  
where great software is inevitable.

---

*Estimates or NoEstimates*

*Woody Zuill (paraphrasing Robert Henri)*





**Introduce improvement monday**

09:00

Pitch and recruit, 09:00  
Define goal, 09:15

10:00

Focus  
09:30 – 11:30

11:00

Lunch, 11:30

12:00

Touch base, 12:00

13:00

Focus  
12:15 – 14:30

14:00

Mini review, 14:30  
Mini retro, 14:45

15:00

09:00

Pitch and recruit, 09:00

Define goal, 09:15

Focus

09:30 – 11:30

11:00

Lunch, 11:30

12:00

Touch base, 12:00

Focus

12:15 – 14:30

13:00

14:00

Mini review, 14:30

15:00

Mini retro, 14:45

Refactor code

Document new  
process

Learn React

Implement  
error handling

09:00

Pitch and recruit, 09:00

Define goal, 09:15

Focus

09:30 – 11:30

11:00

Lunch, 11:30

Touch base, 12:00

Focus

12:15 – 14:30

13:00

14:00

Mini review, 14:30

Mini retro, 14:45

Refactor code



Document new  
process



Learn React

Implement  
error handling

09:00

Pitch and recruit, 09:00

Define goal, 09:15

10:00

Focus

09:30 – 11:30

11:00

Lunch, 11:30

12:00

Touch base, 12:00

13:00

Focus

12:15 – 14:30

14:00

Mini review, 14:30

15:00

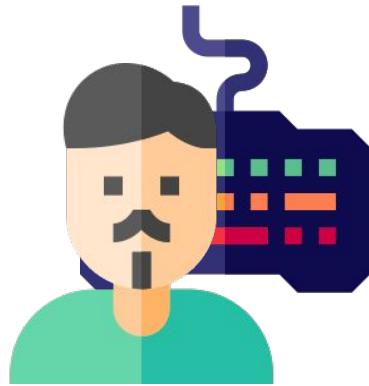
Mini retro, 14:45

Refactor code

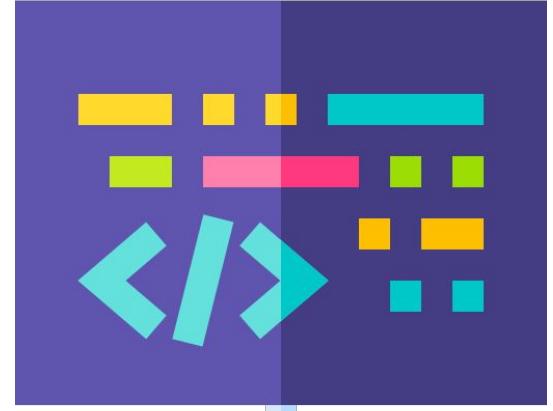
Refactor  
Class.java

Document new  
process

Write section  
about [...]



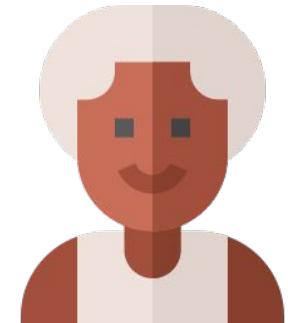
**Driver**



**Ensemble**



3-6 min



**Navigator**

09:00

Pitch and recruit, 09:00  
Define goal, 09:15

10:00

Focus  
09:30 – 11:30

11:00

Lunch, 11:30

12:00

Touch base, 12:00

13:00

Focus  
12:15 – 14:30

14:00

Mini review, 14:30  
Mini retro, 14:45

Refactor code

Refactor  
Class.java

Document new  
process

Write section  
about [...]

09:00

Pitch and recruit, 09:00  
Define goal, 09:15

10:00

Focus  
09:30 – 11:30

11:00

Lunch, 11:30

12:00

Touch base, 12:00

13:00

Focus  
12:15 – 14:30

14:00

Mini review, 14:30  
Mini retro, 14:45

15:00

Refactor code

Refactor  
Class.java

Document new  
process

Write section  
about [...]



Loved



Learned



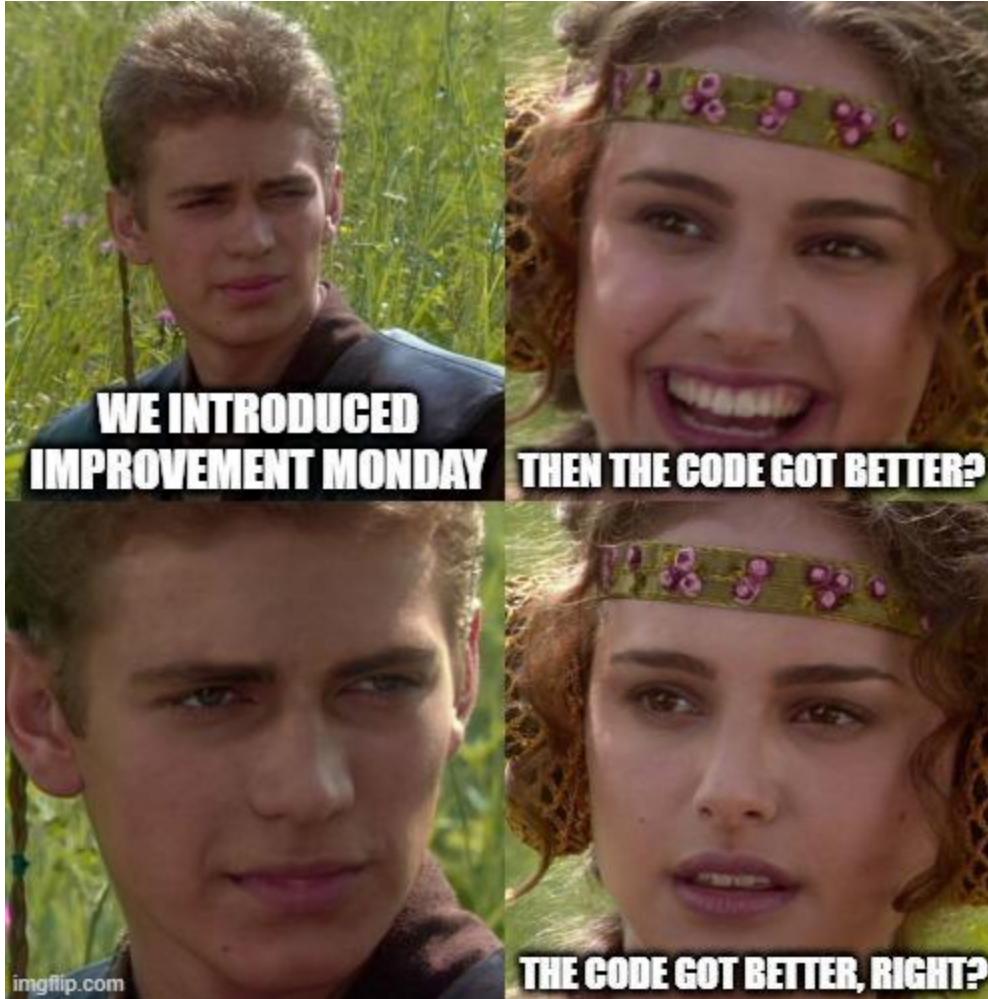
Lacked

*Working  
together*

*New  
shortcuts*

*Too many  
interruptions!*





# **Still nothing happened**





# 4

# The Smells

# Convert smells to rules





**Easy to  
remember**

A photograph of a man with a dark beard and mustache, wearing large, bright orange plastic glasses and a blue and white checkered shirt. He is holding a small, orange rectangular object, possibly a book or a folder, in front of his chest. The background is a solid dark grey.

**Syntactic, ie. eye catching**



**Safe to follow,  
even without  
understanding  
the code**

# Create safer refactoring patterns





**Copy and paste**

A composite image featuring a man with a full, dark brown beard and mustache, and a woman's face framed by a hand. The man has light-colored eyes and is looking directly at the camera. The woman's face is visible through the fingers of a hand held in front of her. The background is a dense, green, out-of-focus foliage.

**Small steps**

# Utilizing the compiler

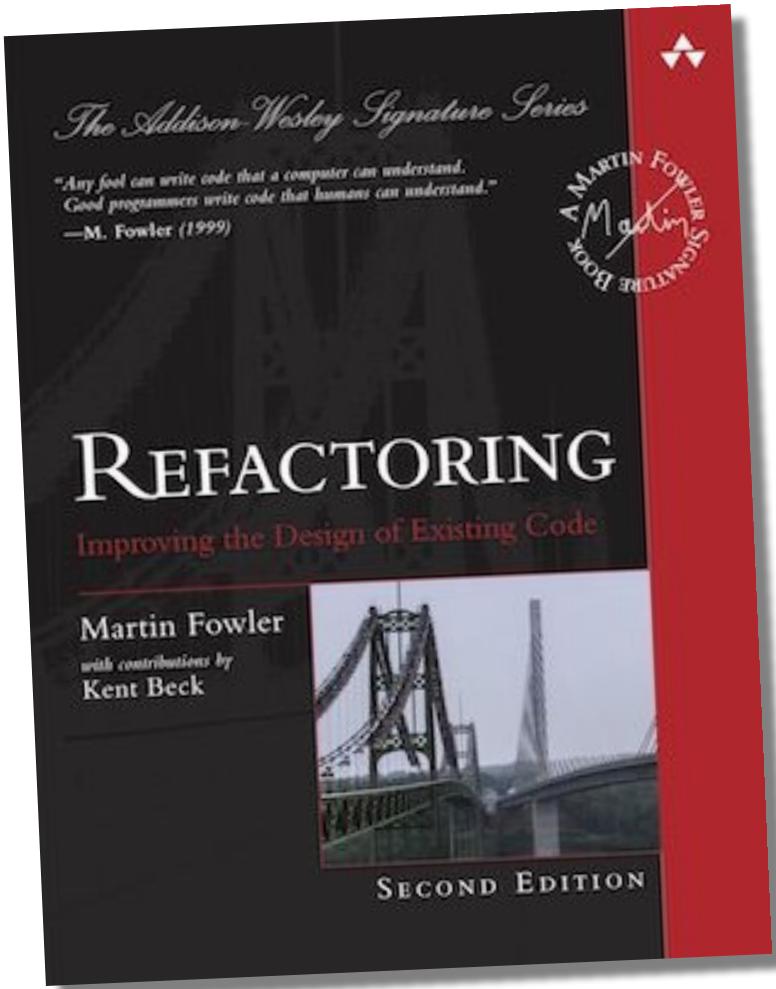




**The codebase started to improve**

# From Smells To Rules





Don't have long  
methods

The Addison Wesley

"Any fool can write code that a computer can understand.  
Good programmers write code that humans can understand."  
—M. Fowler (1999)

REFAC

Improving the

Martin Fowler  
with contributions by  
Kent Beck



# Five !!!! Lines

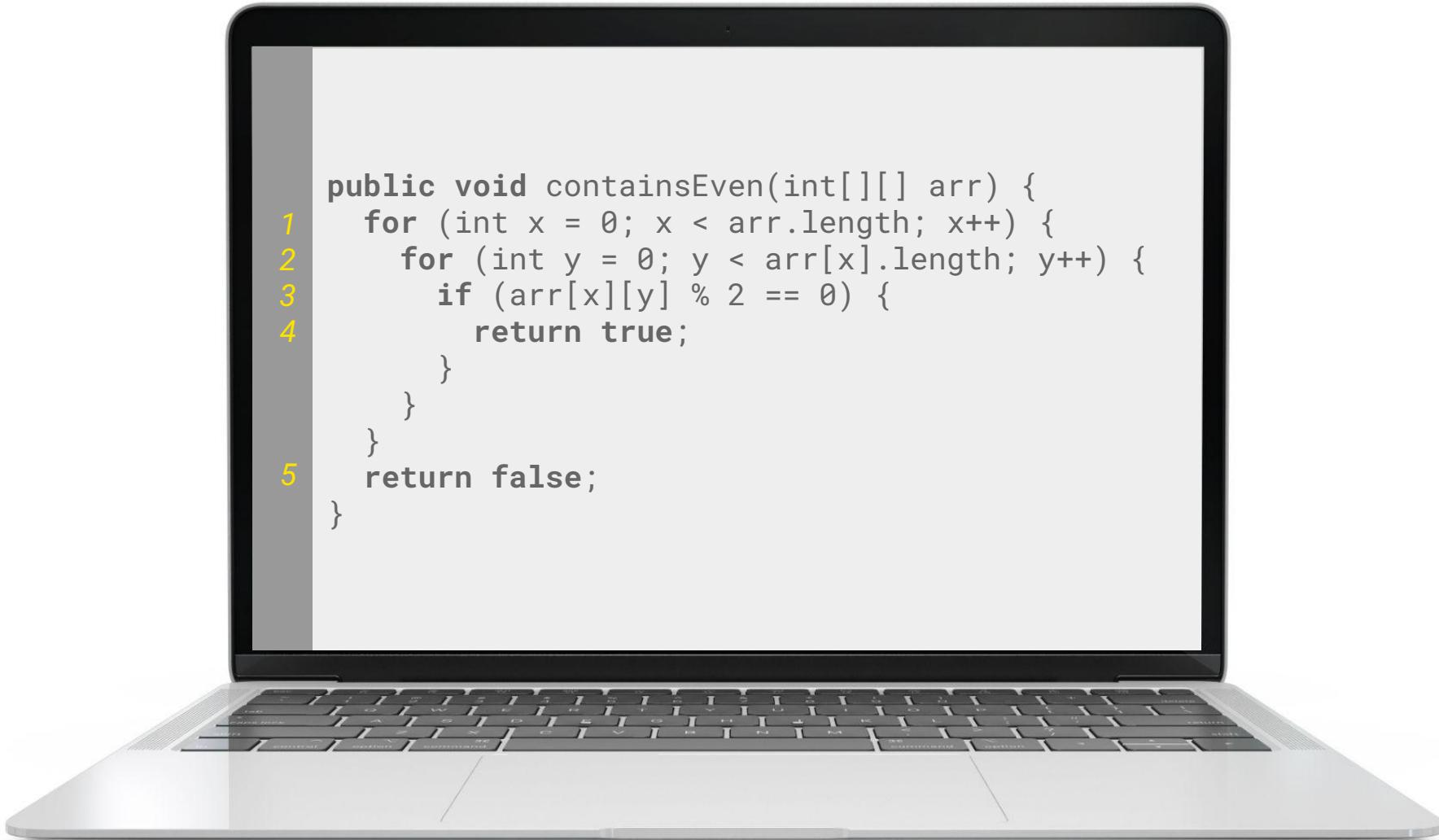
Session 1

STATEMENT  
A method should not contain more  
than five lines, excluding { and }.

ong  
S

@theDrLambda

```
public void containsEven(int[][] arr) {  
    1   for (int x = 0; x < arr.length; x++) {  
    2       for (int y = 0; y < arr[x].length; y++) {  
    3           if (arr[x][y] % 2 == 0) {  
    4               return true;  
    5           }  
    }  
    }  
    5   return false;  
}
```



```
enum Light {
    RED, YELLOW_RED, GREEN, YELLOW
}
function updateCar(car: Car, light: Light) {
    switch (light) {
        case Light.GREEN:
            car.drive();
            break;
        case Light.YELLOW_RED:
            car.putInGear();
            break;
        default:
            car.stop();
    }
}
```

```
enum Light {
    RED, YELLOW_RED, GREEN, YELLOW
}
function updateCar(car: Car, light: Light) {

    if (light === Light.GREEN)
        car.drive();

    else if (light === Light.YELLOW_RED)
        car.putInGear();

    else
        car.stop();

}
```

```
enum Light {
    RED, YELLOW_RED, GREEN, YELLOW
}
function updateCar(car: Car, light: Light) {
    if (light === Light.GREEN)
        car.drive();
    else if (light === Light.YELLOW_RED)
        car.putInGear();
    else
        car.stop();
}
```

```
enum Light_DELETEME {
    RED, YELLOW_RED, GREEN, YELLOW
}
function updateCar(car: Car, light: Light) {
    if (light === Light.GREEN)
        car.drive();
    else if (light === Light.YELLOW_RED)
        car.putInGear();
    else
        car.stop();
}
```

```
enum Light_DELETEME {
    RED, YELLOW_RED, GREEN, YELLOW
}
```

```
function updateCar(car: Car, light: Light) {
    if (light === Light.GREEN)
        car.drive();
    else if (light === Light.YELLOW_RED)
        car.putInGear();
    else
        car.stop();
}
```

```
interface Light {
    isRed(): boolean;
    isYellowRed(): boolean;
    isGreen(): boolean;
    isYellow(): boolean;
}
...
class Yellow implements Light {
    isRed() { return false; }
    isYellowRed() { return false; }
    isGreen() { return false; }
    isYellow() { return true; }
}
function updateCar(car: Car, light: Light) {
    if (light.isGreen())
        car.drive();
    else if (light.isYellowRed())
        car.putInGear();
    else
        car.stop();
}
```

```
interface Light {  
    ...  
}  
...  
class Yellow implements Light {  
    ...  
}  
function updateCar(car: Car, light: Light) {  
    if (light.isGreen())  
        car.drive();  
    else if (light.isYellowRed())  
        car.putInGear();  
    else  
        car.stop();  
}
```

```
interface Light {  
    ...  
    updateCar_tmp(car: Car): void;  
}  
...  
class Yellow implements Light {  
    ...  
    updateCar(car: Car) {  
        if (this.isGreen())  
            car.drive();  
        else if (this.isYellowRed())  
            car.putInGear();  
        else  
            car.stop();  
    }  
}  
function updateCar(car: Car, light: Light) {  
    light.updateCar_tmp(car);  
}
```

```
...
class Yellow implements Light {
...
isYellowRed() { return false; }
isGreen() { return false; }
updateCar(car: Car) {
    if (this.isGreen())
        car.drive();
    else if (this.isYellowRed())
        car.putInGear();
    else
        car.stop();
}
}
```

```
...
class Yellow implements Light {
...
isYellowRed() { return false; }
isGreen() { return false; }
updateCar(car: Car) {
    if (false)
        car.drive();
    else if (false)
        car.putInGear();
    else
        car.stop();
}
}
```

```
...
class Yellow implements Light {
...
updateCar(car: Car) {
    if (false)
        car.drive();
    else if (false)
        car.putInGear();
    else
        car.stop();
}
}
```

```
...
class Yellow implements Light {
...
updateCar_tmp(car: Car) {
    ...
    car.stop();
}
}
```

```
interface Light {  
    ...  
    updateCar_tmp(car: Car): void;  
}  
...  
class Yellow implements Light {  
    ...  
    updateCar_tmp(car: Car) {  
        ...  
    }  
}  
function updateCar(car: Car, light: Light) {  
    light.updateCar_tmp(car);  
}
```

```
interface Light {  
    ...  
    updateCar(car: Car): void;  
}  
...  
class Yellow implements Light {  
    ...  
    updateCar(car: Car) {  
        ...  
    }  
}  
function updateCar(car: Car, light: Light) {  
    light.updateCar(car);  
}
```

```
...
class Yellow implements Light {
    isRed() { return false; }
    isYellowRed() { return false; }
    isGreen() { return false; }
    isYellow() { return true; }
    ...
}
```

```
...
class Yellow implements Light {
    ...
}
```

```
...
class YellowRed implements Light {
    redLightOn() { return true; }
    yellowLightOn() { return true; }
    greenLightOn() { return false; }
    updateCar(car: Car) {
        car.putInGear();
    }
}
class Yellow implements Light {
    redLightOn() { return false; }
    yellowLightOn() { return true; }
    greenLightOn() { return false; }
    updateCar(car: Car) {
        car.stop();
    }
}
```

```
...
class YellowRed implements Light {
    redLightOn() { return true; }
    yellowLightOn() { return true; }
    greenLightOn() { return false; }
    updateCar(car: Car) {
        if (this.redLightOn())
            car.putInGear();
    }
}
class Yellow implements Light {
    redLightOn() { return false; }
    yellowLightOn() { return true; }
    greenLightOn() { return false; }
    updateCar(car: Car) {
        if (!this.redLightOn())
            car.stop();
    }
}
```

```
...
class YellowRed implements Light {
...
updateCar(car: Car) {
    if (this.redLightOn())
        car.putInGear();

}
class Yellow implements Light {
...
updateCar(car: Car) {

    if (!this.redLightOn())
        car.stop();
}
}
```

```
...
class YellowRed implements Light {
...
updateCar(car: Car) {
    if (this.redLightOn())
        car.putInGear();
    else if (!this.redLightOn())
        car.stop();
}
}
class Yellow implements Light {
...
updateCar(car: Car) {
    if (this.redLightOn())
        car.putInGear();
    else if (!this.redLightOn())
        car.stop();
}
}
```

```
...
class Yellow implements Light {
...
updateCar(car: Car) {
    if (this.redLightOn())
        car.putInGear();
    else if (!this.redLightOn())
        car.stop();
}
}
```

```
...
class Yellow implements Light {
...
updateCar(car: Car) {
    if (this.redLightOn())
        car.putInGear();
    else
        car.stop();
}
}
```

```
...
class Yellow implements Light {
    redLightOn() { return false; }
    ...
    updateCar(car: Car) {
        if (this.redLightOn())
            car.putInGear();
        else
            car.stop();
    }
}
```

```
...
class Yellow implements Light {
    constructor(private hasRed: boolean) { }
    redLightOn() { return this.hasRed; }
    ...
    updateCar(car: Car) {
        if (this.hasRed)
            car.putInGear();
        else
            car.stop();
    }
}
```

```
...
class YellowRed implements Light {
    ...
}
class Yellow implements Light {
    ...
}
```

```
...
class Yellow implements Light {
    ...
}
```

```
...
class Yellow implements Light {
    ...
    updateCar(car: Car) {
        if (this.hasRed)
            car.putInGear();
        else
            car.stop();
    }
}
```



# Type Code Elimination

```
...  
class Yellow  
...  
updateCar(c  
  if (this.  
    car.put  
  else  
    car.stop  
}  
}
```

## FLOW

(Extract method)\*

**switch** ⇒ **if-else** chain

**if (\_\_. == \_\_)** ⇒ Replace type code with classes\*

**if (\_\_.is\_\_())** ⇒ Push code into classes\*

(Inline method)\*

A photograph of a young man with dark hair and glasses, wearing a red polo shirt. He is covering his mouth with both hands, looking directly at the camera with a surprised or shocked expression.

# “Invariant”

Something that the developer knows  
but the compiler does not.

Example:  
`array.length` cannot be negative.

# “Coupling”

Code shared by multiple teams.

Example:  
Utility and data classes.



Coupling:  
A numeric  
array

Coupling:  
Probably C#

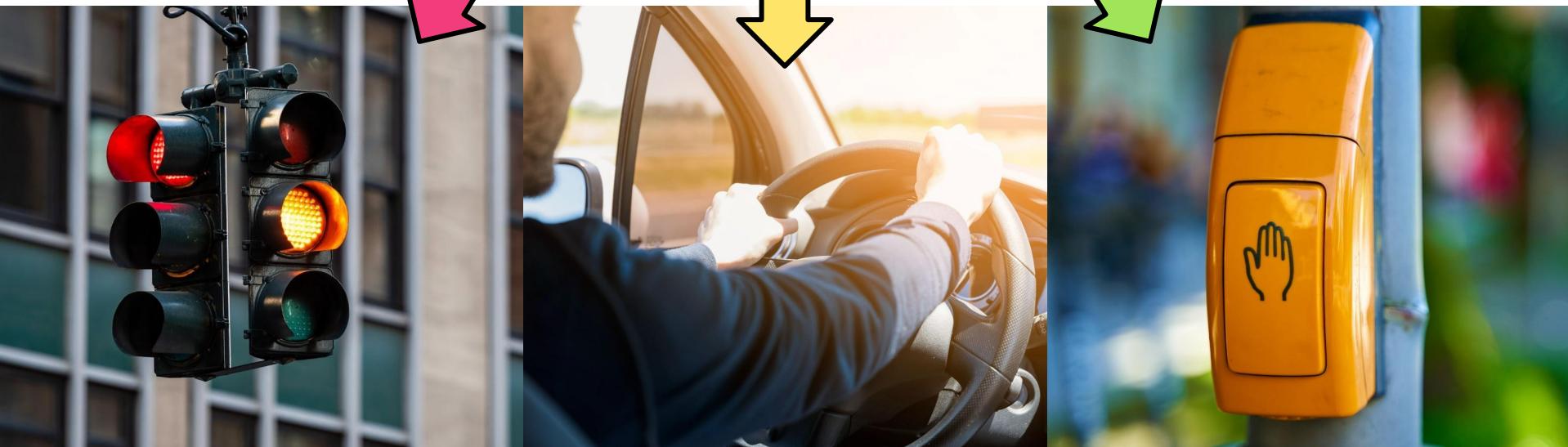
```
for (int i = 0; i < arr.Length; i++)  
{  
    arr[i] += 2;  
}
```

```
arr.Map(x => x + 2);
```

Invariant:  
.Length is  
well-behaved

Invariant:  
No overflow

# enum Light





class Light



# We did become the happiest team



# Trail map



## 0. Technical Vision in the Team

With a tech lead or similar



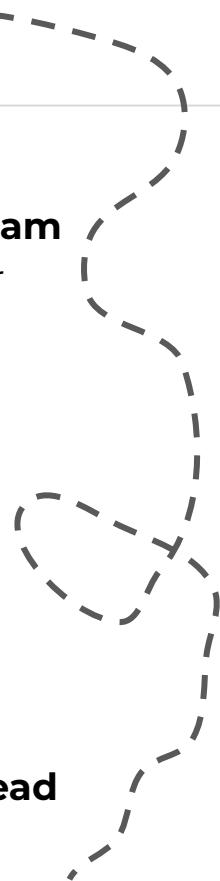
## 2. Remove Waste

Comments  
Interruptions  
Bus factor



## 4. Reduce Cognitive Overhead

Simpler rules  
Simpler code



## 1. Environment for Excellence

Improvement monday  
Pair programming  
Ensemble programming



## 3. Prioritize Safety

Small batches  
Use the tools we have  
Localize invariants

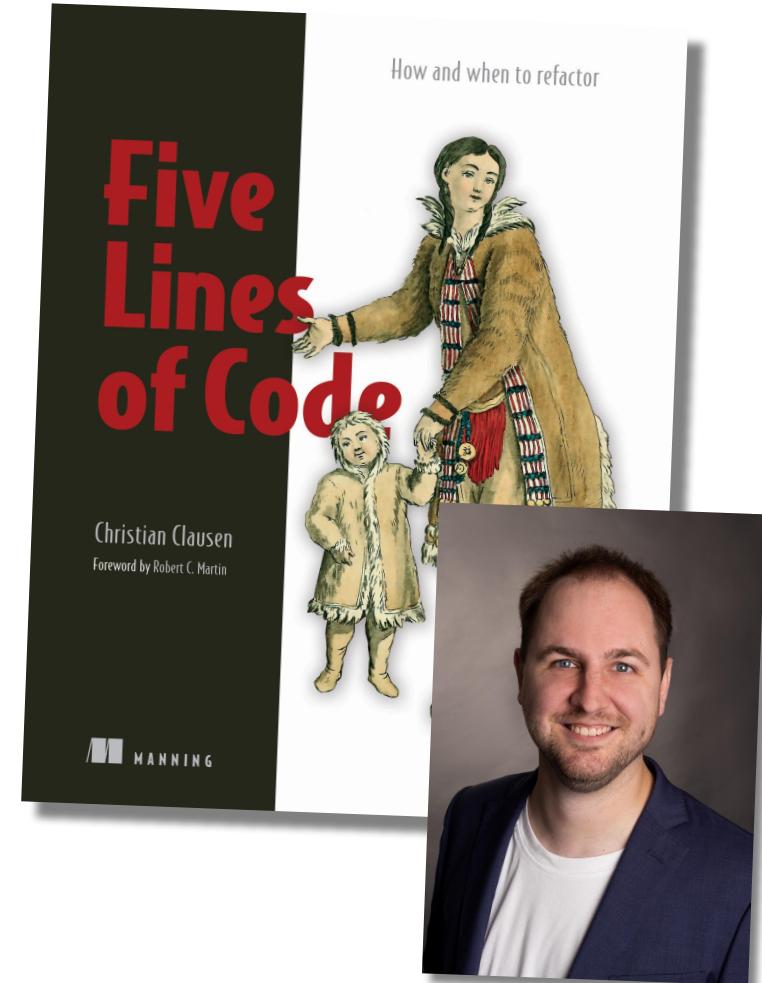


# Thank you!

## Questions?

Christian Clausen

@theDrLambda

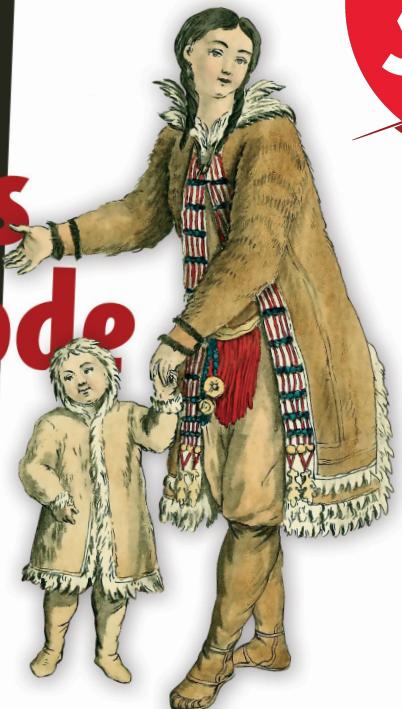


# Five Lines of Code

Christian Clausen  
Foreword by Robert C. Martin



How and when to refactor



35%  
OFF



goto;

Don't forget to  
**rate this session**  
in the **GOTO Guide app**