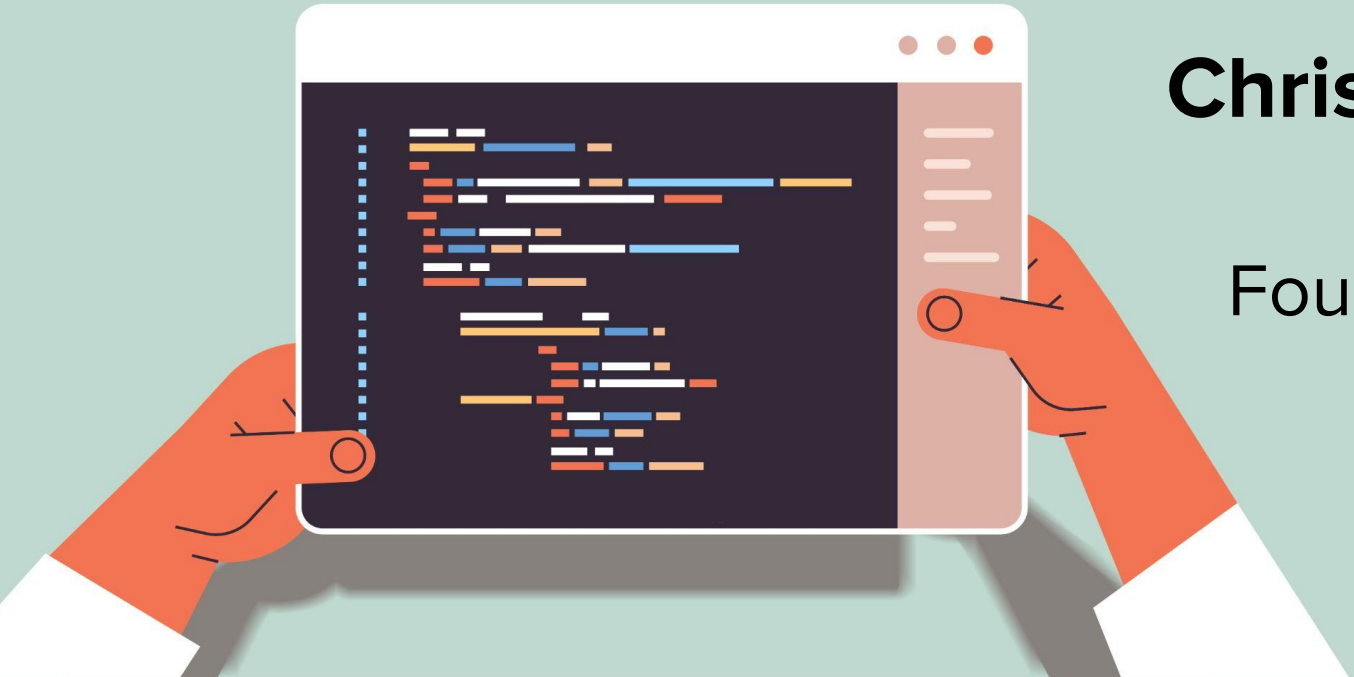
goto;

GOTO
Copenhagen 2022

#GOTOcph

Five Lines of Code



Christian Clausen

Founder, mist-cloud

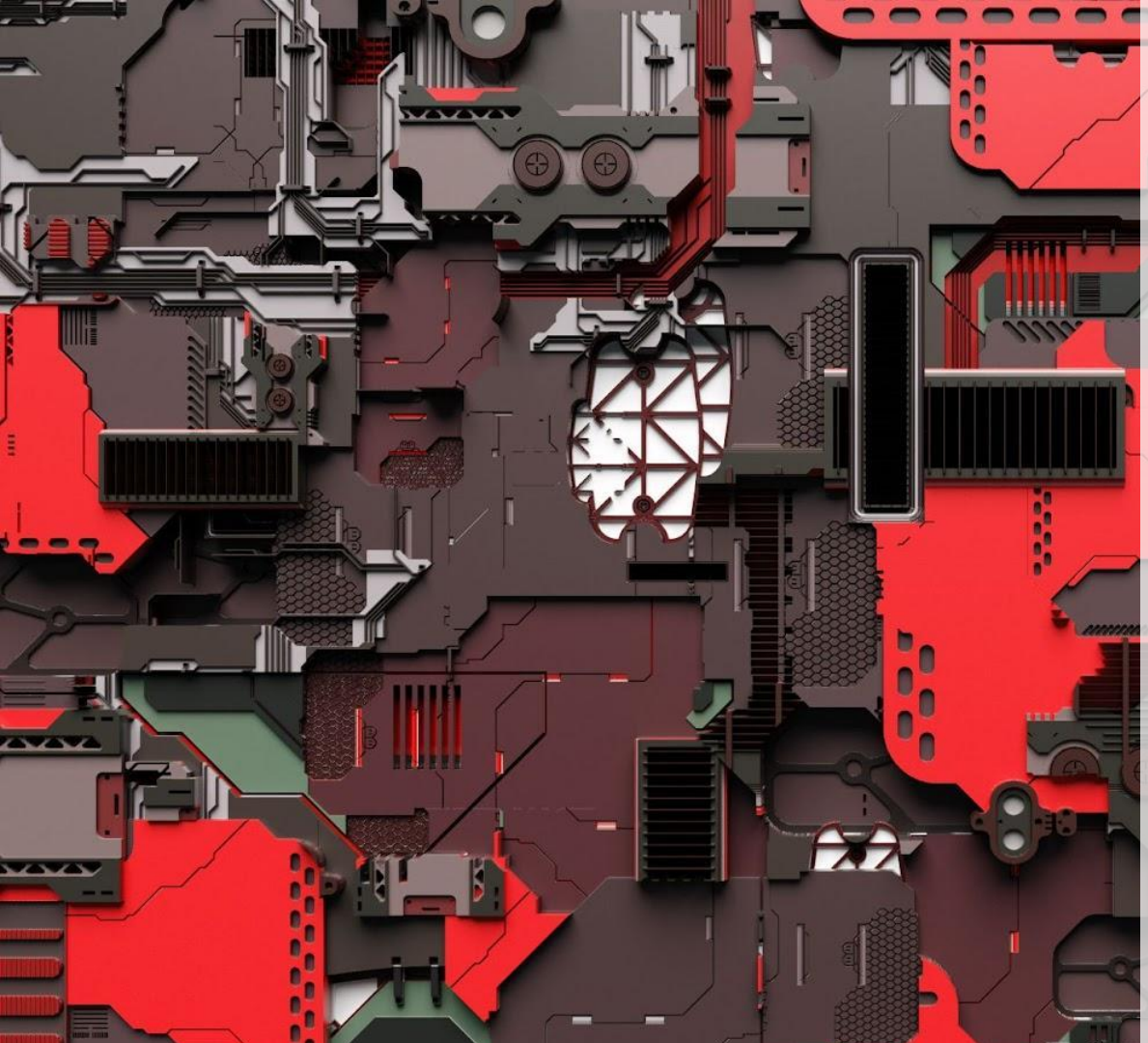
@theDrLambda

Technical Agile Coach


Technical Agile Coach

Technical Agile Coach

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

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”

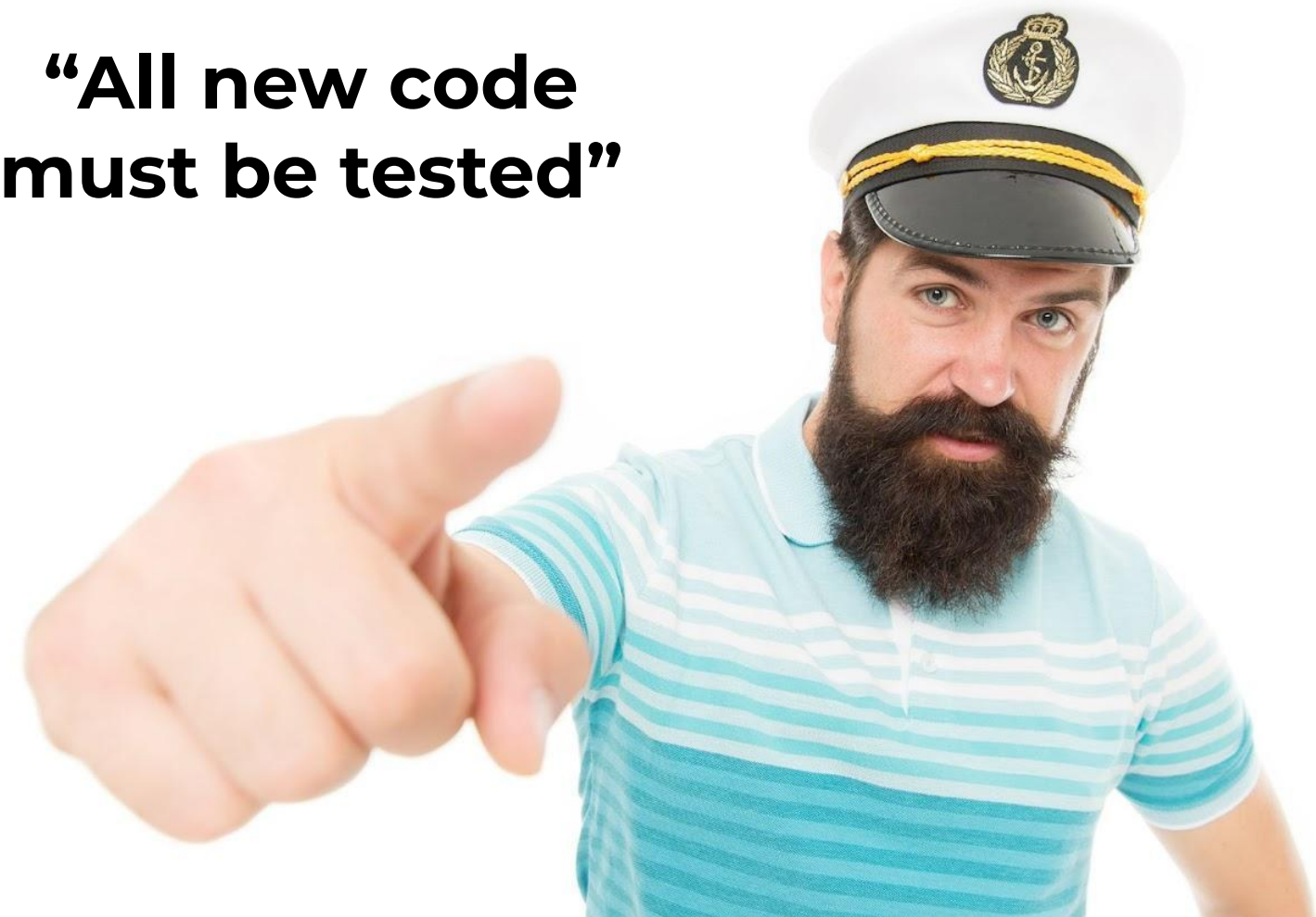
– Llewellyn’s strong-style pairing

http://llewellynfalco.blogspot.com/2014/06/llewellyns-strong-style-pairing.html

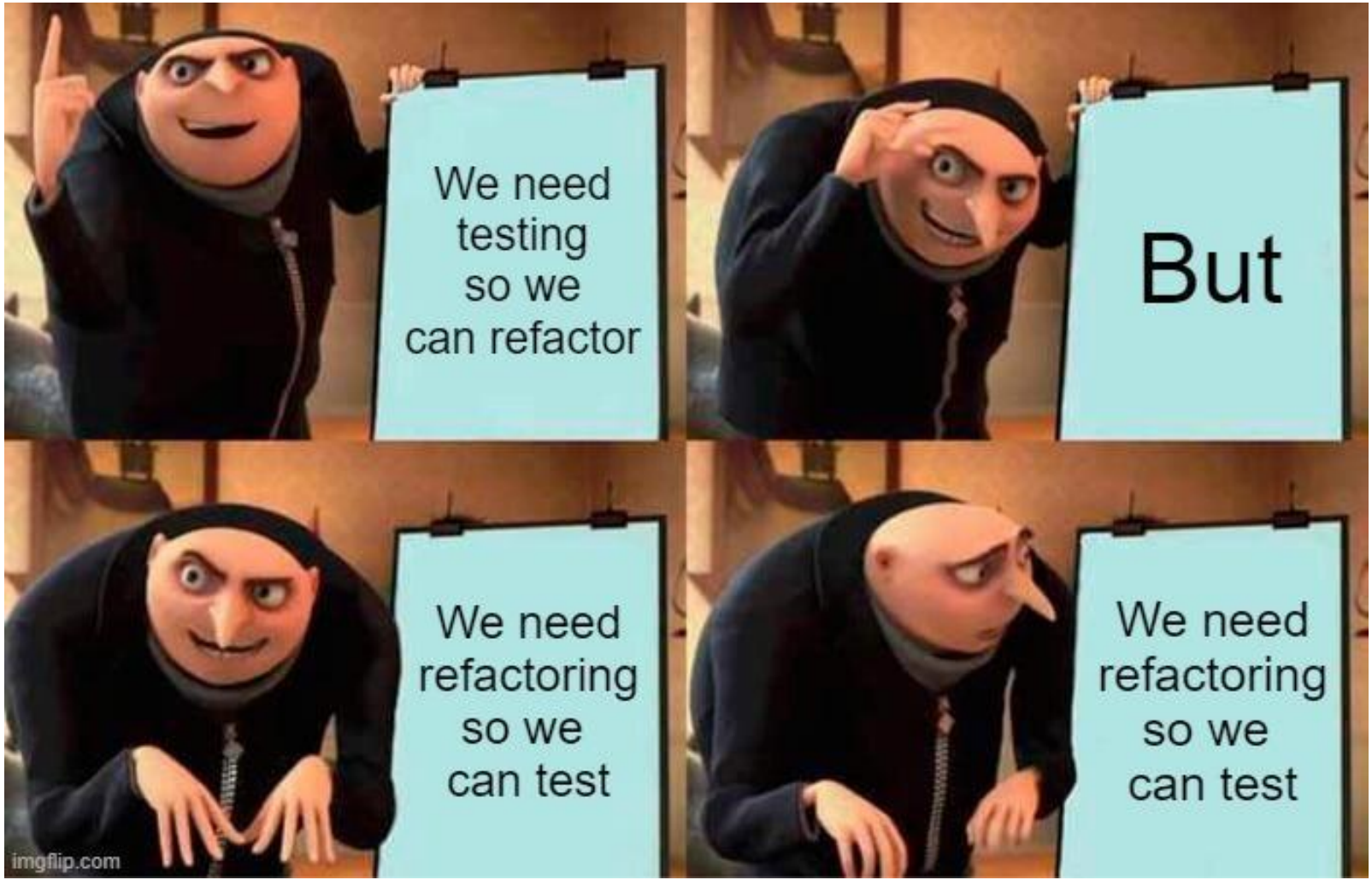
1

Safety

“All new code must be tested”



Nothing happened

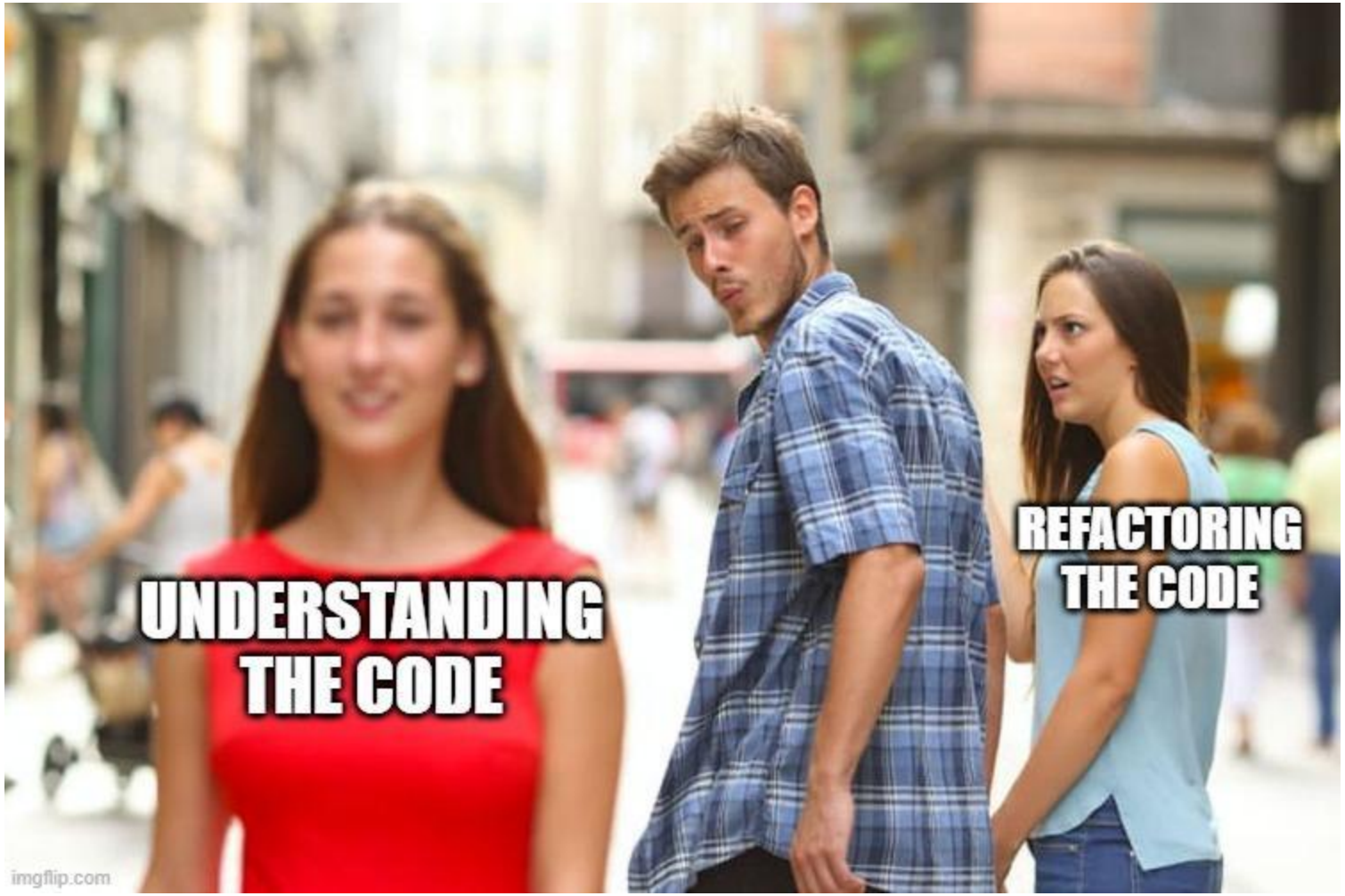

We need testing so we can refactor
But
We need refactoring so we can test
We need refactoring so we can test
imgflip.com

2

The Code

Each two weeks introduce a new code smell

Still nothing happened


UNDERSTANDING
THE CODE
REFACTORING
THE CODE
imgflip.com

3

The Environment

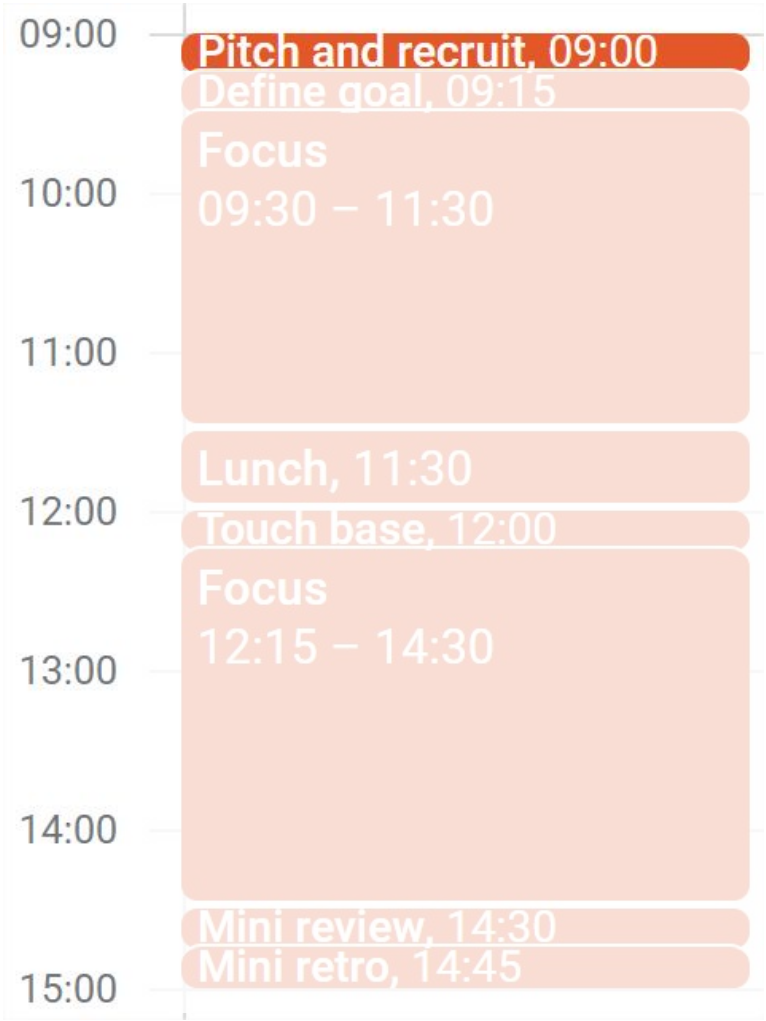
The object is *not* to
create great software,
but to create an **environment**
where great software is inevitable.

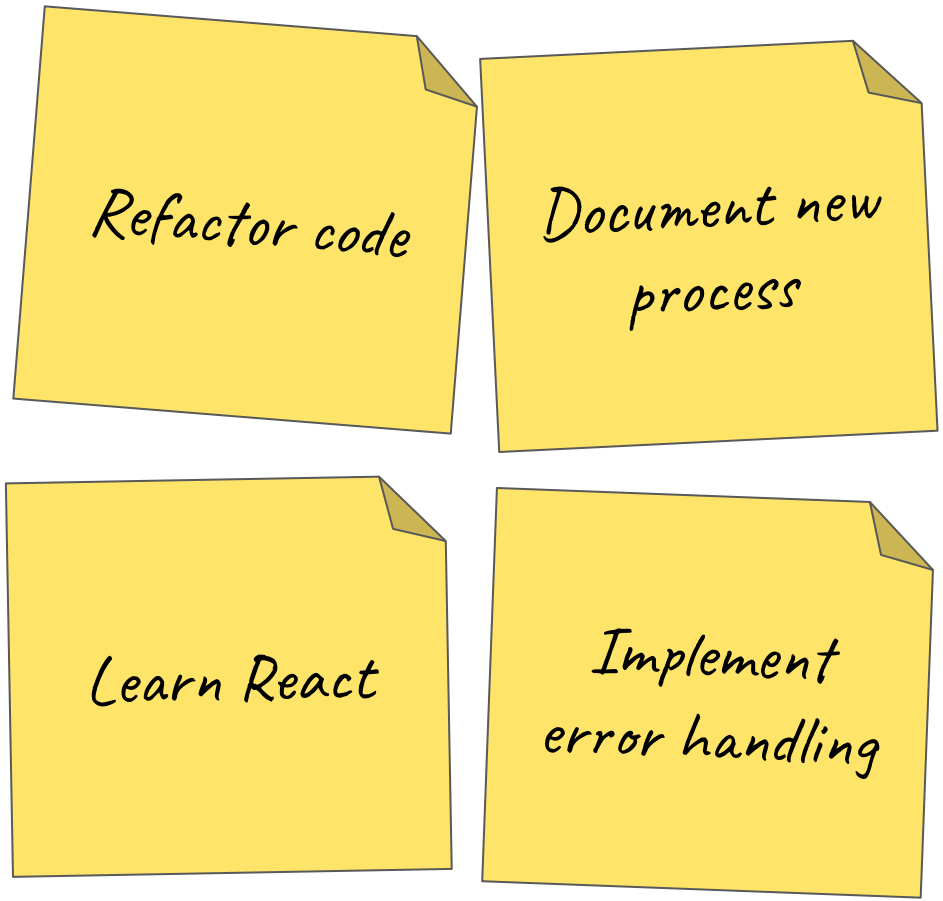
Estimates or NoEstimates

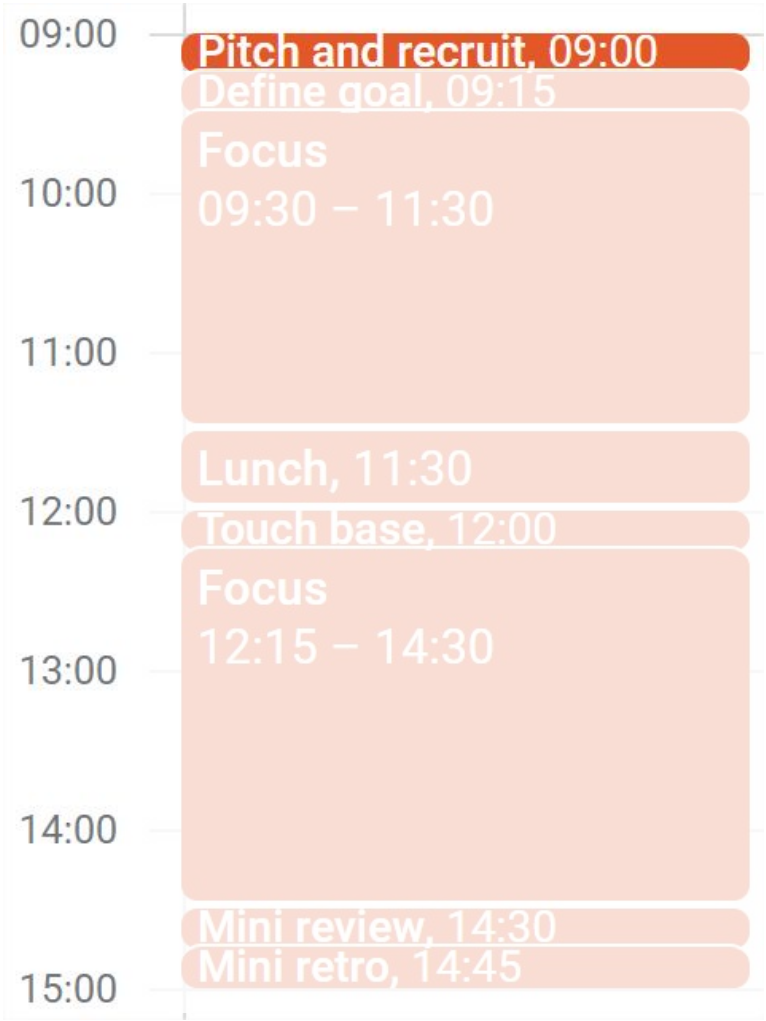
Woody Zuill (paraphrasing Robert Henri)

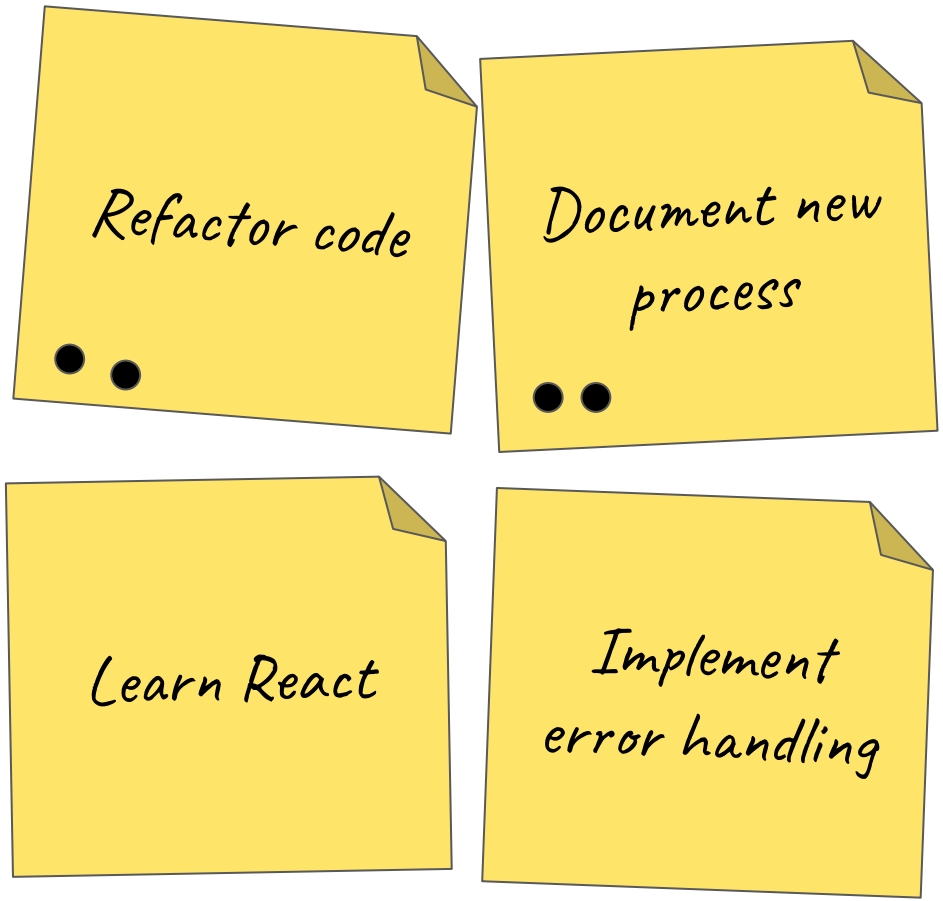
Introduce improvement monday


09:00
Pitch and recruit, 09:00
Define goal, 09:15
Focus
09:30 – 11:30
10:00
11:00
Lunch, 11:30
12:00
Touch base, 12:00
Focus
12:15 – 14:30
13:00
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
10:00
11:00
Lunch, 11:30
12:00
Touch base, 12:00
Focus
12:15 – 14:30
13:00
14:00
Mini review, 14:30
Mini retro, 14:45
15:00


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
10:00
11:00
Lunch, 11:30
12:00
Touch base, 12:00
Focus
12:15 – 14:30
13:00
14:00
Mini review, 14:30
Mini retro, 14:45
15:00


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

10:00

11:00

Lunch, 11:30

12:00 Touch base, 12:00

Focus
12:15 – 14:30

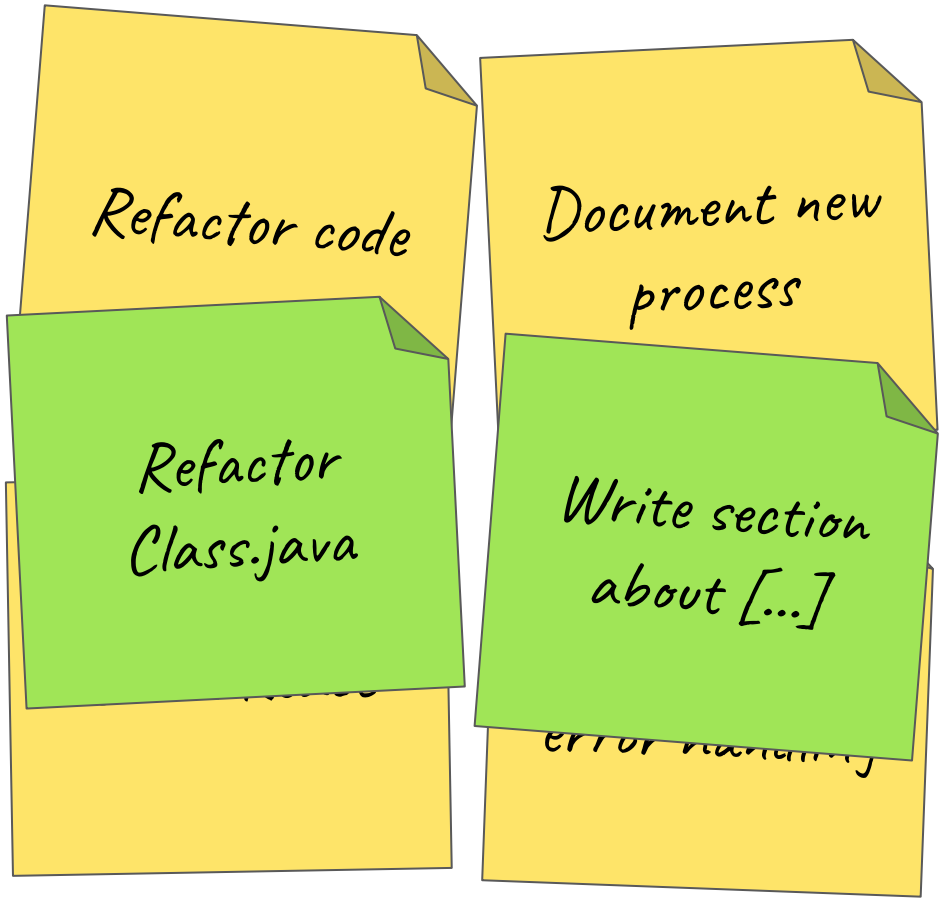
13:00

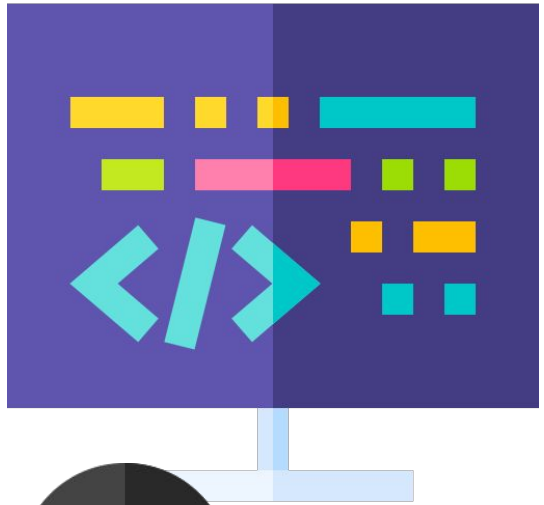
14:00

Mini review, 14:30

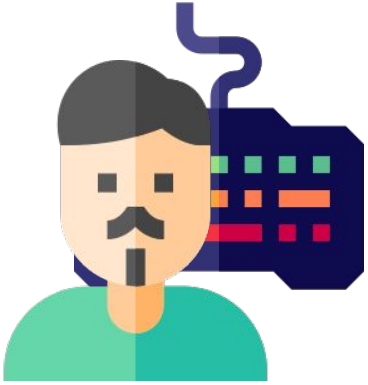
Mini retro, 14:45

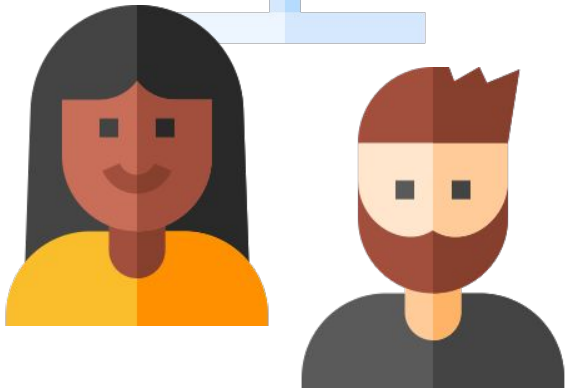
15:00

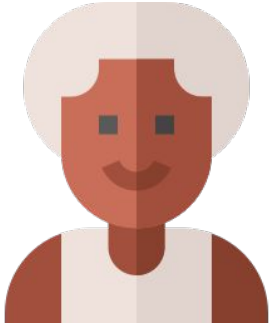




3-6 min





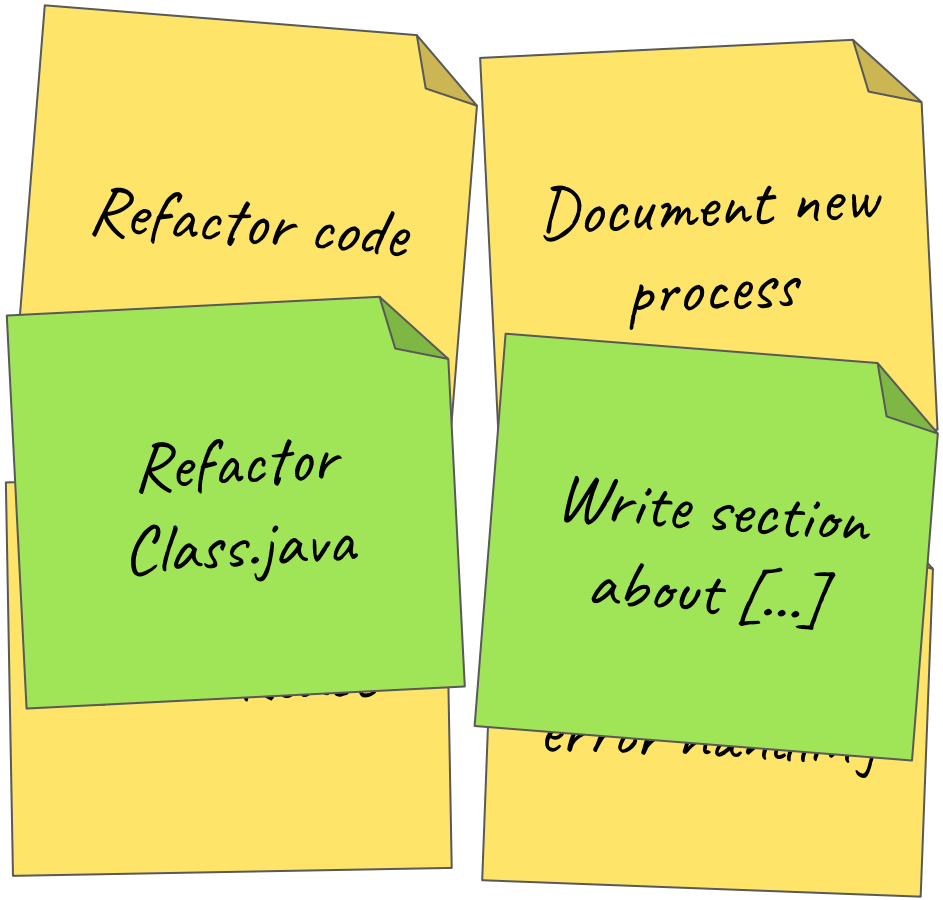


Driver

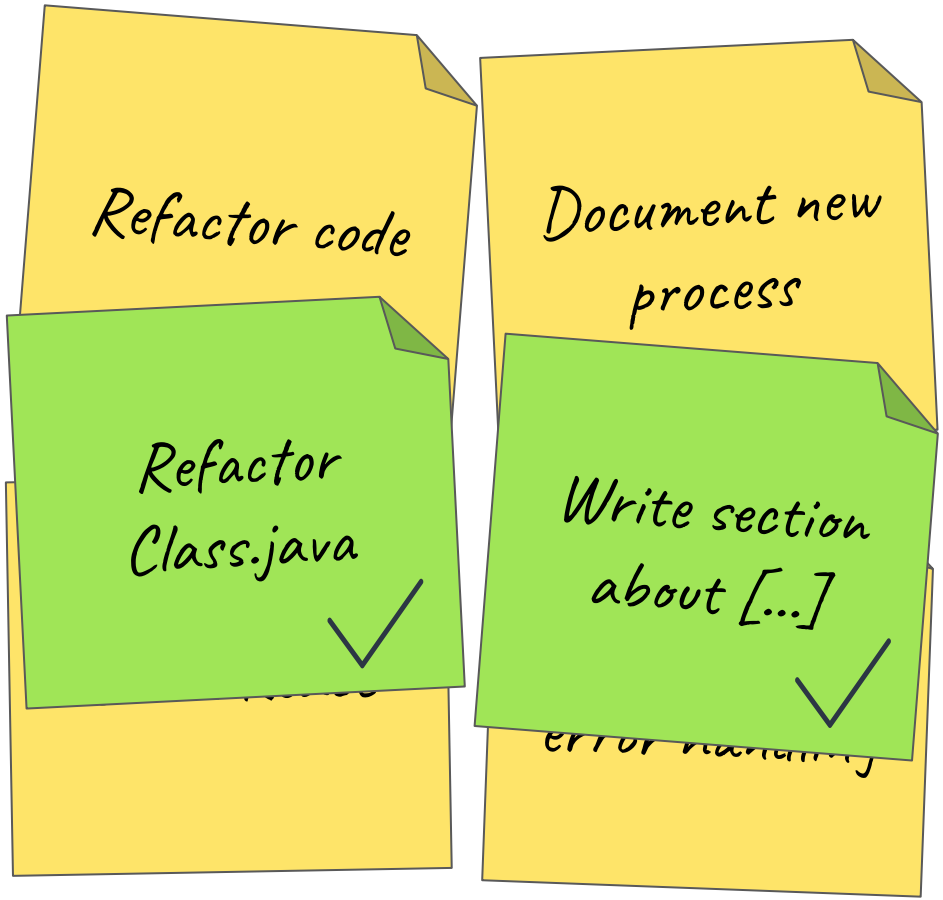
Ensemble

Navigator

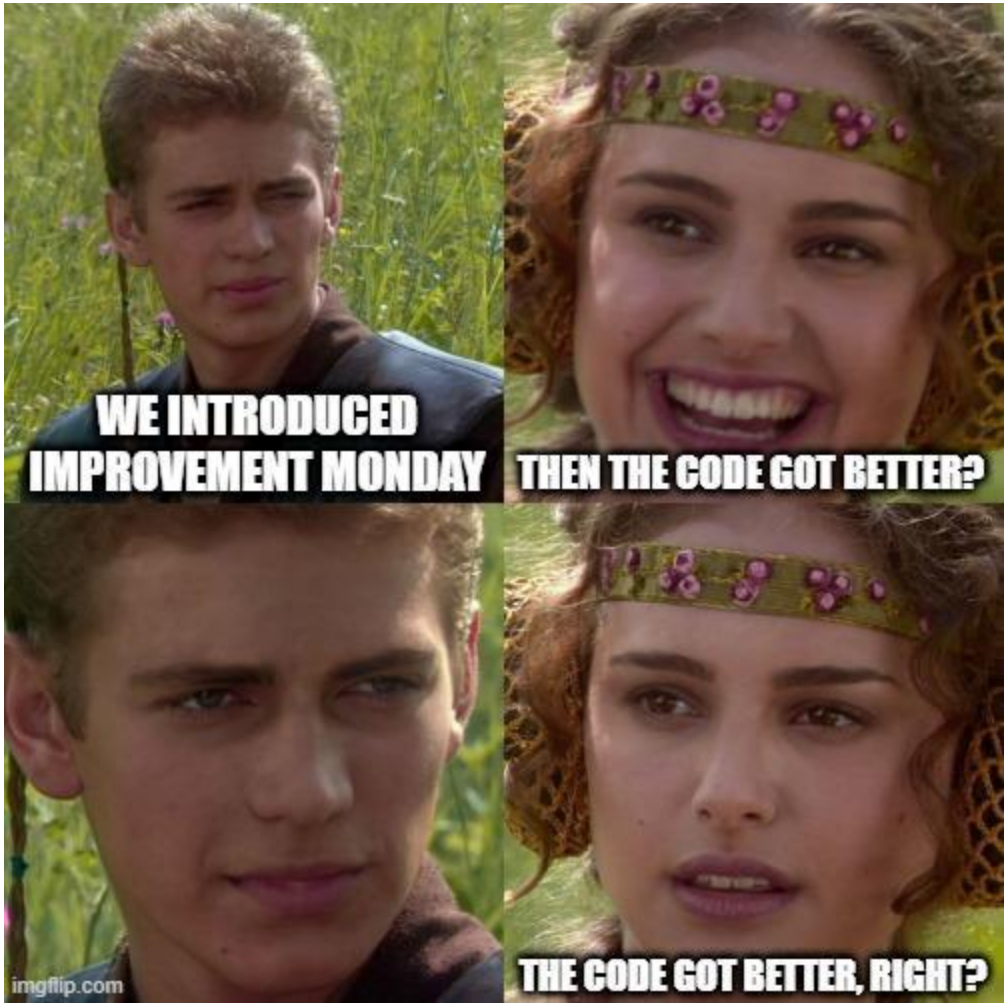
09:00
Pitch and recruit, 09:00
Define goal, 09:15
Focus
09:30 – 11:30
10:00
11:00
Lunch, 11:30
12:00
Touch base, 12:00
Focus
12:15 – 14:30
13:00
14:00
Mini review, 14:30
Mini retro, 14:45
15:00



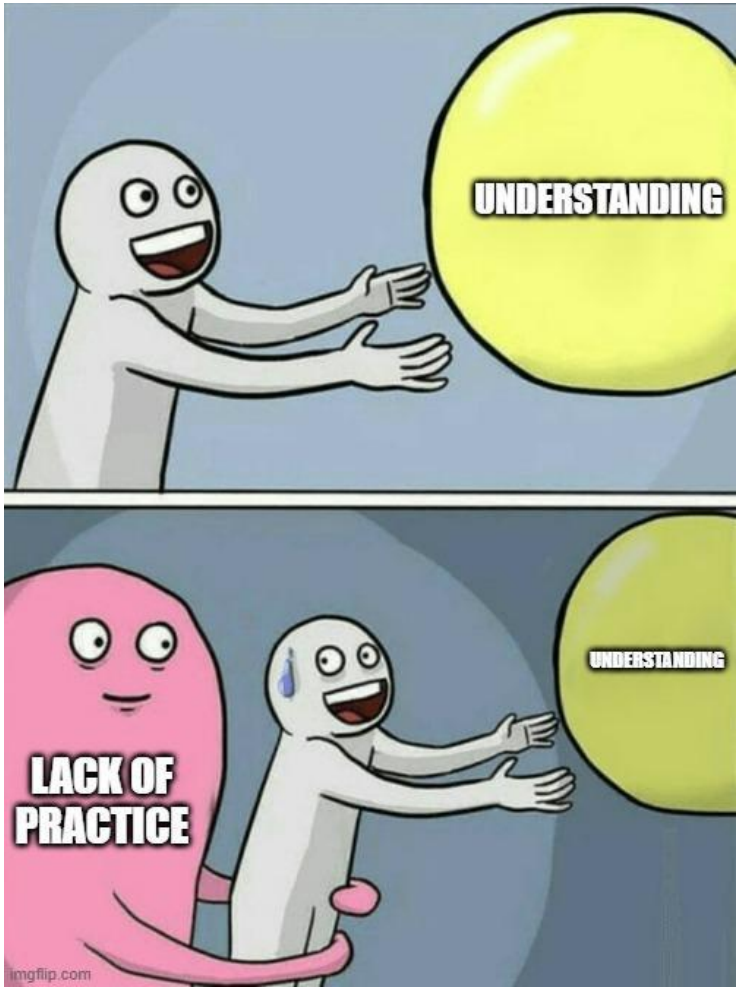
09:00
Pitch and recruit, 09:00
Define goal, 09:15
Focus
09:30 – 11:30
10:00
11:00
Lunch, 11:30
12:00
Touch base, 12:00
Focus
12:15 – 14:30
13:00
14:00
Mini review, 14:30
Mini retro, 14:45
15:00



Loved	Learned	Lacked
Working together	New shortcuts	Too many interruptions!


WE INTRODUCED IMPROVEMENT MONDAY
THEN THE CODE GOT BETTER?
THE CODE GOT BETTER, RIGHT?
imgflip.com

Still nothing happened


UNDERSTANDING
LACK OF PRACTICE
UNDERSTANDING
imgflip.com

4

The **Smells**

Convert smells to rules

Easy to remember

Syntactic, ie. eye catching

Safe to follow, even without understanding the code

Create safer refactoring patterns

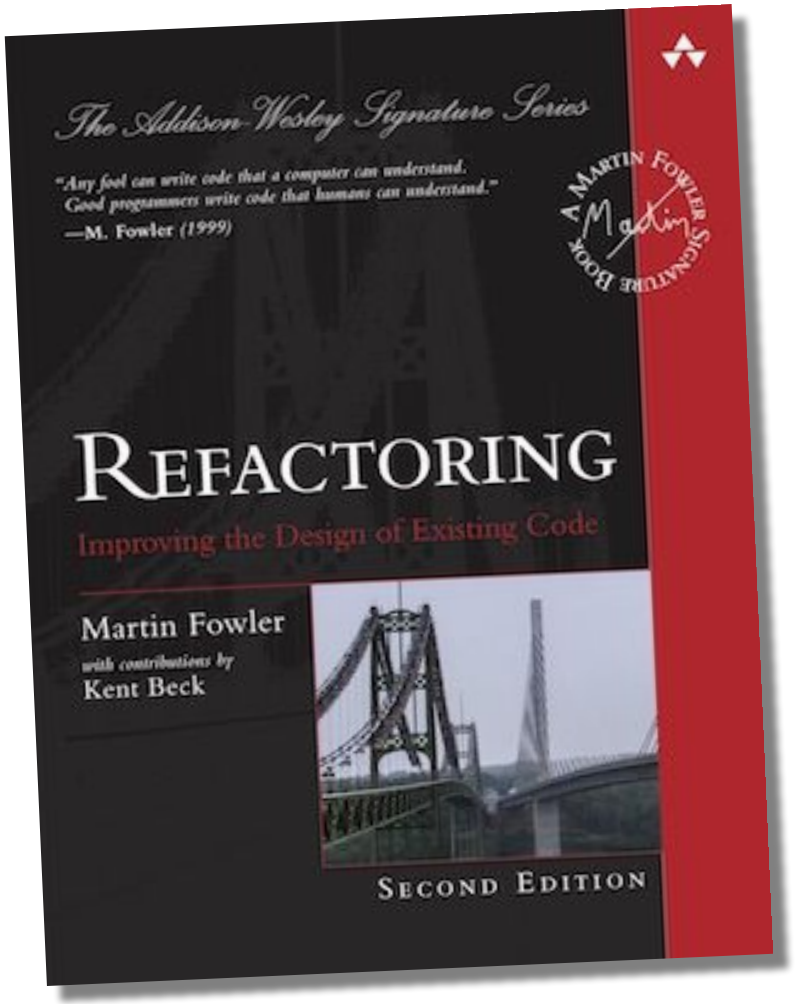
Copy and paste

Small steps

Utilizing the compiler

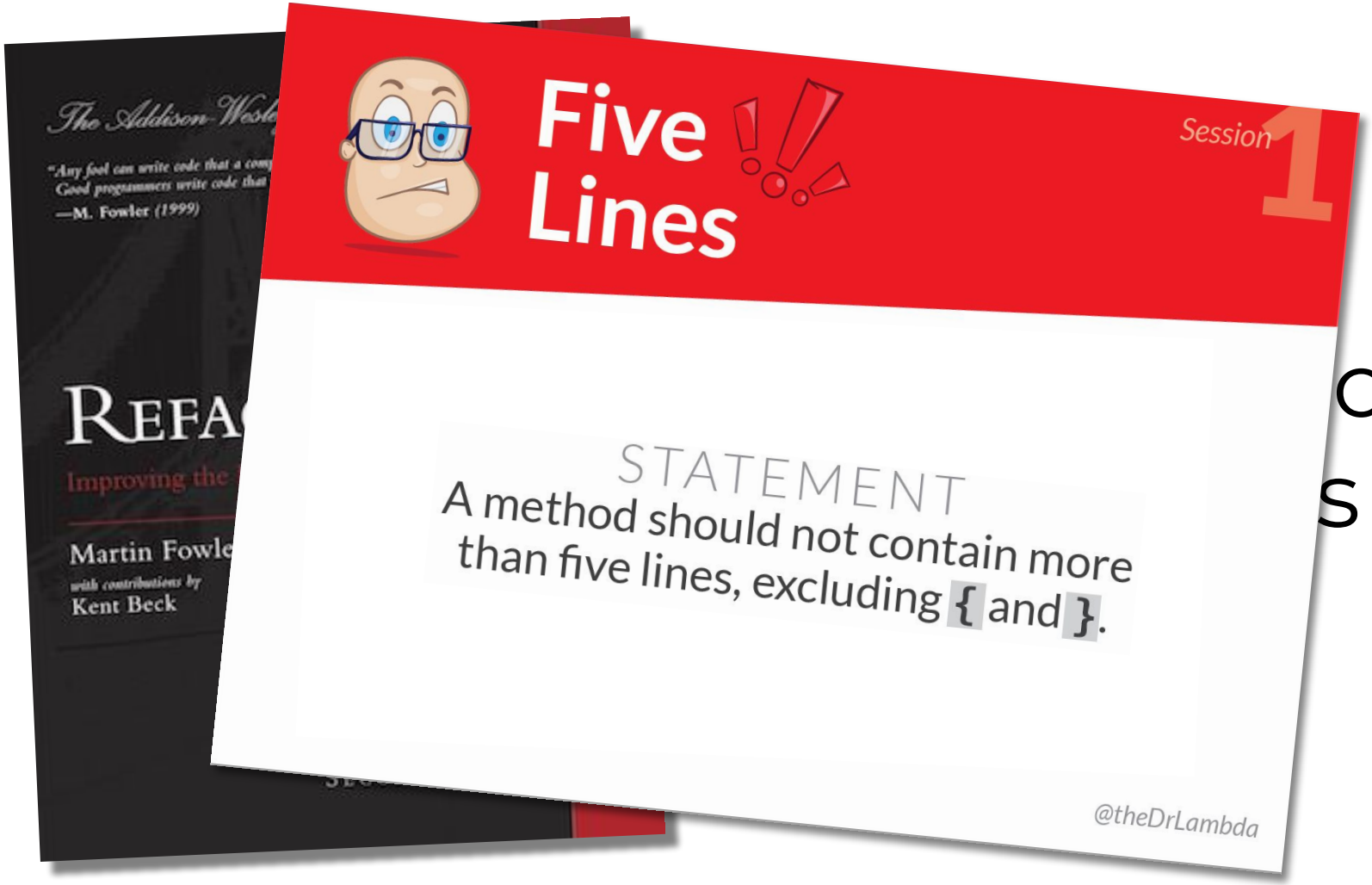
The codebase started to improve

From Smells To Rules



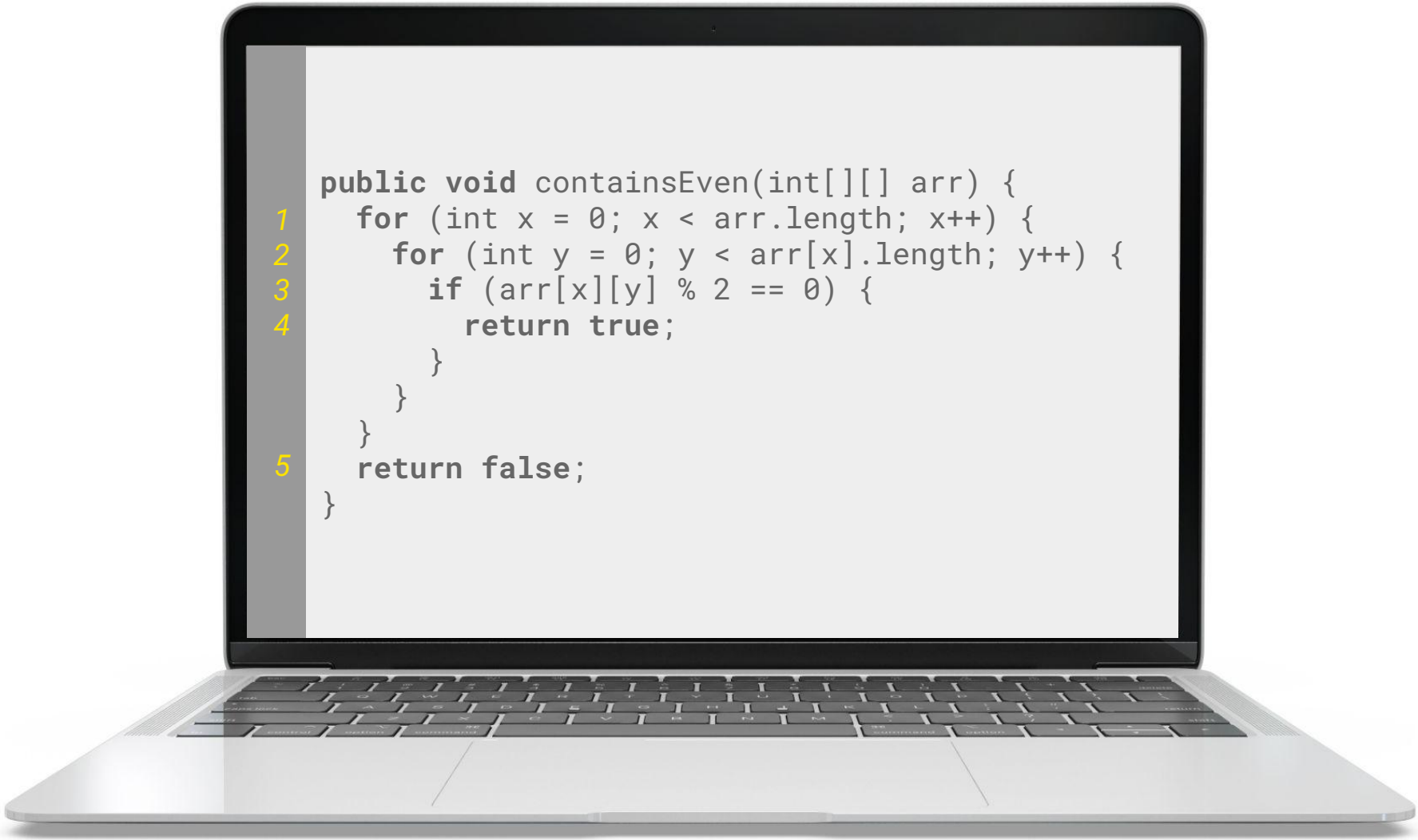
“

Don’t have long methods


Five Lines
Session 1
STATEMENT
A method should not contain more than five lines, excluding { and }.
@theDrLambda
Martin Fowle
with contributions by
Kent Beck

ong

s


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 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();
  }
}
```

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

Type Code Elimination

Session 2

FLOW

(Extract method)*

`switch` ⇒ `if-else` chain

`if (__ == __)` ⇒ Replace type code with classes*

`if (__.is__())` ⇒ Push code into classes*

(Inline method)*

@theDrLambda

“Invariant”

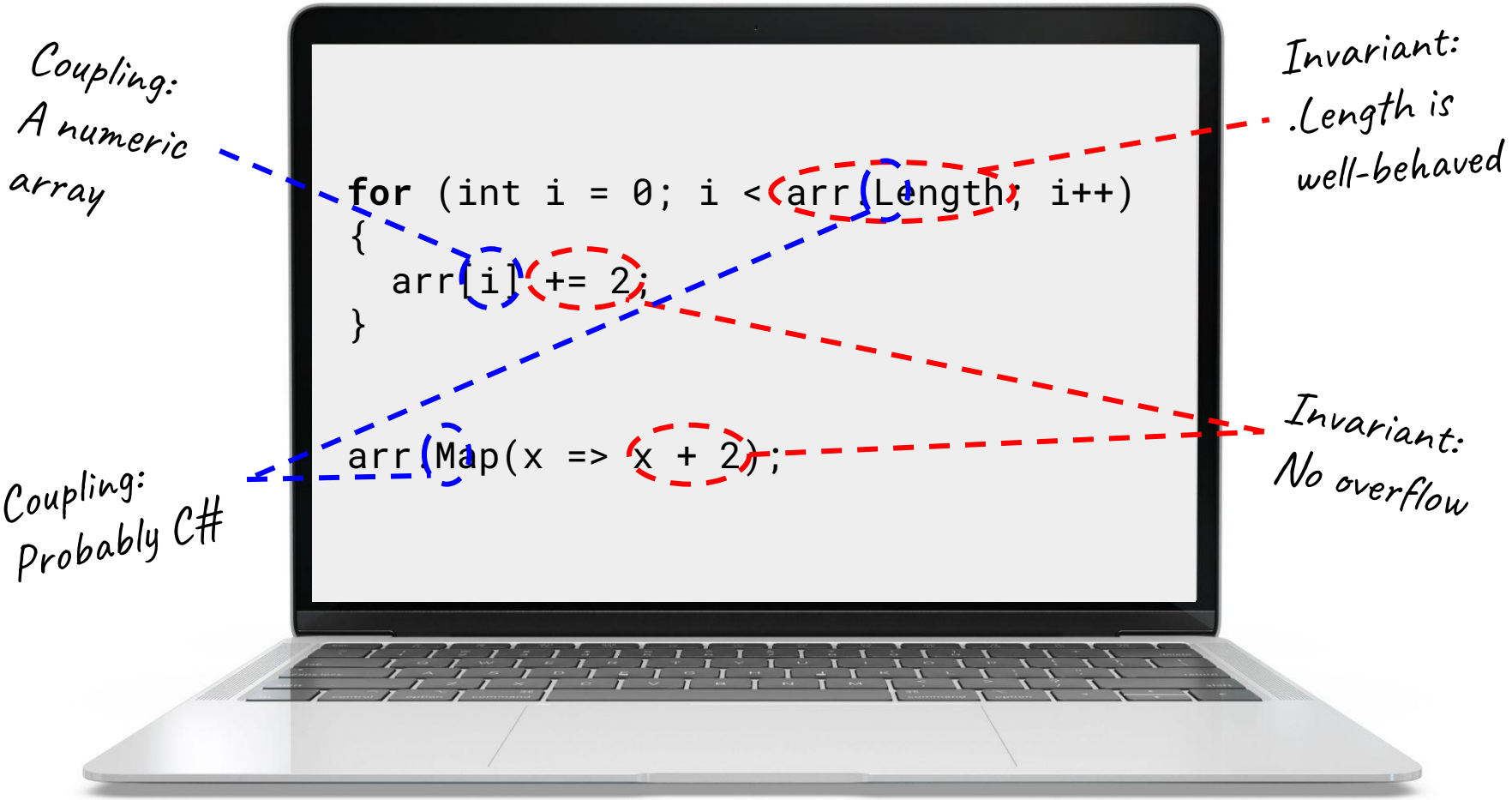
Something that the developer knows but the compiler does not.

Example:
`array.length` cannot be negative.

“Coupling”

Code shared by multiple teams.

Example:
Utility and data classes.


Coupling:
A numeric
array
for (int i = 0; i < arr.Length; i++)
{
arr[i] += 2;
}
arr.Map(x => x + 2);
Coupling:
Probably C#
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

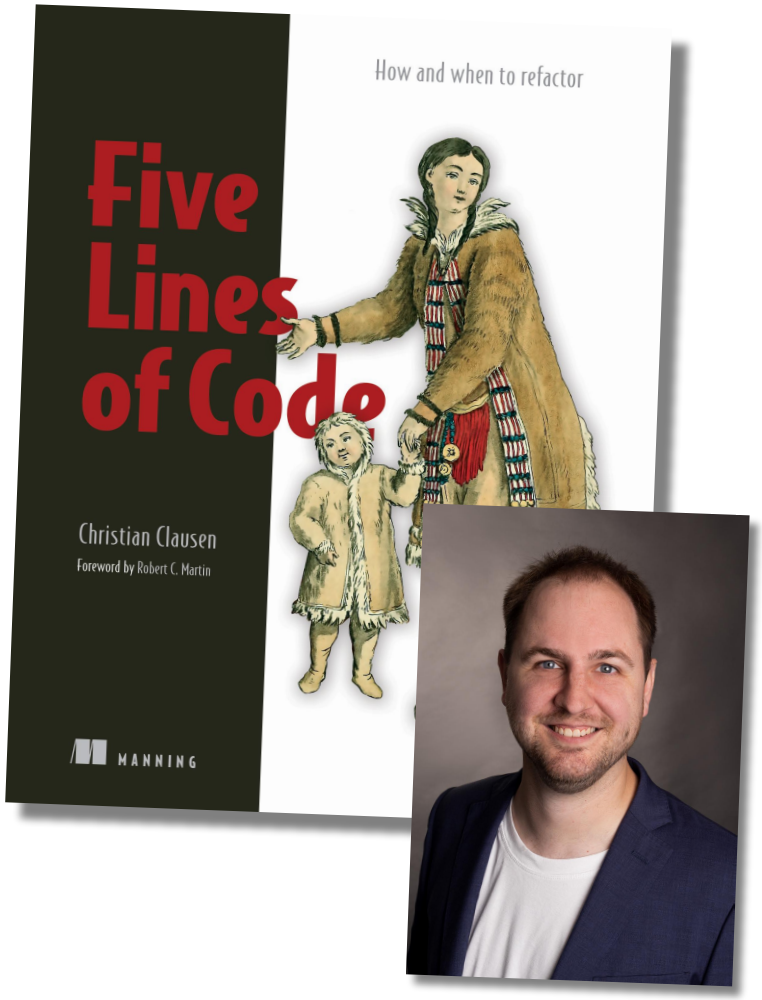
Thank you!

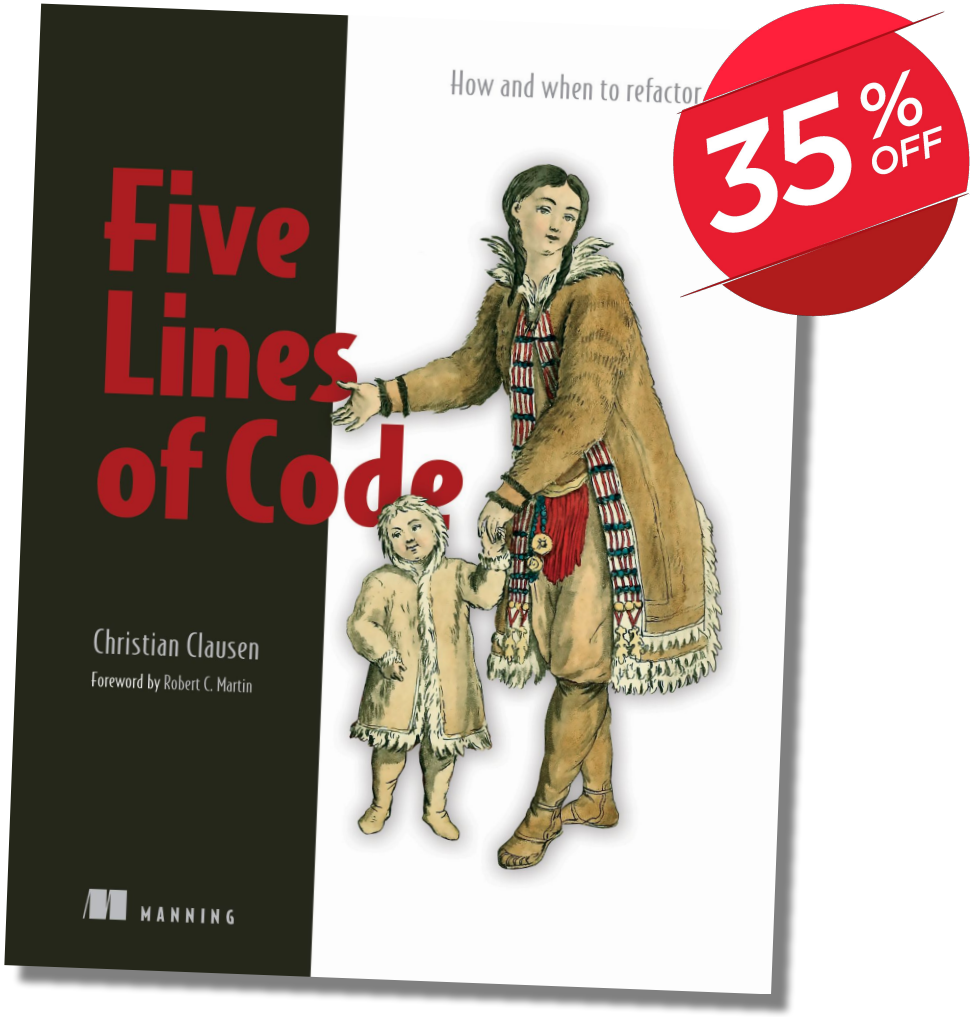
Questions?

Christian Clausen

@theDrLambda

mist-cloud




How and when to refactor
Five Lines of Code
Christian Clausen
Foreword by Robert C. Martin
MANNING
35 % OFF

goto;

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