

Krakow, Vienna, Sofia, Casablanca, Bucharest, Iasi, Belgrade, Kiew, Antwerp



The Art of Clean Code



Clean Code ...

...does one thing well

...reads like **well written prose**

...was written by **someone who cared**

...when each method you read turns out to be
pretty much what you expected

Any fool can write code that a computer understands,

`MOV AX, BX`

but few programmers know how to write
code that a human can understand

**COMMUNICATE,
DON'T CODE**

Bjarne Stroustrup
inventor of C++

Grady Booch
inventor of UML

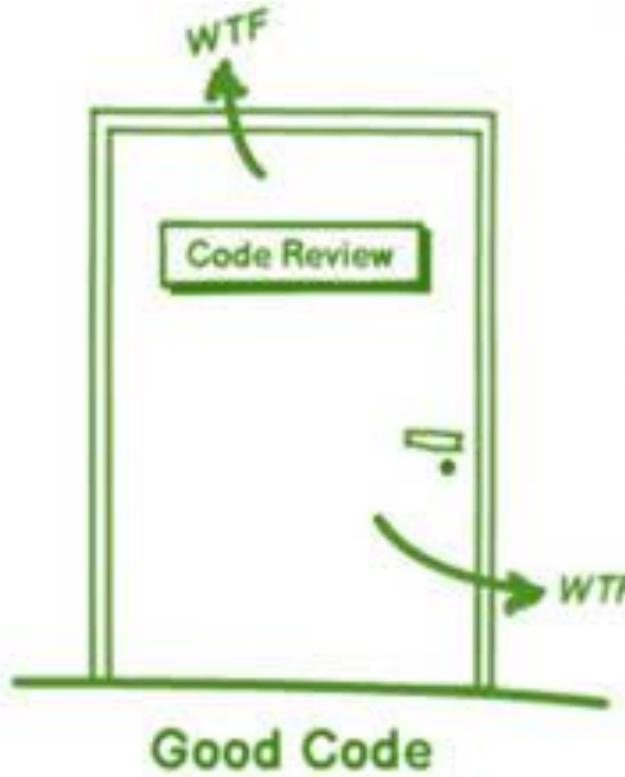
Michael Feathers
Working Effectively with Legacy Code

Ward Cunningham
inventor of Wiki, eXtreme Programming

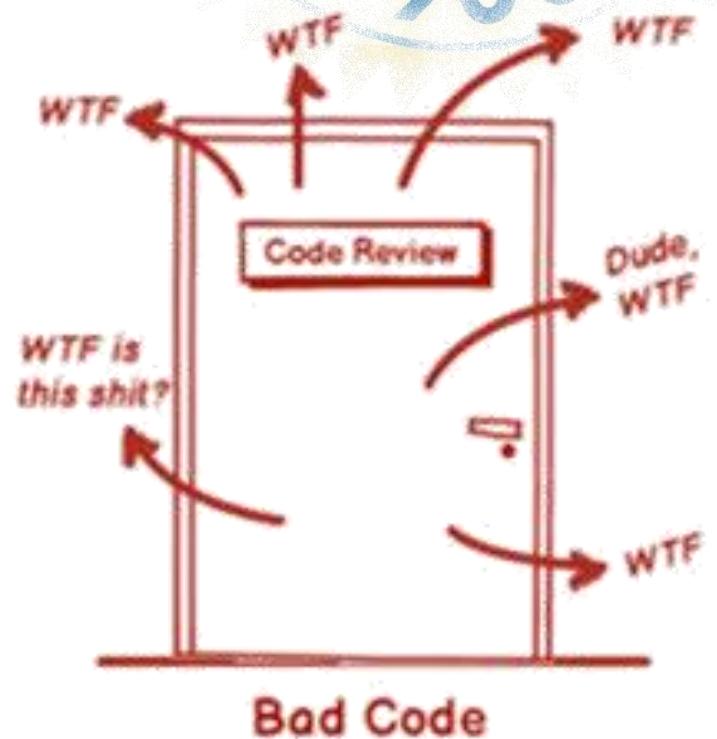
Martin Fowler
author of *Refactoring*

Unit of Measure

The code is pretty much what you expected



code quality meter



Why Clean Code ?

→ **80%** of the total

True cost of software = its maintenance



Real Work Starts

AFTER

Deploy in Production

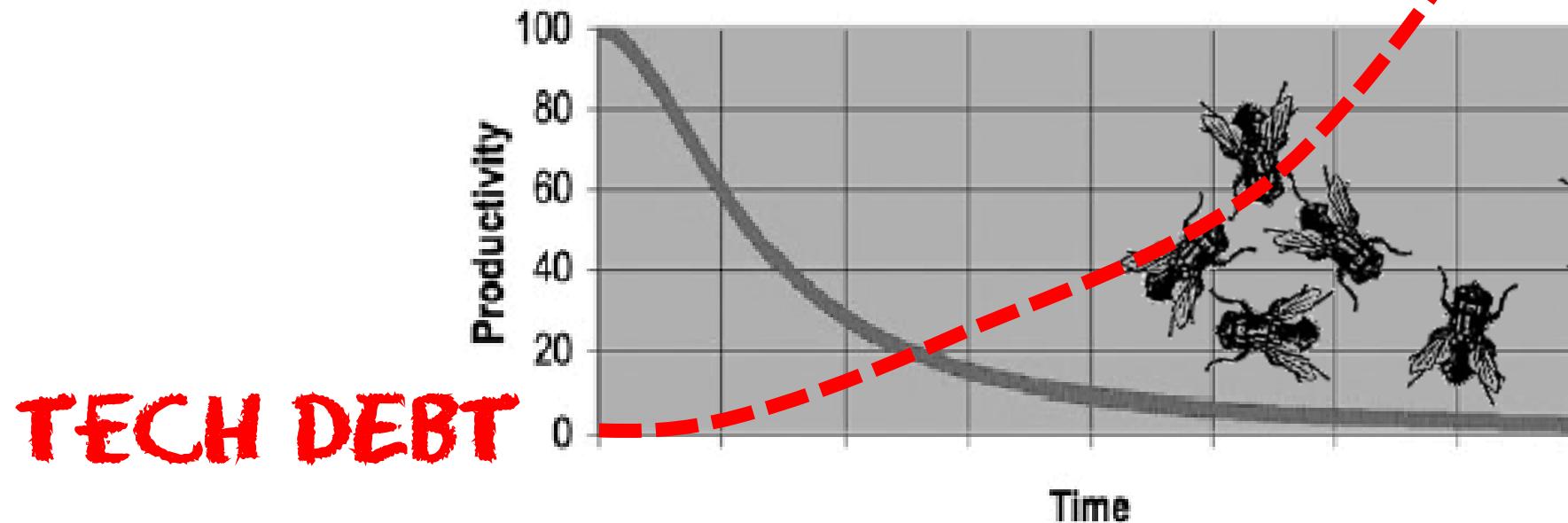
Why Clean Code ?

→ 80% of the total

True cost of software = its maintenance

WHY SO MUCH !?

Cause we get slower, as the code degrades



TECH DEBT

Code smell



code rot

Causes of

TECH DEBT

After the project end:

"we made good decisions, but only now do we understand how we should have built it"
- <https://martinfowler.com/articles/is-quality-worth-cost.html>

Ignorance



Accidental



Deliberate

Throw away or
Planned Refactoring

4



Technical POC
or Prototype
to clarify requirements

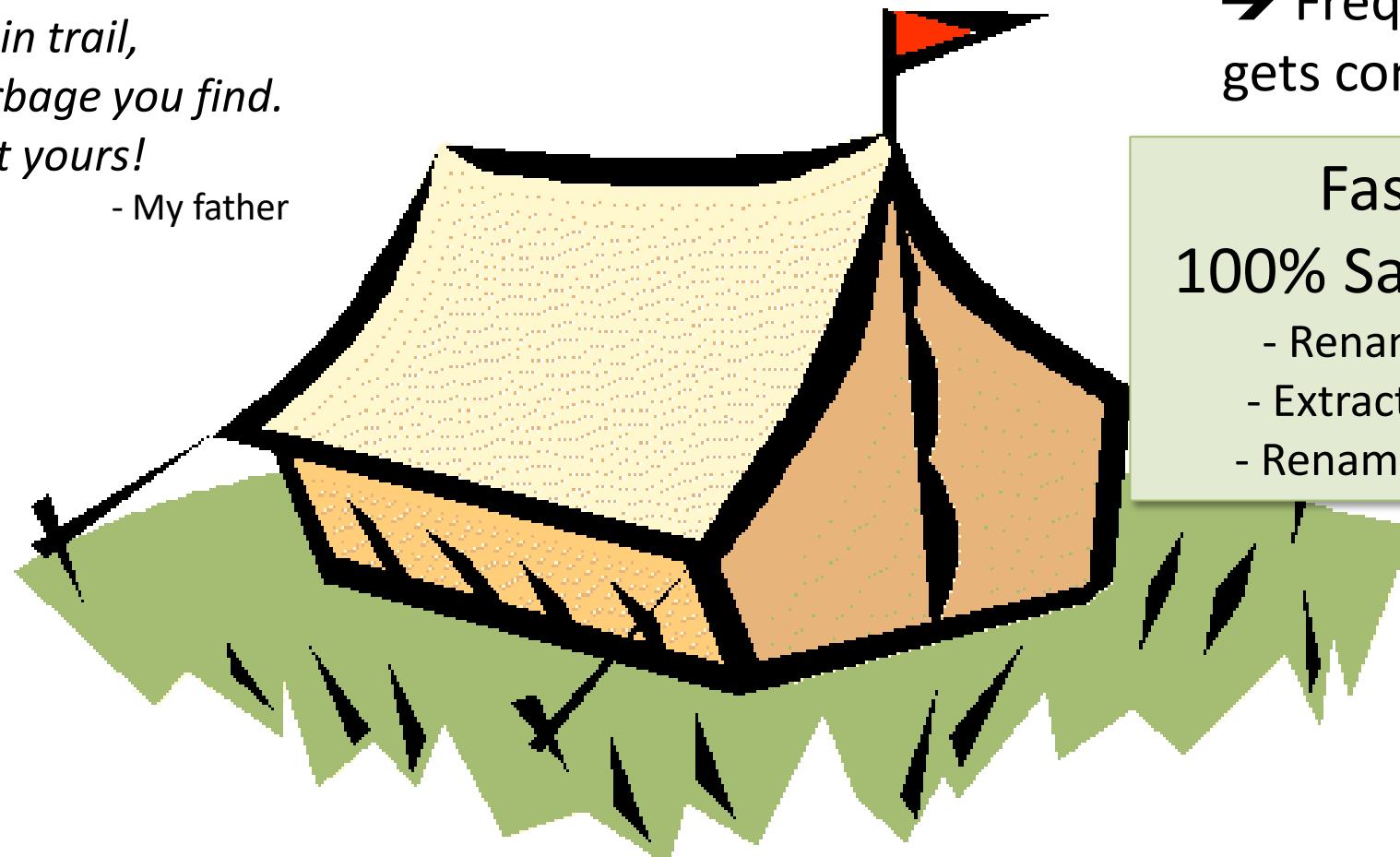
Boy scout rule

Always leave the campground cleaner than you found it

1

*On a mountain trail,
always clean the garbage you find.
Even if it's not yours!*

- My father



→ Frequently Read Code
gets continuously better

Fast, Simple,
100% Safe Refactoring:

- Rename local variable
- Extract private function
- Rename private function

FEAR



What must you do ?

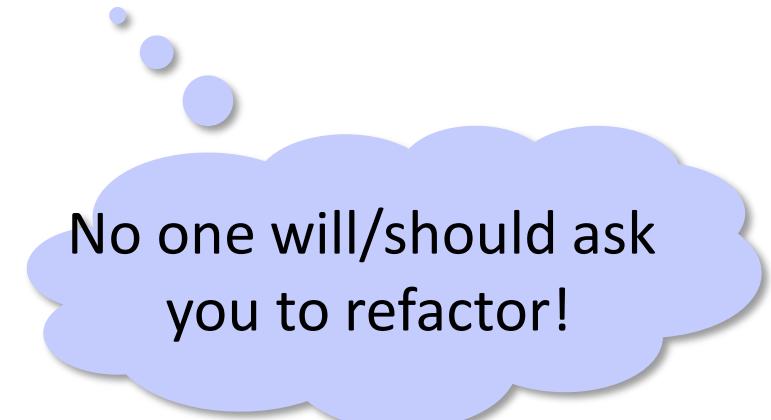
Refactoring

WHAT'S THAT?

Refactoring

Simplify existing code

without changing its external behavior



You have to do a *difficult* change.

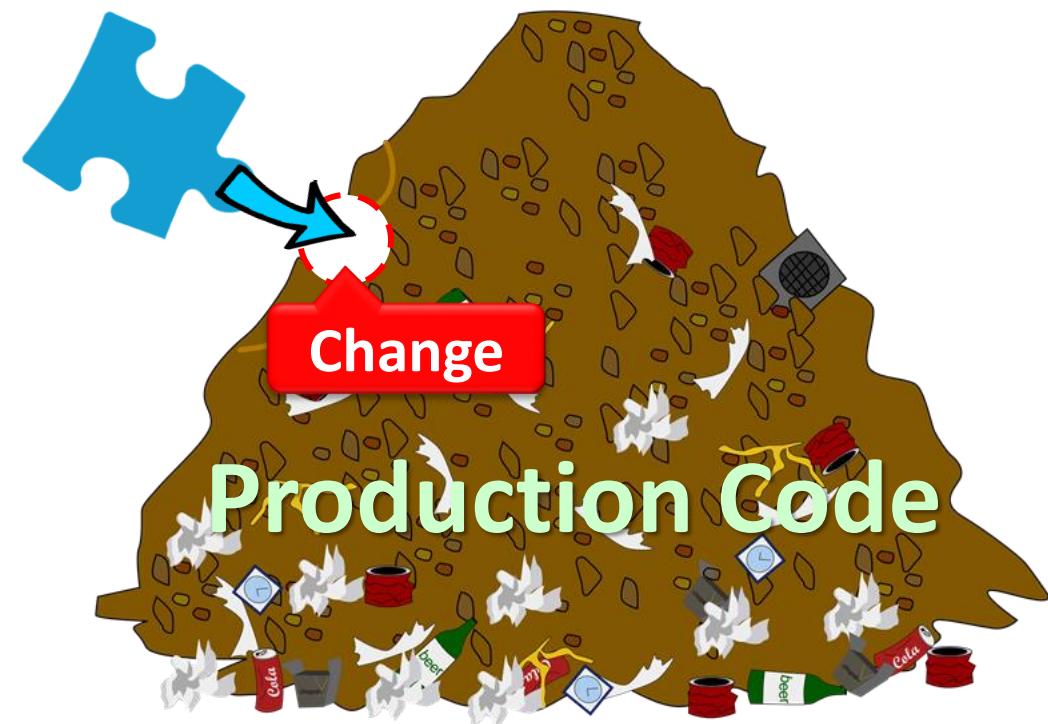
First, you make that change *easy* (*this might be difficult*).

Then, you do the *easy* change.

= preparatory refactoring

2

-- Kent Beck



When Business Thinks:

BRACE YOURSELVES



THE REFACTORING IS COMING

memegenerator.net

Disciplined Refactoring

tiny safe steps

Tarzan-Style Refactoring





Tarzan-Style Refactoring

Mastery: Safe Refactoring



Safe Refactoring Tips

Don't Break Compilation

Further Refactoring Fails

(unless for exploration)

17 errors left... stress

~~Copy > Make old=useless
Cut > Fix compilation~~

Decompose Large Changes

into small, mastered moves

Automated Refactoring

obviously

Alt-Enter (Quickfixes)

keep your focus on the design

Break it into Tiny Steps



Micro-commits

Break a large change set into:



→ Large + Safe Automated Refactoring

→ Small + Manual Refactoring

→ Adding Logic

+ Easier Code Review

+ Less Bugs

- Prior Exploration Needed



Refactoring

= Blockers and Code Smells =

Victor Rentea

Java Champion

♥ Simple Design, Refactoring, Unit Testing ♥

Founder of

Bucharest **Software Craftsmanship** Community

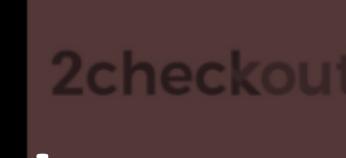
Join us on *Meetup*

Blog, Talks, Goodies on

VictorRentea.ro

Independent Trainer

dedicated for companies / masterclasses for individuals



8 years

2000 devs

40 companies

400 days
(100+ online)

Spring

Hibernate

Functional Prog

Design Patterns
DDD

Clean Code
Refactoring

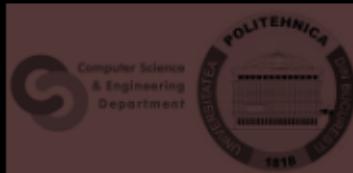
Unit Testing
TDD

any
lang

Reactive

Java Performance

Training for you or your company:
VictorRentea.ro



Posting
Good Stuff on:

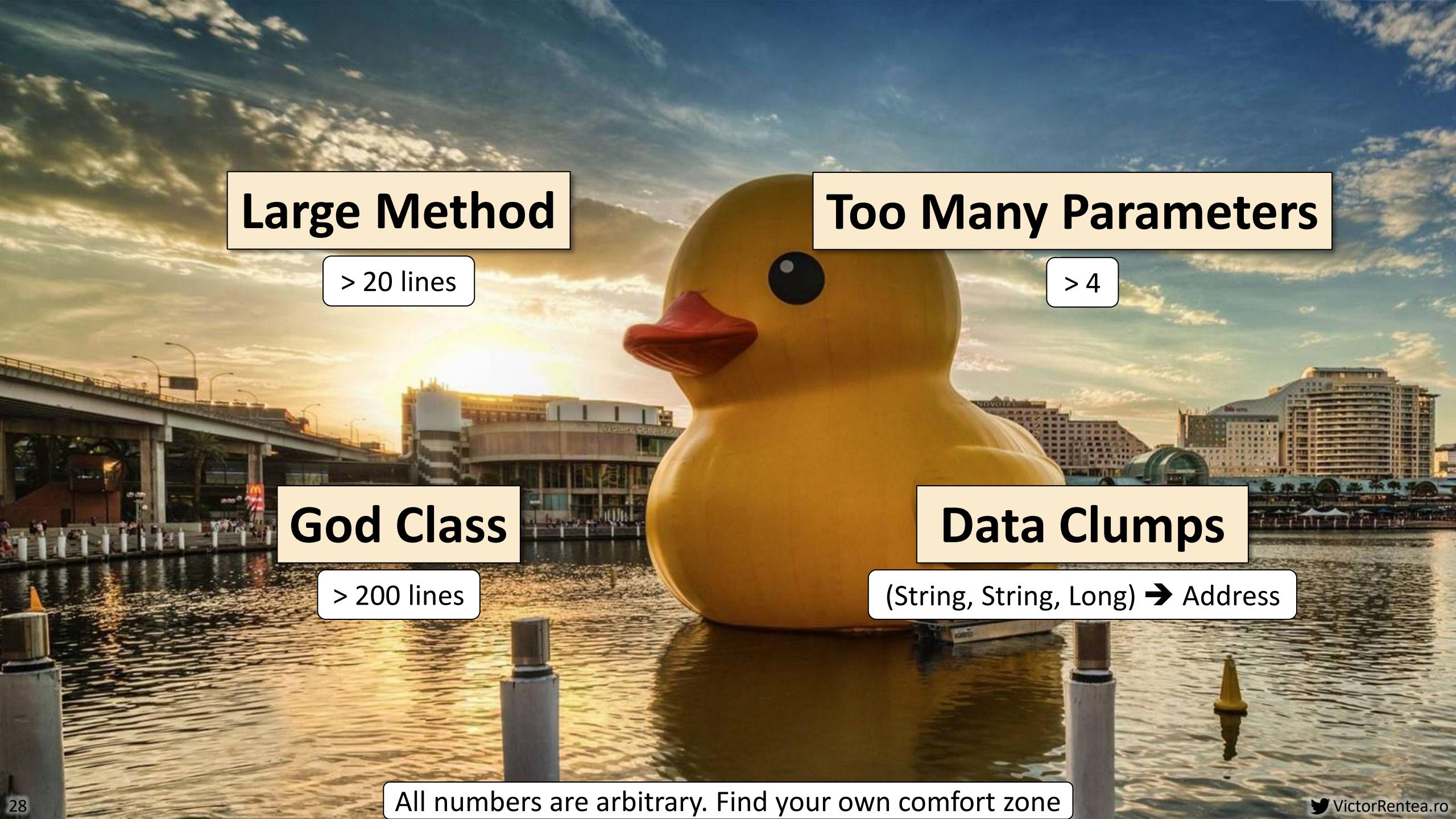
@victorrentea



Code Smells

“If it stinks, change it.”

— Grandma Beck, discussing childrearing philosophy



Large Method

> 20 lines

Too Many Parameters

> 4

God Class

> 200 lines

Data Clumps

(String, String, Long) → Address

All numbers are arbitrary. Find your own comfort zone

Data Classes

get/set mania

Feature Envy

I ❤️ your state. Can I move in?

Primitive Obsession

→ PhoneNumber, OrderId, enum

Speculative Generality

→ ✎ KISS Principle

Middle Man

person.getAge()

~~int getAge() {
 return bio.getAge();
}~~

person.getBio().getAge()



Primitive Obsession

```
redeemCoupon(Long, Long, String)
```

```
Map<Long, List<Long>> customerIdToOrderIds
```

Micro-Types

PRO
= an early design decision

```
redeemCoupon(Long, Long, String)
```

```
redeemCoupon(CustomerId, CouponId, PhoneNumber)
```

```
Map<Long, List<Long>> customerIdToOrderIds
```

```
Map<CustomerId, List<OrderId>> orders
```



Escape the

Primitive Obsession

PRO

Make concepts explicit
by introducing new small classes

Even if it's a single

`String → PhoneNumber`

`Long → CustomerId → ID type:`

```
@Value
class CustomerId {
    Long id;
}
```

Code Smell: Iterating Map Keys

```
List<CustomerOrderIds>
```

```
Map<Long, List<Long>> customerIdToOrderIds
```

If the only usage of a map is:

```
for (Long customerId : map.keySet()) {  
    List<Long> orderIds = map.get(customerId);  
    ...  
}
```

You might be missing an abstraction (class):

```
class CustomerOrderIds {  
    private Long customerId;  
    private List<Long> orderId;  
    ...  
}
```

Deploy to Prod in

0:10



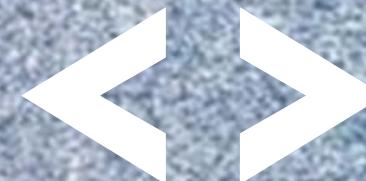
Is it a bug or a feature?



Duplicated Code

Is terrible when changes

Shotgun Surgery



Divergent Changes

SRP



repeated switches

vs Polymorphism

vs Logic in enum

vs "functional" switch (java17)

CP

```
for (e : list) {  
    A(); list.stream()...A...  
    B(); list.stream()...B...  
}
```

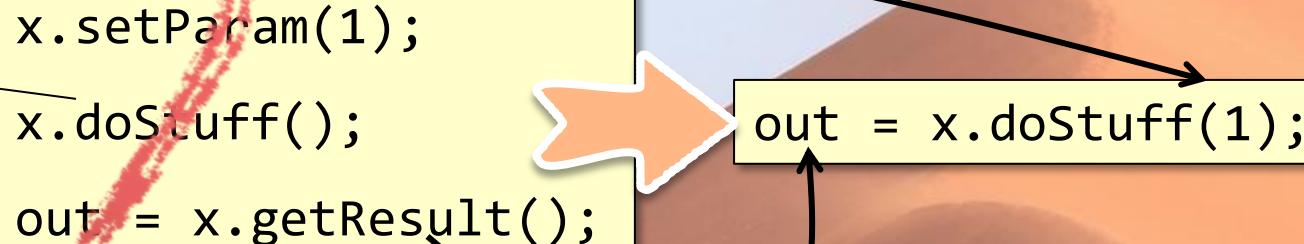
loops

reads from param field
and writes to result field

Temporary Field

```
x.setParam(1);  
x.doStuff();  
out = x.getResult();
```

TEMPORAL COUPLING



Long-Lived Mutable Data



+ MULTI-THREADING =





Code Smells Sheet

Defeating the Evil starts with Naming It

Code Smells Sheet

Long Method

God Class

Data Clumps

Long Parameter List

Primitive Obsession

Data Class

Feature Envy

Middle Man

Defeating the Evil starts with Naming It

Duplicated Code

Shotgun Surgery

Divergent Code

Repeated Switches

Loops

Temporary Field

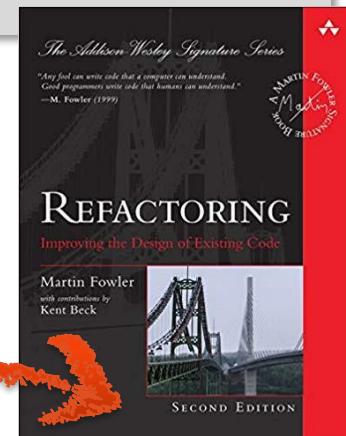
Long-Lived Mutable Data

Speculative Generality

Comments

+ many more in

CHAPTER 3



What must you do ?

Refactoring

WHAT'S THAT?

Refactoring

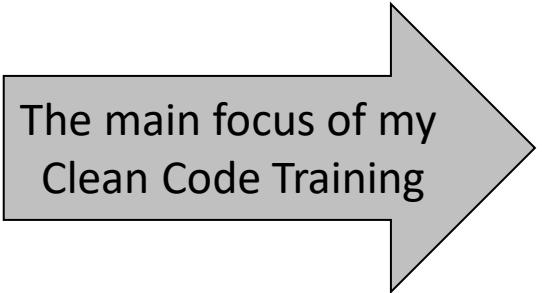
Simplify existing code

without changing its external behavior



Disciplined Refactoring

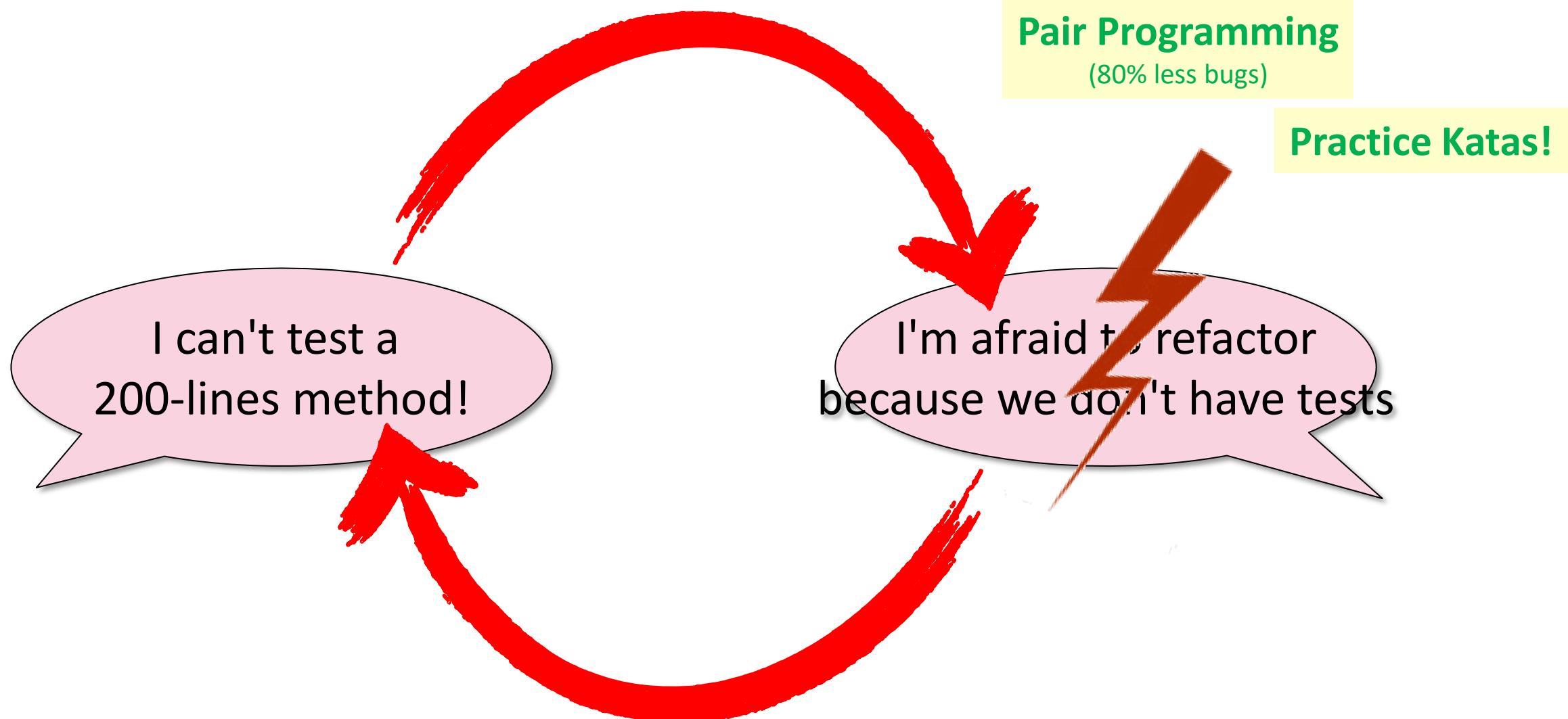
tiny safe steps



The main focus of my
Clean Code Training

FEAR
What Spills?

The Vicious Circle of Legacy Code



Time



"I don't have time. I'm too busy"

Being busy is a form of laziness

- Tim Ferriss

DO less

Automate. Reflect.



Habbits of Efficient Developers

Improve Focus

7 habits of highly efficient people

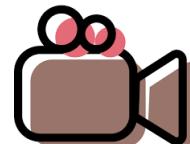
pomodoro

no morning inbox

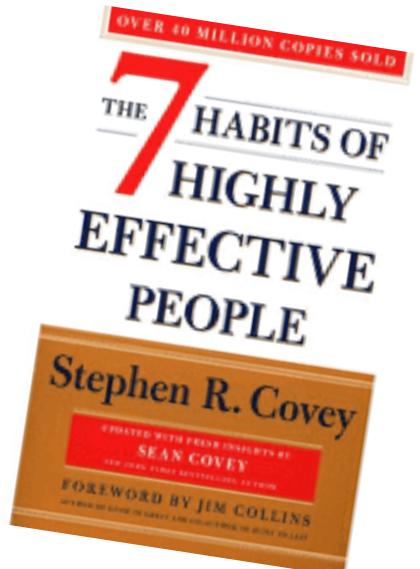
<30 min meetings

disable desktop/mobile notifications

digital detox



Get your brain to focus



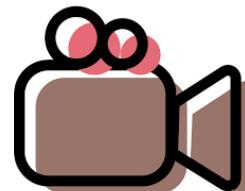
After you finish a task,

Spend 30 minutes trying to improve your design

... then 1 hour

... then 2

...



[The Well-Balanced Programmer](#)

by @jbrains

Estimates







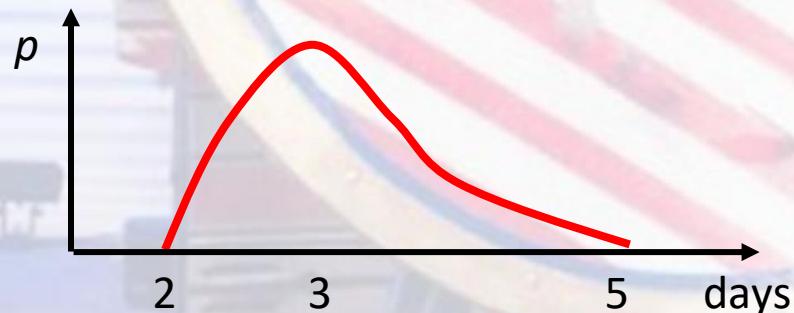
Requirements Changing all the Time

Estimates

Looking at code + an old dev

Proper Estimates, including cleanup + unit-testing

Measure It!



Estimate = {min=2, expected=3, worst=5}

if they take min=2 => give them 4

* shit_factor

- per code area
- or estimate in FP

Expose Risk

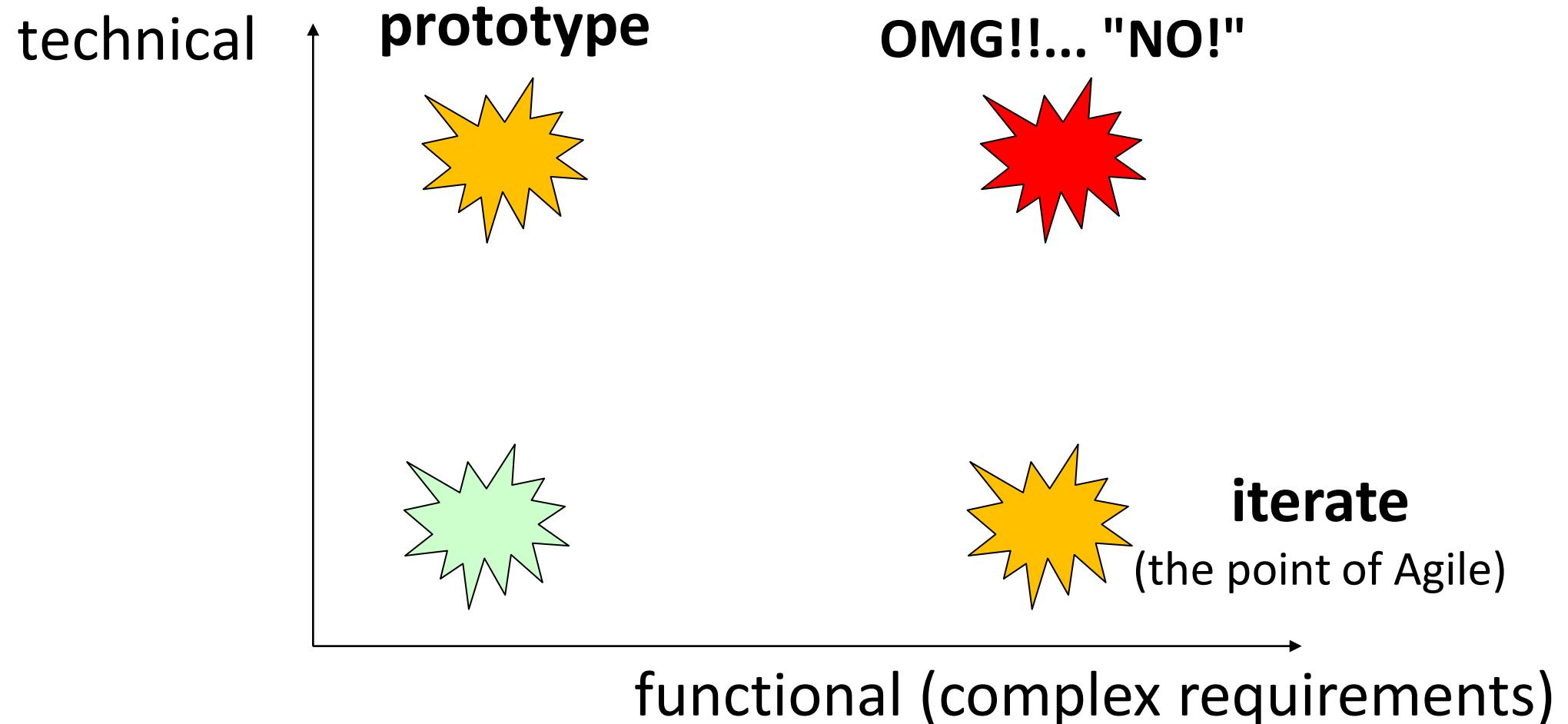
Say "NO"

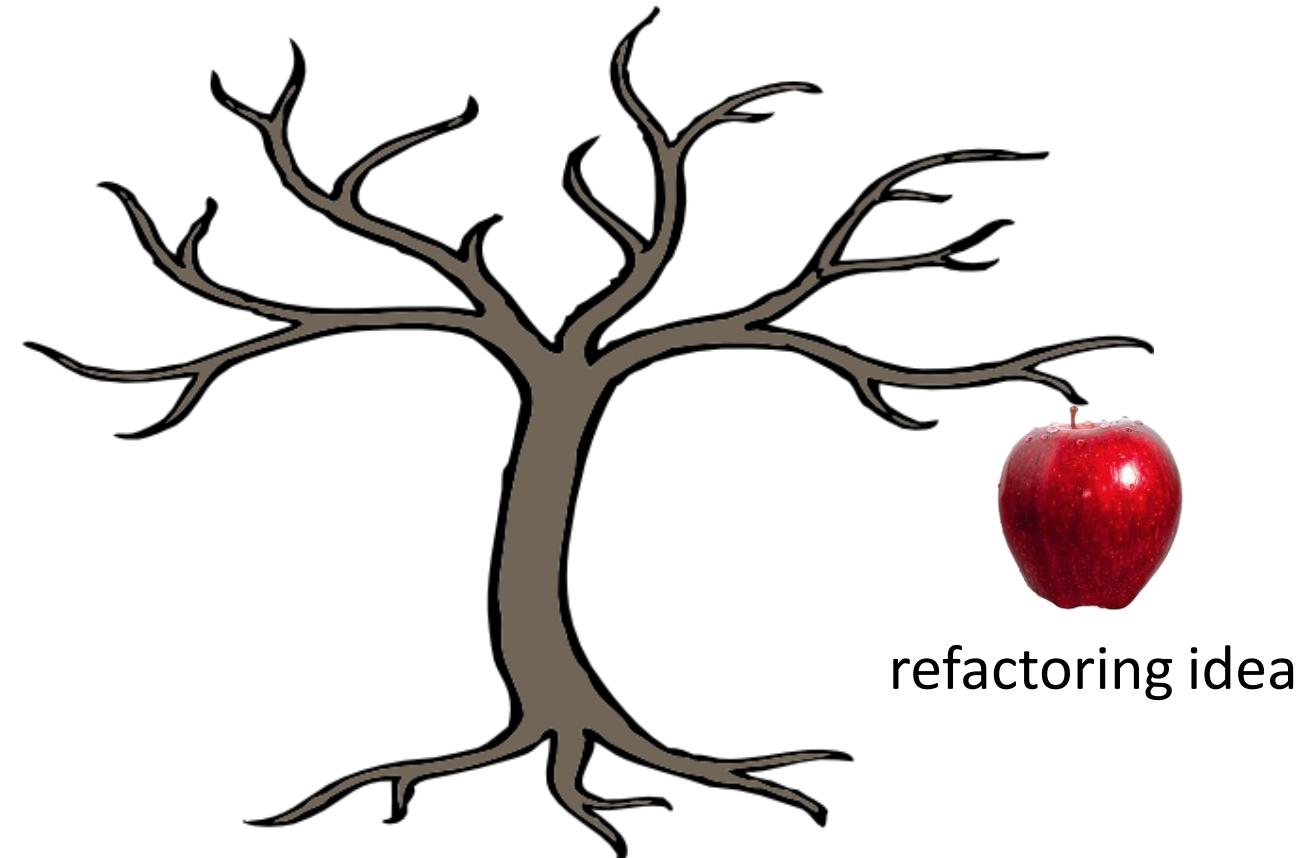
How DARE YOU?!
suggest I'm not trying!!

Could you at
least TRY?



Estimations Risks





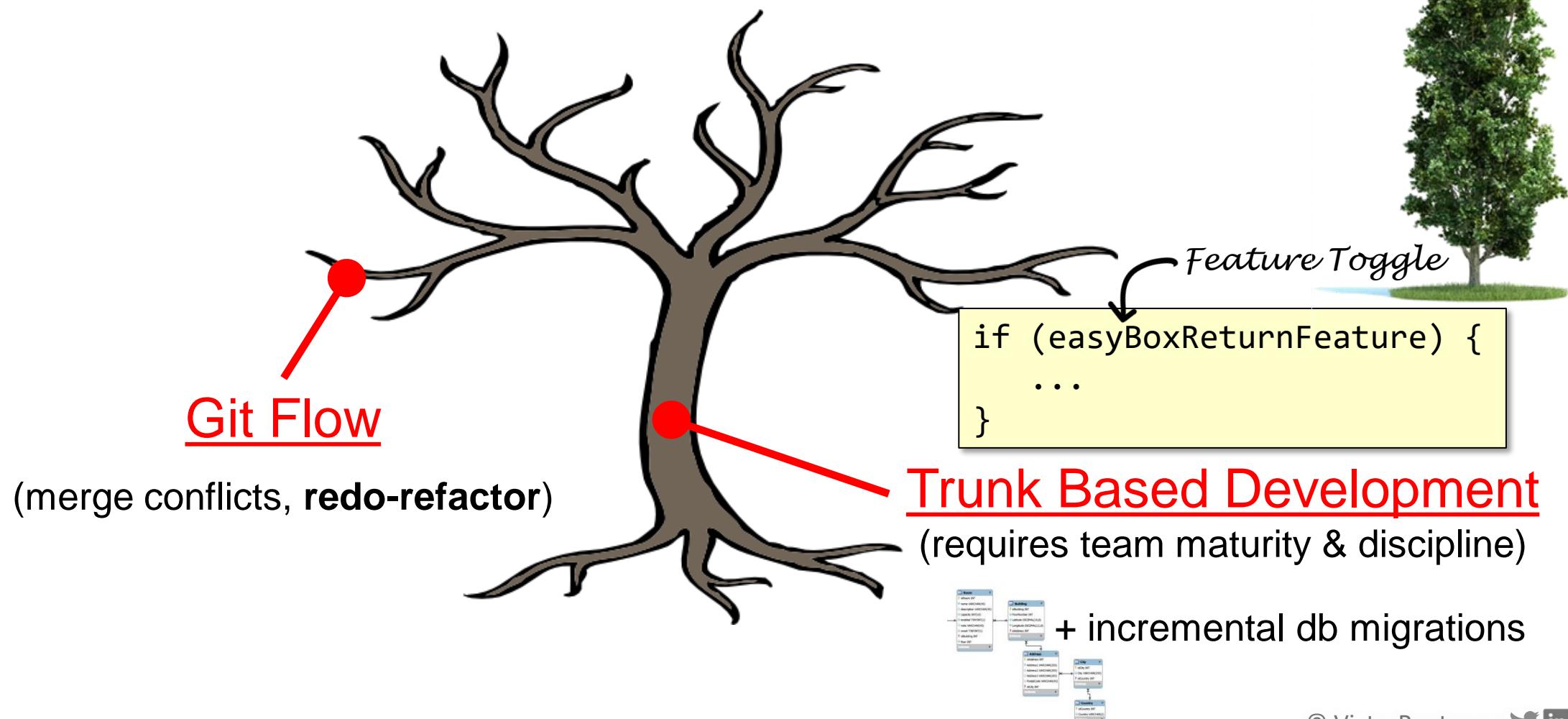
Other annoying things in Dev's life:

- Interruptions
- Refactoring Tests
- Long meetings

Merge Conflicts



Long Branches Inhibit Refactoring



Trunk Based Must Have

- Experienced / bright developers
- Pair Programming 100%
- TDD
- Incremental DB scripts + Undo scripts
- [opt] In parallel with other feature branches

What Stops Refactoring?

Deadlines

→ Proper estimates:
(min, max), shit_factor

Fear of Bugs

→ tests > pair programming > practice

Lack of Skills

extract a method ?!!
→ Learn Code Smells

Mindset of Rushing

although your Lead tells you not to

Merge Conflicts

→ short-lived branches/trunk-based

Explicitly Forbidden!

change guard = angry bosses
= distrust
Seek redemption!

Fear of Breaking your Eggs

to make the omelet

Lack of long-term Vision

→ regular team design brainstorm

Bad Cost/Benefit Ratio

after serious consideration

<https://www.hyrumslaw.com/>

Unknown Code

Refactoring is a learning activity:
from spec/tests/code

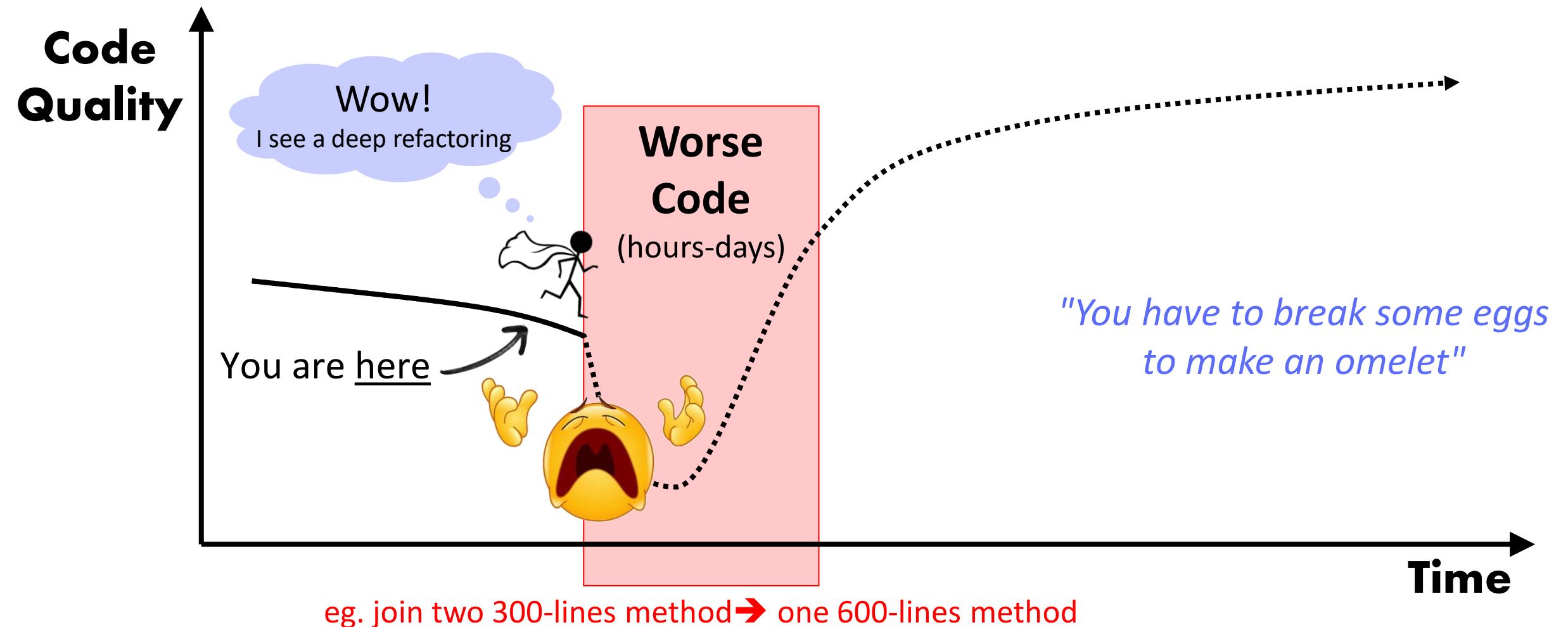
Will

YOLO Driven dev

- A. Make it fun, play, exercise
- B. Legacy Experience ™

Initial Chaos

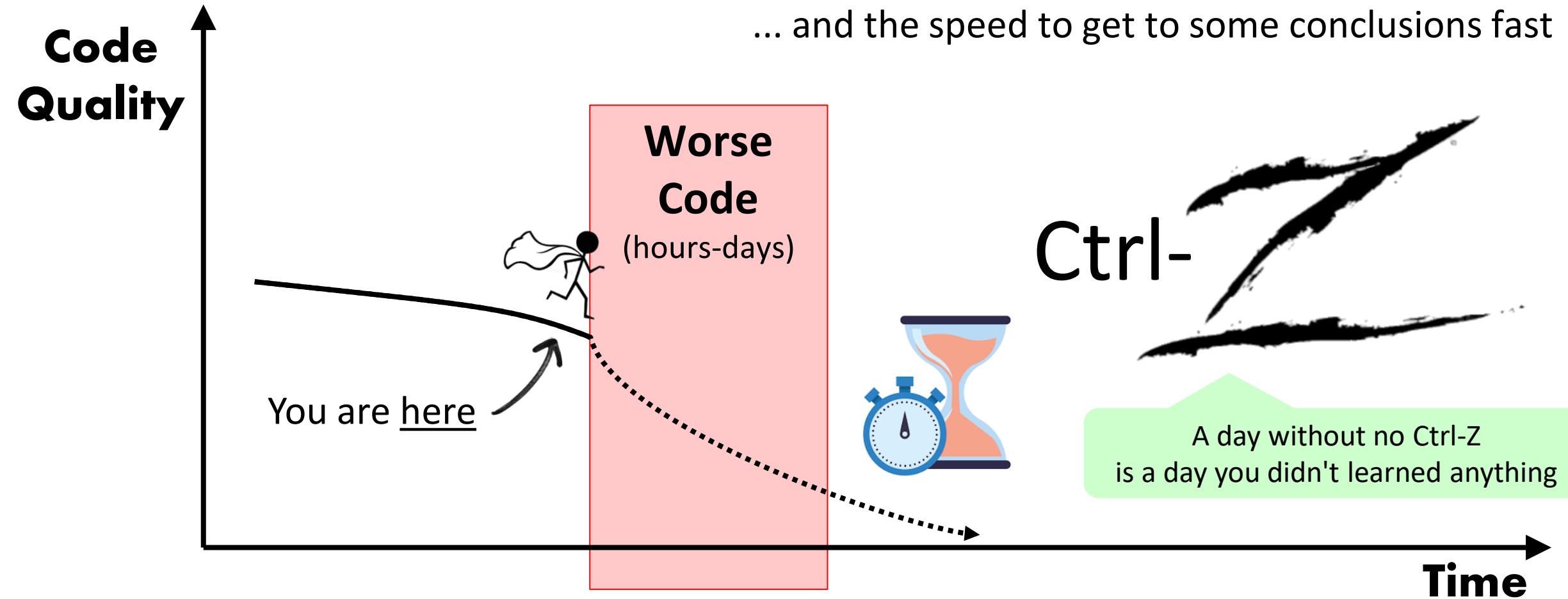
Deeper Refactoring starts with making a mess



You need

The Strength to Revert

(despite emotional involvement)

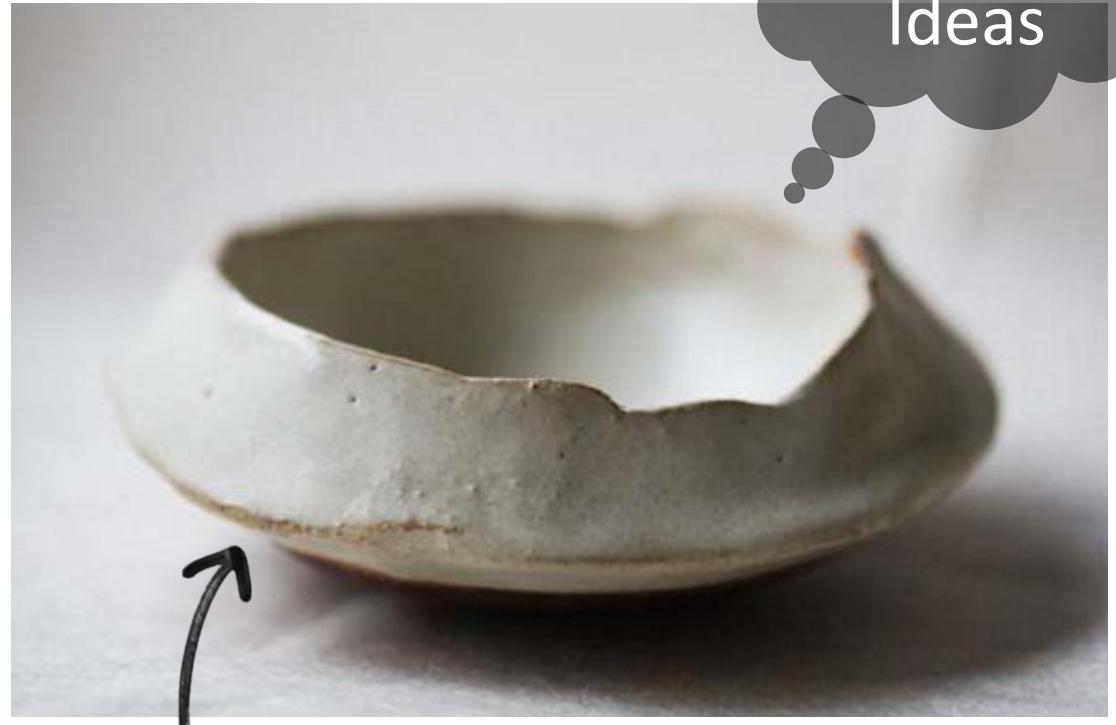


Refactoring is a Learning Activity

Which do you know best:



Admired at a museum



Modeled by your own hands

More
Refactoring
Ideas

A wide-angle photograph of a waterfall in a dense tropical forest. The waterfall consists of several distinct tiers, each cascading down dark, mossy rock ledges into pools of clear, turquoise water. Lush green foliage, including various trees and ferns, surrounds the waterfall, creating a sense of a hidden, natural paradise.

Falling From
One Refactoring
To Another

until...

Getting Out of Refactoring

Code refactoring

Getting Out of Refactoring

3 hours later, the commit message:

code improvements

refactoring

I gotta leave

Major changes

Challenge



Write Down Your Next Commit Message

On a piece of paper.

With your hand.



Identify the Steps

What's Missing from our Practice:

Tentative Refactoring

Ctrl -

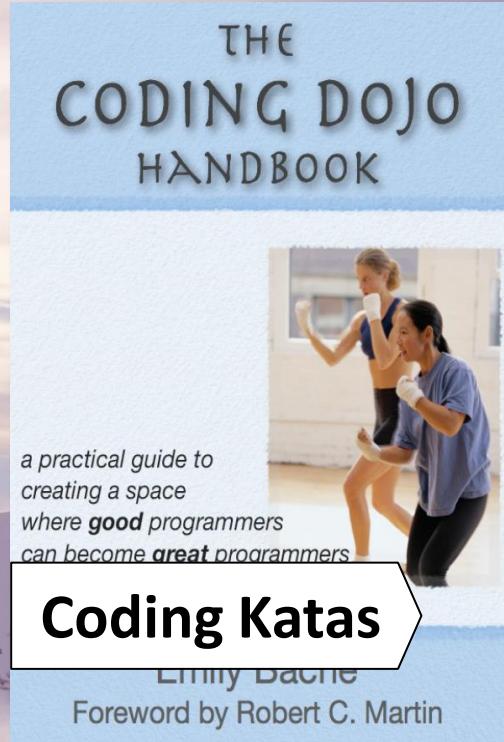
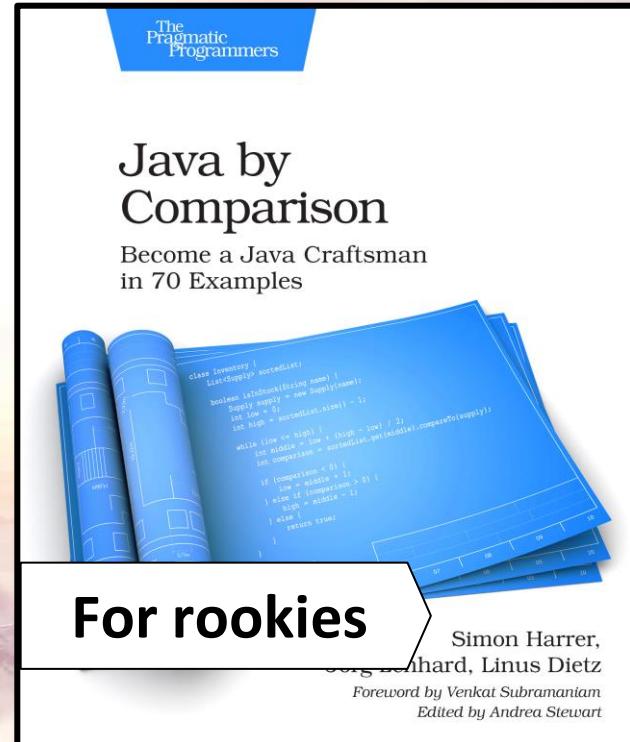
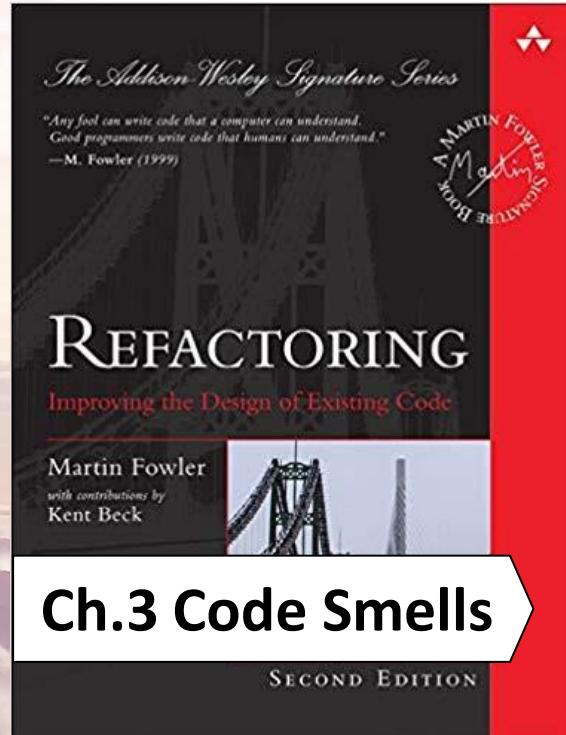
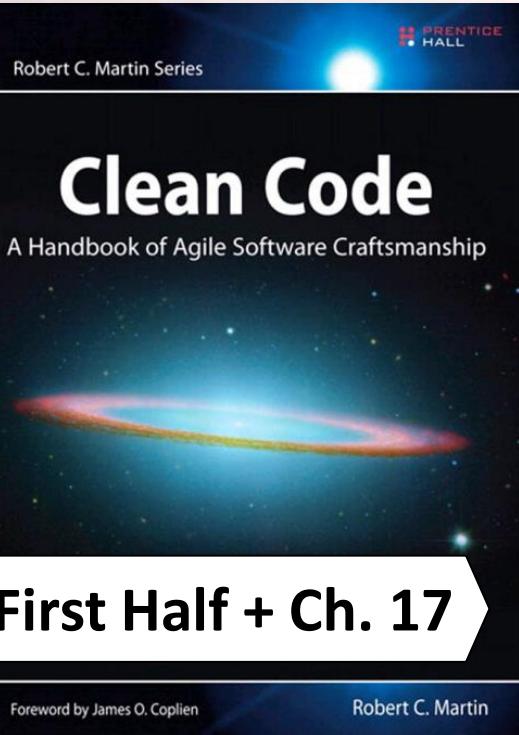
Explore. Play.

15 min.

Revert.

Only the Brave admit they were wrong

The End ?



Join me:

victorrentea.ro/community

blog, best talks, training offer

VictorRentea.ro

Share your thoughts



@VictorRentea

end

Performance

often

≠ readable

different goals

Bottlenecks **will** happen,
in < 5% of the code



Two strategies

1) Optimized code everywhere

-> unreadable code -> impossible to find that 5%

2) Readable code

-> easy to find those 5% -> optimize

Pro Tip:

Piggyback performance improvements tasks
to clean & refactor

keep your code clean today
so that you can find
the performance bottleneck tomorrow

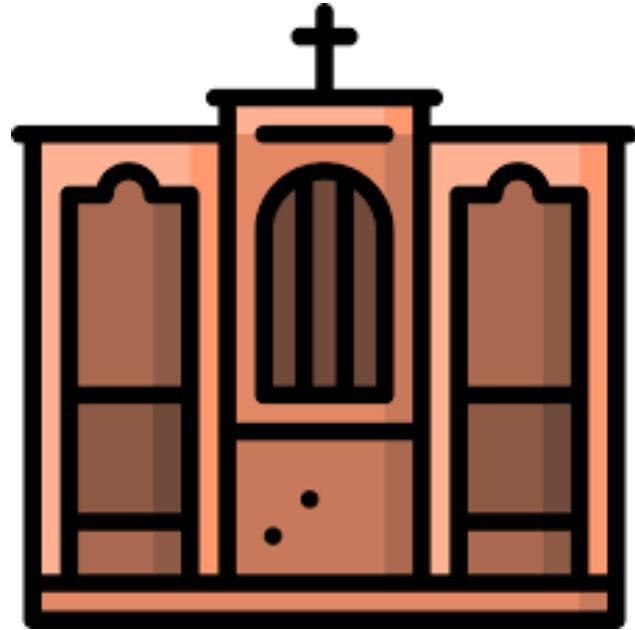
“ We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil** ”

— Donald Knuth

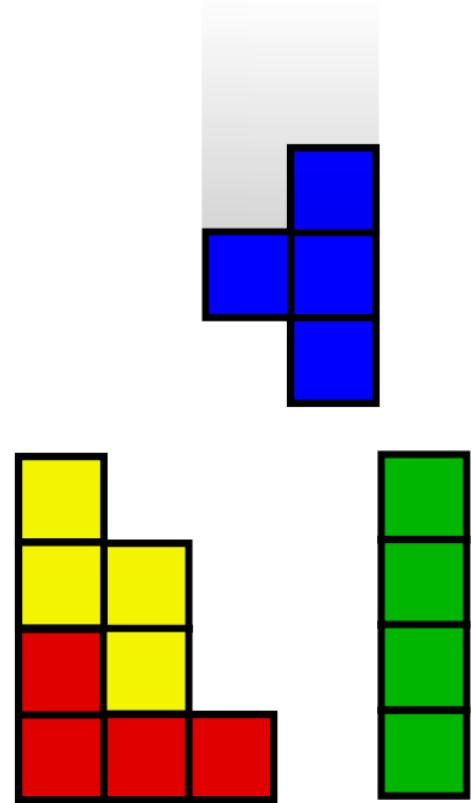


but don't do stupid things of course:
eg careful with remote calls

TOD



**THE Refactoring
Sprint**
(on code written by you)



Coding



Unit Testing

Never Separate Them



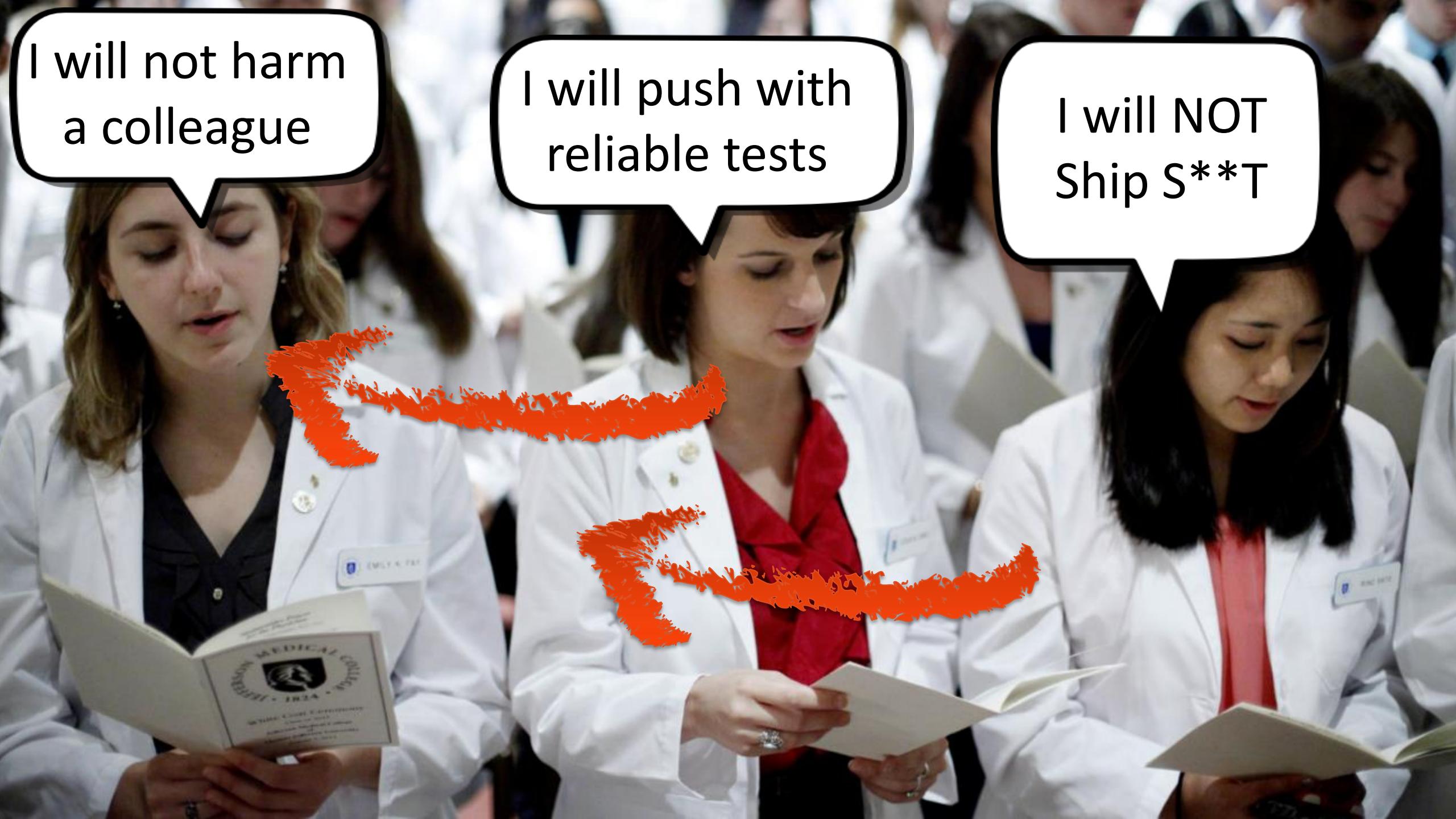
Never Separate Them

US-13

Develop

Me showing up
to sprint planning
after finishing nothing
last sprint ...





I will not harm
a colleague

I will push with
reliable tests

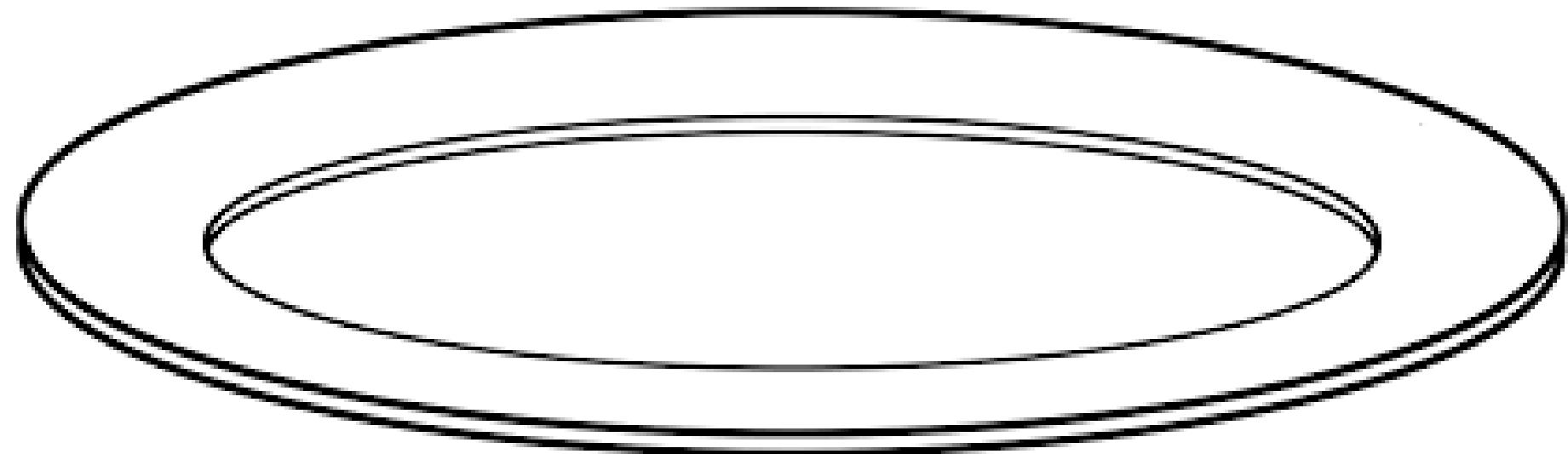
I will NOT
Ship S**T

Dear Manager,

*You hired me because I am the only person on Earth
who has the competence to tell "**NO**" to your requirements.*

- Uncle Bob
(approximate quote)

<https://youtu.be/LmRI0D-RkPU?t=3435>



Refactoring Improves Dev's Quality of Life

WHY?

Nobody gets the design right the first time

Refactoring Habit eliminates the guilt of failing

You'll feel less guilty than you felt yesterday

knowing that you will clean it anyway tomorrow

What is needed to make
Refactoring happen
Every Single Day in your team?

DO NOT CROSS

deadline noun

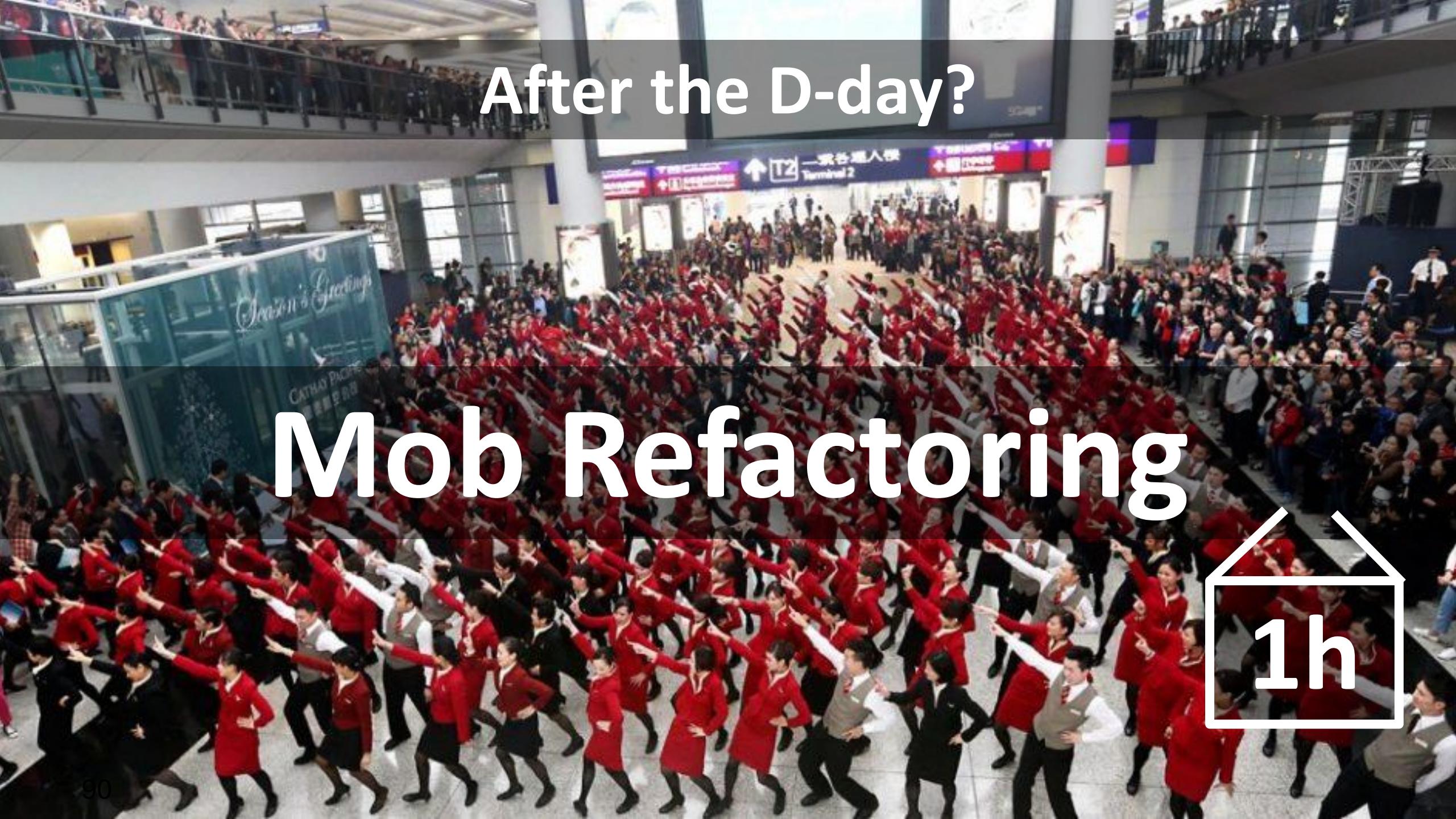
dead·line | \ 'ded-,līn \ 

Definition of *deadline*

- 1 : a line drawn within or around a prison that a prisoner passes at the risk of being shot

What do you do
After the D-day?

After the D-day?

A wide-angle photograph of a large crowd of people in red uniforms, likely airline staff, dancing in a public space. They are performing a synchronized routine, with many people raising their arms. The setting appears to be a modern airport terminal, with a large digital sign above reading "After the D-day?" and "Mob Refactoring".

After the D-day?

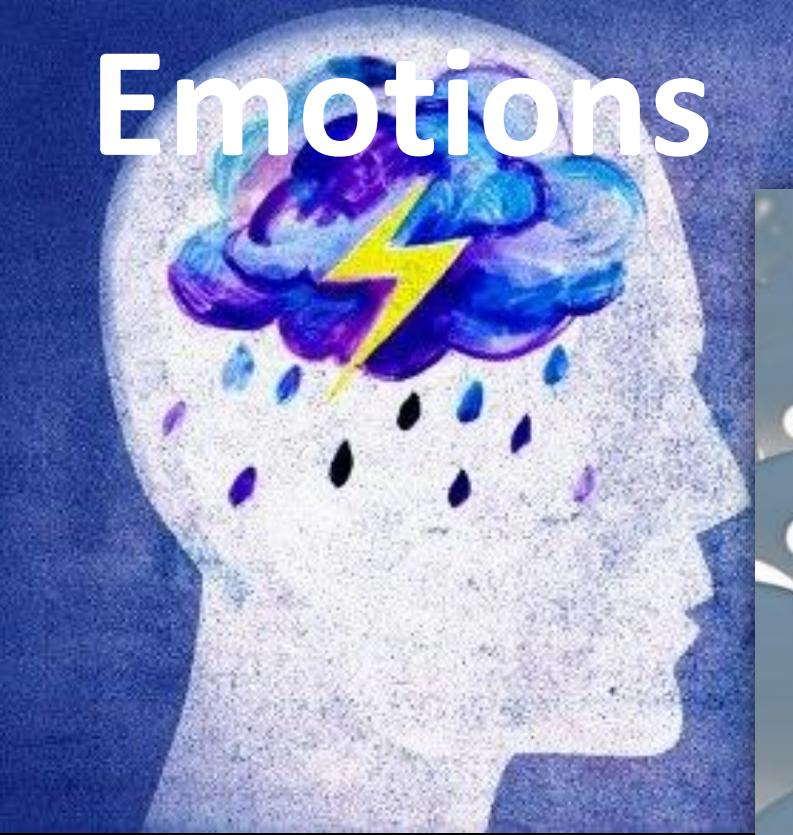
Mob Refactoring

A white house-shaped icon with the number "1h" inside, indicating a duration of one hour.

1h

Clean Code

is about



Emotions



Attitude



Team



What Will You Choose?



My trigger was cleancoders.com

Agenda

Introduction

Names **The power that YOU have**

Functions

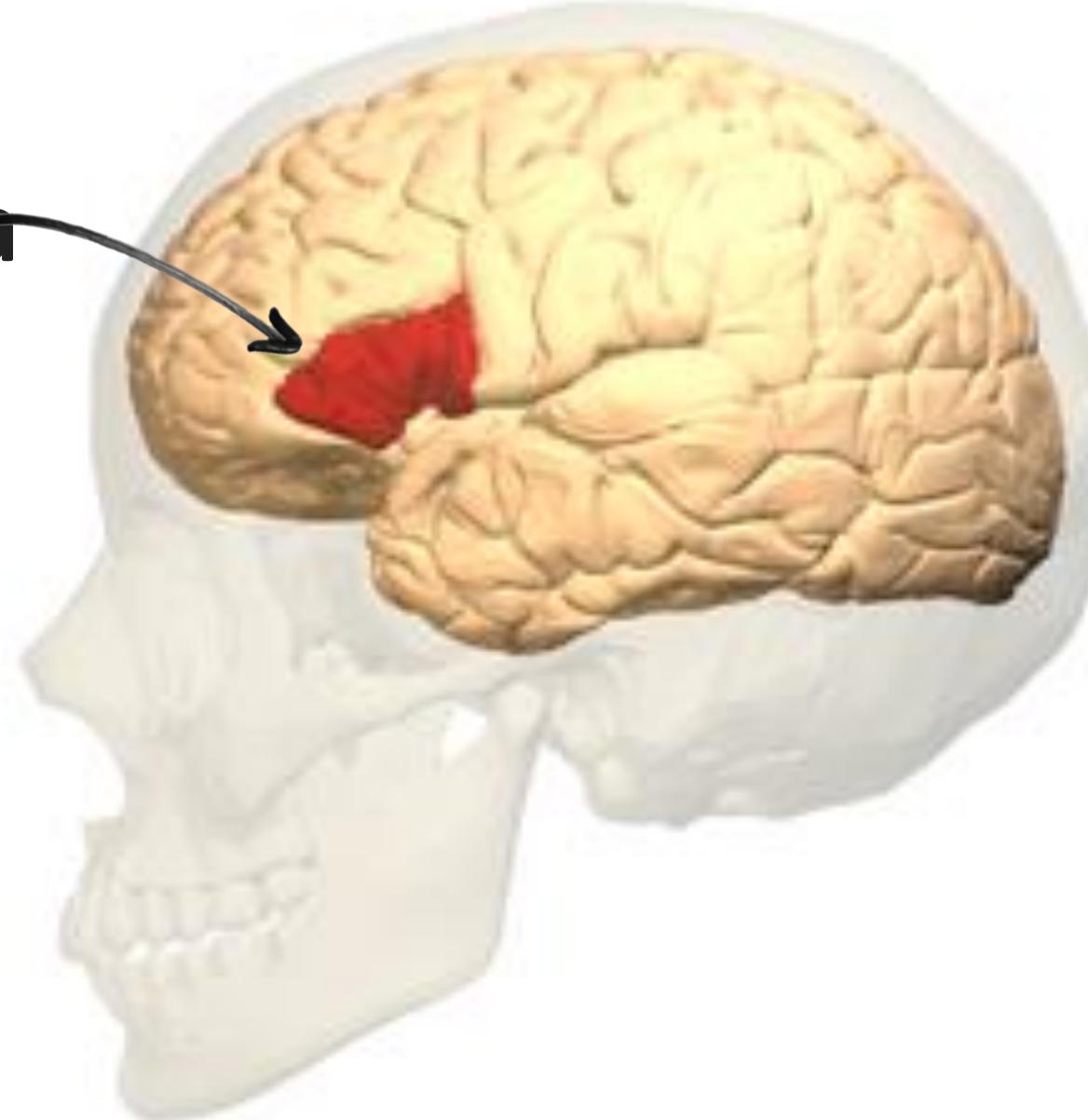
Classes

Formatting & Comments

**With Great Power
Comes Great Responsibility**

- *SpiderMan[®]*

Broca's Area



Classes are nouns

Customer, OrderDetails, OrderFacade

Seek 3-5 options
for any name

Avoid meaningless names

Order~~Info~~, Order~~Data~~ vs Order

OrderDetails OrderSummary

refusing
your Power



Delete the Interfaces

~~CustomerService, OrderService~~Impl

Delete the Interfaces

except:

... with ONE implementation
in the same module

Remoting/API

your-app-api.jar

MapRepo **implements** IRepo

BufferedWriter, FileWriter

Test Impl (?) or Strategy® Pattern or Decorator®

Select implementation dynamically at runtime

Dependency Inversion

Implemented in lower-level modules



Have you ever blamed some code?



... then find out

YOU were the author?



You Forget



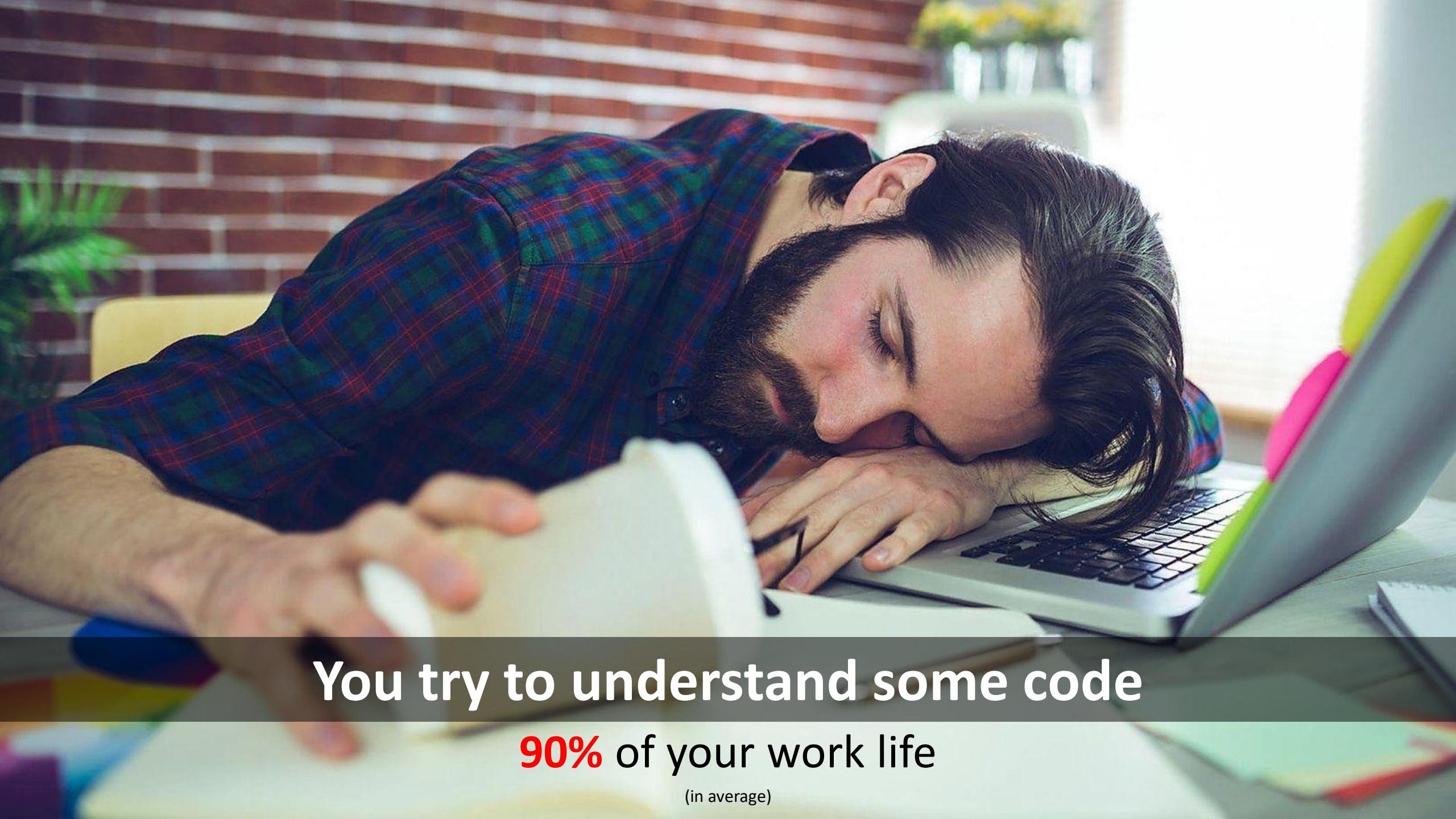
Continuous Renaming

(as you learn the application)

The team will be grateful!
 Yourself will be grateful, in 3 months!



Continuous Renaming



You try to understand some code

90% of your work life

(in average)

You try to understand some code

2 hours

You READ code 10x more time than you WRITE

Then, you get it



You change only one line

1 minute

How do you **feel?**

A man in a dark suit and white shirt is shouting at a laptop screen. The screen is shattered, with many shards of glass and plastic flying outwards. The word "FRUSTRATION" is written in large red letters across the top of the image.

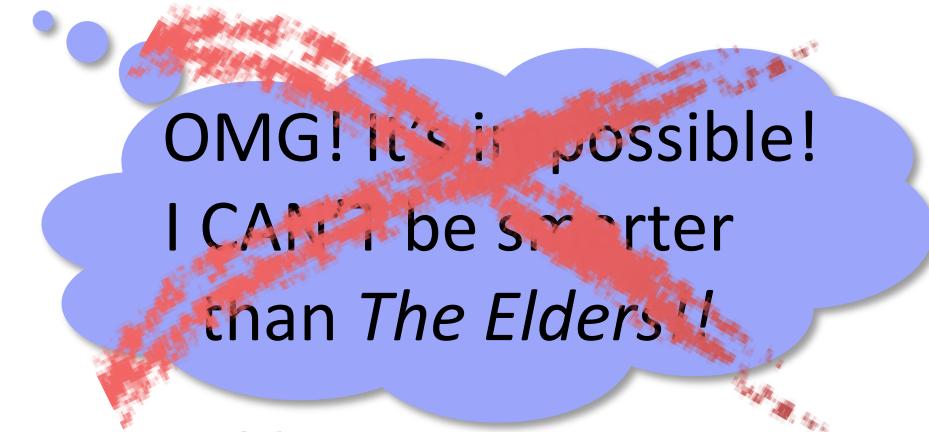
FRUSTRATION

USE IT!

Comprehension
Refactoring®
-Martin Fowler

Make the code speak for itself

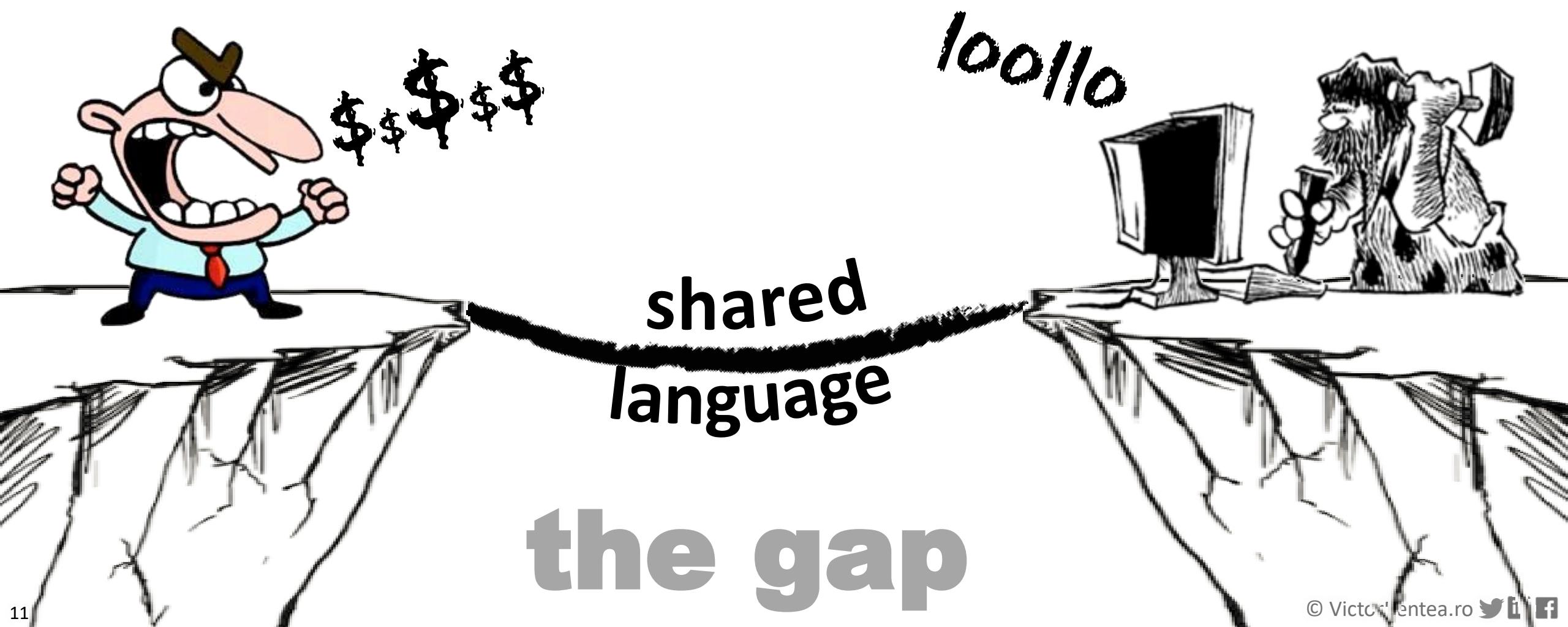
You found a better name.



Rename it !

It takes seconds with an IDE
(rarely fails: XMLs, JSON, reflection, API...)

Keep Code Close to Domain





Read it for free on: https://leanpub.com/ddd_first_15_years/cdddeu or Rentea.ro

More Functions, Less Names!



```
public void generateInvoice(Customer, Address, int, long)
```

```
public void generateInvoice(Customer, Address, int, long, Date)
```

Util Classes

Util Classes



At Least Collect Selectively

Discover Value Objects



`privateMethod(α, β)`

General-purpose?
+



Never need to mock it
(simple & safe)

`Util1.publicHelper(α, β)`



`VO vo = new VO(α, β);
vo.method();`



Enrich your Domain Language by Discovering Value Objects



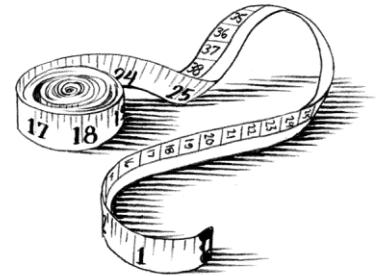
Simplify your code

Constraints

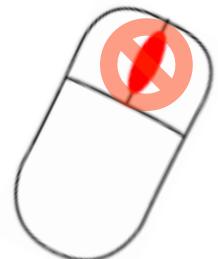
Methods

Reuse in larger Entities

Name Length



Variables: longer scope \Rightarrow longer name



Avoid
scrolling

1+ word

```
public class OrderService {
    public final static int STATUS_ACTIVE = ...
    private OrderRepository orderRepository;

    public void copy(Order from, Order to) {
        boolean wasDelivered;
        for (int i = 0; i < 100; i++) { ... }
    }
}
```

Global Name
*under the class name

Live Doc
shown by IDE

1 letter

Agenda

 Introduction

 Names

 Functions

 Classes

 Formatting & Comments

$$f(x)$$

A function **should do one thing**

They should be

SMALL

Functions Should be **SMALL !!**

How small ?

Why so small ?!!

... **5 lines***

Every function!?
That's absurd!

So they do just **1 THING**

What CAN you do in 5 lines ??

You CAN find a good name for it ➔

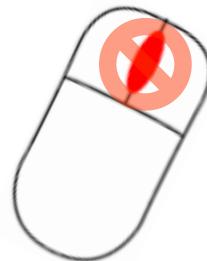


* Actually, Uncle Bob now says 5 lines is too much => extract 'till you drop

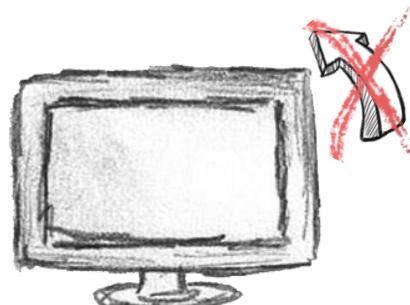
Functions Should be **SMALL** !!

How small ?

Harder to understand,
→ smaller



Smaller than a page of your IDE !



The screenshot shows the Borland C++ IDE interface. The menu bar includes File, Edit, Search, Run, Compile, Debug, Project, Options, Window, and Help. The title bar displays the path \BORLANDC\EXAMPLES\VCIRC.CPP and the file name TCDISPLAY.C. A status bar at the bottom shows '1' on the left and '3=[↑]' on the right. The main window contains C++ code for setting colors and writing strings. A modal 'About' dialog box is centered, containing the text: 'Borland C++ Version 3.1 Copyright (c) 1990, 1992 by Borland International, Inc.' An 'OK' button is at the bottom of the dialog. The bottom of the screen shows a file list with columns for File name, Location, Size, Code, and Data. The file TCDISPLAY.C is selected.

| File name | Locat | S | Code | Data |
|---------------|-------|-----|------|------|
| TCALC.C | . | n/a | n/a | n/a |
| • TCDISPLAY.C | . | 284 | 1694 | 425 |
| TCINPUT.C | . | 210 | 1074 | 84 |
| TCOMMAND.C | . | 830 | 5583 | 1194 |

\approx 20 lines

The function is a landscape

Easy to remember by its author

But for the team,



complete wilderness
if you don't...

```
protected void doExecution()
throws be.bpost.schemas.services.service.postal.sndinternalconsumers.v001.Exception {
    // new RetrieveMerchantDetailsResponse();
    if (parameters.getMerchantManagementSearchCriteria().getPartyIdList() != null
        && !parameters.getMerchantManagementSearchCriteria().getPartyIdList().isEmpty()) {
        List<PartyIdType> partyIdTypes = parameters.getMerchantManagementSearchCriteria().getPartyIdList();
        Long[] ids = new long[partyIdTypes.size()];
        int i = 0;
        for (PartyIdType partyIdType : partyIdTypes) {
            ids[i] = partyIdType.getId();
            i++;
        }
        result.setMerchantDetailList(ShopConverter.convertDomainToCodomo(ShopConverter.PARTY_SPECIFIC,
            getShopService().findShopsByIds(ids), null, null));
    }
    // Platform specific
    else if (parameters.getMerchantManagementSearchCriteria().getContractedPlatformIdList() != null
        && !parameters.getMerchantManagementSearchCriteria().getContractedPlatformIdList().isEmpty()) {
        Long platformId = Long.valueOf(parameters.getMerchantManagementSearchCriteria()
            .getContractedPlatformIdList().get(0));
        ServicePlatform platform = getServicePlatformEager(platformId);
        if (platform == null) {
            throw new SdlBusinessException(SdlExceptionCause.ENTITY_NOT_FOUND, "platform", platformId);
        }
        Set<Shop> shops = getShopService().findServicePlatformByPlatformId(platformId);
        if (shops != null &amp; !shops.isEmpty()) {
            String body = shopService.getConveniencesWithoutTime();
            final int appointmentsMaxFutureDays = ApplicationConfiguration.getInstance()
                .getAppointmentsMaxFutureDays();
            Date endDay = org.apache.commons.lang.time.DateUtils.addDay(sToday, appointmentsMaxFutureDays);
            Map<Shop, Map<ServicePlatform, XMLGregorianCalendar>> earliestPossibleDeliveryStartTimestamps =
                new HashMap<Shop, Map<ServicePlatform, XMLGregorianCalendar>>();
            Date earliestPossibleDeliveryStartTimeStamp = TimeProvider.DEFAULT
                .getNextAvailableDeliveryStartTimeStamp();
            Map<Shop, Map<ServicePlatform, SortedSet<Date>>> availableShopDaysForAll =
                new HashMap<Shop, Map<ServicePlatform, SortedSet<Date>>>();
            final List<WeekPlanning> platformWeekPlanningsInDateRange = getPlatformService()
                .getWeekPlanningsInDateRange(platform, toDay, endDay);
            for (Shop shop : shops) {
                SortedSet<Date> availableShopDays = calculateAvailableShopDays(shop,
                    platformWeekPlanningsInDateRange, toDay, appointmentsMaxFutureDays);
                Map<ServicePlatform, SortedSet<Date>> availableShopDaysMap =
                    availableShopDaysForAll.put(platform, availableShopDays);
                availableShopDaysForAll.put(shop, availableShopDays);
            }
            Map<ServicePlatform, XMLGregorianCalendar> earliestPossibleDeliveryStartTimestampsForPlatform =
                new HashMap<ServicePlatform, XMLGregorianCalendar>();
            earliestPossibleDeliveryStartTimeStampForPlatform.put(platform, DateUtils
                .extractXmlDateFromTime(calculateForFirstPossibleDeliveryStartTimeStamp(
                    earliestPossibleDeliveryStartTimeStamp, availableShopDays)));
            earliestPossibleDeliveryStartTimeStampForPlatform.put(shop, DateUtils
                .extractXmlDateFromTime(calculateForFirstPossibleDeliveryStartTimeStamp(
                    earliestPossibleDeliveryStartTimeStamp, availableShopDays)));
        }
        result.setMerchantDetailList(ShopConverter.convertDomainToCodomo(
            ShopConverter.PLATFORM_SPECIFIC, shops, availableShopDaysForAll,
            earliestPossibleDeliveryStartTimeStamp));
    } else {
        result.setMerchantDetailList(ShopConverter.convertDomainToCodomo(
            ShopConverter.PARTY_SPECIFIC, shops, availableShopDaysForAll,
            earliestPossibleDeliveryStartTimeStamp));
    }
}
// Party specific (based on prc id)
else if (parameters.getMerchantManagementSearchCriteria().getPartySearchCriteria() != null
    && parameters.getMerchantManagementSearchCriteria().getPartySearchCriteria()
        .getExternalIdList() != null
    && !parameters.getMerchantManagementSearchCriteria().getExternalIdList().isEmpty()) {
    List<ExternalIdType> externalIdTypes = parameters.getMerchantManagementSearchCriteria()
        .getPartySearchCriteria().getExternalIdList();
    for (Shop shop : shops) {
        if (shop.getExternalId() == null) {
            // get ID where system == "PRS" -> this is the PRSID
            for (ExternalIdType externalIdType : externalIdTypes) {
                if (externalIdType.getSystem().equalsIgnoreCase("PRS")) {
                    shop.setExternalId(externalIdType.getId());
                }
            }
        }
        Set<Shop> shops = new HashSet<Shop>();
        SetServicePlatformService servicePlatformService = getShopService();
        Shop shop = servicePlatformService.getShop(Long.valueOf(merchantId));
        if (shop == null) {
            throw new SdlBusinessException(SdlExceptionCause.ENTITY_NOT_FOUND, "merchant", merchantId);
        }
        shops.add(shop);
        SetServicePlatform platforms = getPlatformService().getServicePlatformEager(
            shop.getAvailableServicePlatform());
        Date endDay = DateUtils.addCurrentDateWithoutTime();
        final int appointmentsMaxFutureDays = ApplicationConfiguration.getInstance()
            .getAppointmentsMaxFutureDays();
        Date endDay = org.apache.commons.lang.time.DateUtils.addDay(sToday, appointmentsMaxFutureDays);
        Map<Shop, Map<ServicePlatform, SortedSet<Date>>> availableShopDaysForAll =
            new HashMap<Shop, Map<ServicePlatform, SortedSet<Date>>>();
        availableShopDaysForAll.put(shop, new HashMap<ServicePlatform, SortedSet<Date>>());
        Map<Shop, Map<ServicePlatform, XMLGregorianCalendar>> earliestPossibleDeliveryStartTimestamps =
            earliestPossibleDeliveryStartTimeStampForShop(shop);
        new XmlGregorianCalendar();
        for (ServicePlatform sp : platforms) {
            earliestPossibleDeliveryStartTimeStampForSP =
                earliestPossibleDeliveryStartTimeStampForShop(sp);
            earliestPossibleDeliveryStartTimeStampForSP = availableShopDaysForAll
                .get(sp);
            SortedSet<Date> availableShopDays = calculateAvailableShopDays(shop, getServicePlatformService(),
                getAvailablePlanningsInDateRange(sp, today, endDay), today, appointmentsMaxFutureDays);
            availableShopDaysForSP.put(sp, availableShopDays);
            earliestPossibleDeliveryStartTimeStampForSP.get(sp).put(
                sp,
                DateUtils.extractXmlDateFromTime(calculateForFirstPossibleDeliveryStartTimeStamp(
                    earliestPossibleDeliveryStartTimeStamp, availableShopDays)));
        }
        result.setMerchantDetailList(ShopConverter.convertDomainToCodomo(ShopConverter.PARTY_SPECIFIC_PRS,
            shops, earliestPossibleDeliveryStartTimeStamp));
    }
}
```

Change Request #323

WHAT DO YOU DO ?

Here, I also need
to do this:

```
if (cr323) {
    doOtherStuff();
}
```

```
protected void doExecution()
throws be.bpost.schema.services.postal.sndinternalconsumers.v001.Exception {
    result = new RetrieveMerchantDetailsResponse();
    // Party specific (based on list of party IDs)
    if ((parameters.getMerchantManagementSearchCriteria().getPartyIdList() != null
        && !parameters.getMerchantManagementSearchCriteria().getPartyIdList().isEmpty())
        || List<PartyIdType> partyIdTypes = parameters.getMerchantManagementSearchCriteria().getPartyIdList()
        .getPartyId());
    Long[] ids = new long[partyIdTypes.size()];
    int i = 0;
    for (PartyIdType partyIdType : partyIdTypes) {
        ids[i++] = long.parseLong(partyIdType.getId());
    }
    result.setMerchantDetailList(ShopConverter.convertDomainToCodomo(ShopConverter.PARTY_SPECIFIC,
        getShopService().findShopsByIds(ids, null, null));
}

// Platform specific
else if ((parameters.getMerchantManagementSearchCriteria().getContractedPlatformIdList() != null
    && !parameters.getMerchantManagementSearchCriteria().getContractedPlatformIdList().isEmpty())
    || assumedPlatformId != null) {
    Long platformId = long.valueOf(parameters.getMerchantManagementSearchCriteria()
        .getContractedPlatformIdList().getPlatformId().get@0);
    ServicePlatform platform = getPlatformService().getServicePlatformEager(platformId);
    if (platform == null) {
        throw new Sd1BusinessException(Sd1ExceptionCause.ENTITY_NOT_FOUND, "platform", platformId);
    }

    Set<Shop> shops = getShopService().findByServicePlatformById(platformId);
    if (shops.isEmpty()) {
        Date today = DateUtils.getCurrentDateInHourTime();
        final int appointmentsMaxFutureDays = ApplicationConfiguration.getInstance()
            .getAppointmentsMaxFutureDays();
        Date endDay = org.apache.commons.lang.time.DateUtils.addDays(today, appointmentsMaxFutureDays);
        HashMap<Shop, Map<ServicePlatform, XMLGregorianCalendar>> earliestPossibleDeliveryStartTimestamps =
            new HashMap<Shop, Map<ServicePlatform, XMLGregorianCalendar>>();
        Date earliestPossibleDeliveryStartTimeStamp = TimeProvider.DEFAULT_DATE;
        for (Shop shop : shops) {
            SortedSet<Date> availableShopDays = calculateAvailableShopDays(shop,
                platform.getWeekPlanningInDateRange, toDay, appointmentsMaxFutureDays);
            Map<ServicePlatform, SortedSet<Date>> availableShopDaysForAll =
                new HashMap<ServicePlatform, SortedSet<Date>>();
            final List<WeekPlanning> platformWeekPlanningInDateRange = getPlatformService()
                .getWeekPlanningInDateRange(platform);
            for (Shop shop : shops) {
                SortedSet<Date> availableShopDays = calculateAvailableShopDays(shop,
                    platform.getWeekPlanningInDateRange, toDay, appointmentsMaxFutureDays);
                availableShopDaysForAll.put(shop, availableShopDays);
                Map<ServicePlatform, SortedSet<Date>> earliestPossibleDeliveryStartTimestampsForPlatform =
                    new HashMap<ServicePlatform, SortedSet<Date>>();
                earliestPossibleDeliveryStartTimestampsForPlatform.put(platform, DateUtils
                    .extractXmLDateFrom(collectEarliestPossibleDeliveryStartTimestamps(
                        earliestPossibleDeliveryStartTimeStamp, availableShopDays)));
                earliestPossibleDeliveryStartTimeStamp = earliestPossibleDeliveryStartTimeStamp
                    .compareTo(DateUtils.extractXmLDateFrom(earliestPossibleDeliveryStartTimestampsForPlatform));
            }
            result.setMerchantDetailList(ShopConverter.convertDomainToCodomo(
                ShopConverter.PLATFORM_SPECIFIC, shops, availableShopDaysForAll,
                earliestPossibleDeliveryStartTimeStamp));
        } else {
            result.setMerchantDetailList(ShopConverter.convertDomainToCodomo(
                ShopConverter.PLATFORM_SPECIFIC, shops,
                new HashMap<Shop, Map<ServicePlatform, SortedSet<Date>>>(),
                new HashMap<Shop, Map<ServicePlatform, XMLGregorianCalendar>>()));
        }
    }
    // Party specific based on PSS
    else if ((parameters.getMerchantManagementSearchCriteria().getPartySearchCriteria() != null
        && parameters.getMerchantManagementSearchCriteria().getPartySearchCriteria().getExternalIdList() != null
        && !parameters.getMerchantManagementSearchCriteria().getPartySearchCriteria().getExternalIdList().isEmpty())
        || List<ExternalIdType> externalIdTypes = parameters.getMerchantManagementSearchCriteria()
        .getPartySearchCriteria().getExternalIdList();
    // For the shop/merchant split we will map PRS IDs to Shop IDs (in order to minimize the impact)
    // See http://wiki.bpost.be/display/BdNvp/Story+4.4+Impact+on+channel+services+for+more+details
    // PRS ID where system="null" -> this is the PRSId
    for (ExternalIdType externalIdType : externalIdTypes) {
        if (externalIdType.getSystem().equalsIgnoreCase("PRS")) {
            merchantId = externalIdType.getId();
        }
    }
    Set<Shop> shops = new HashSet<Shop>();
    Shop shop = getShopService().getShop(long.valueOf(merchantId));
    if (shop == null) {
        throw new Sd1BusinessException(Sd1ExceptionCause.ENTITY_NOT_FOUND, "merchant", merchantId);
    }
    shops.add(shop);

    Set<ServicePlatform> platforms = getPlatformService().getServicePlatformEager(
        shop.getServicePlatforms());
    Date toDay = DateUtils.getCurrentDateInHourTime();
    final int appointmentsMaxFutureDays = ApplicationConfiguration.getInstance()
        .getAppointmentsMaxFutureDays();
    Date endDay = org.apache.commons.lang.time.DateUtils.addDays(toDay, appointmentsMaxFutureDays);
    HashMap<Shop, Map<ServicePlatform, SortedSet<Date>> availableShopDaysForAll =
        new HashMap<Shop, Map<ServicePlatform, SortedSet<Date>>();
    availableShopDaysForAll.put(shop, new HashMap<ServicePlatform, SortedSet<Date>>());
    HashMap<Shop, Map<ServicePlatform, XMLGregorianCalendar>> earliestPossibleDeliveryStartTimestamps =
        new HashMap<Shop, Map<ServicePlatform, XMLGregorianCalendar>>();
    earliestPossibleDeliveryStartTimeStamp = TimeProvider.DEFAULT_DATE;
    for (ServicePlatform sp : platforms) {
        Date earliestPossibleDeliveryStartTimeStamp = TimeProvider.DEFAULT_DATE;
        Map<ServicePlatform, SortedSet<Date>> availableShopDaysForSP =
            availableShopDaysForAll.get(sp);
        for (Shop shop : shops) {
            SortedSet<Date> availableShopDays = calculateAvailableShopDays(shop, getServicePlatform()
                .getWeekPlanningInDateRange(sp, toDay, endDay), toDay, appointmentsMaxFutureDays);
            availableShopDaysForSP.put(sp, availableShopDays);
            earliestPossibleDeliveryStartTimestamps.get(shop).put(sp, DateUtils
                .extractXmLDateFrom(collectEarliestPossibleDeliveryStartTimestamps(
                    earliestPossibleDeliveryStartTimeStamp, availableShopDays)));
        }
    }
    result.setMerchantDetailList(ShopConverter.convertDomainToCodomo(ShopConverter.MERCH_PSS,
        shops, availableShopDaysForAll, earliestPossibleDeliveryStartTimeStamp));
}
}
```

..., boolean cr323)



EXTRACT
METHOD

EXTRACT METHOD



methods

Performance vs Clean Code

Smaller methods run faster !

(get  faster)

Google “Just-In-Time Compiler Optimizations”

JMH

p6spy

jMeter

GlowRoot

DynaTrace...



Performance



Measure, Don't Guess®

*We should forget about **small** inefficiencies 97% of the time,*

Premature optimization is the root of all evil

– Donald Knuth

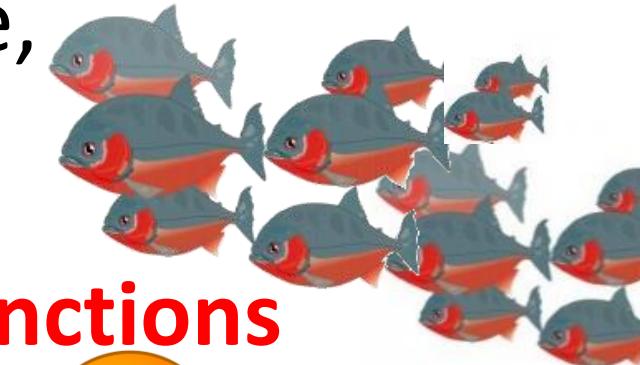
Scared of small functions?

Instead of one familiar landscape,

(The Author)



You're juggling with **dozens of small functions**



You can't even recall all their names



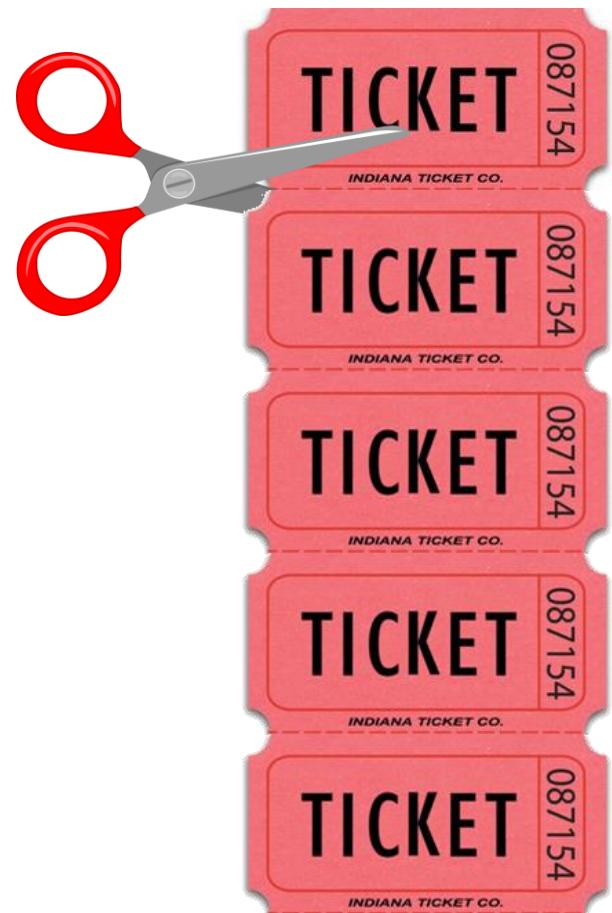
sendActivation
EmailToCustomer()

but,

The Team will thank you!

removeAllPast
CancelledOrders
ofConsumer()

Flat Functions



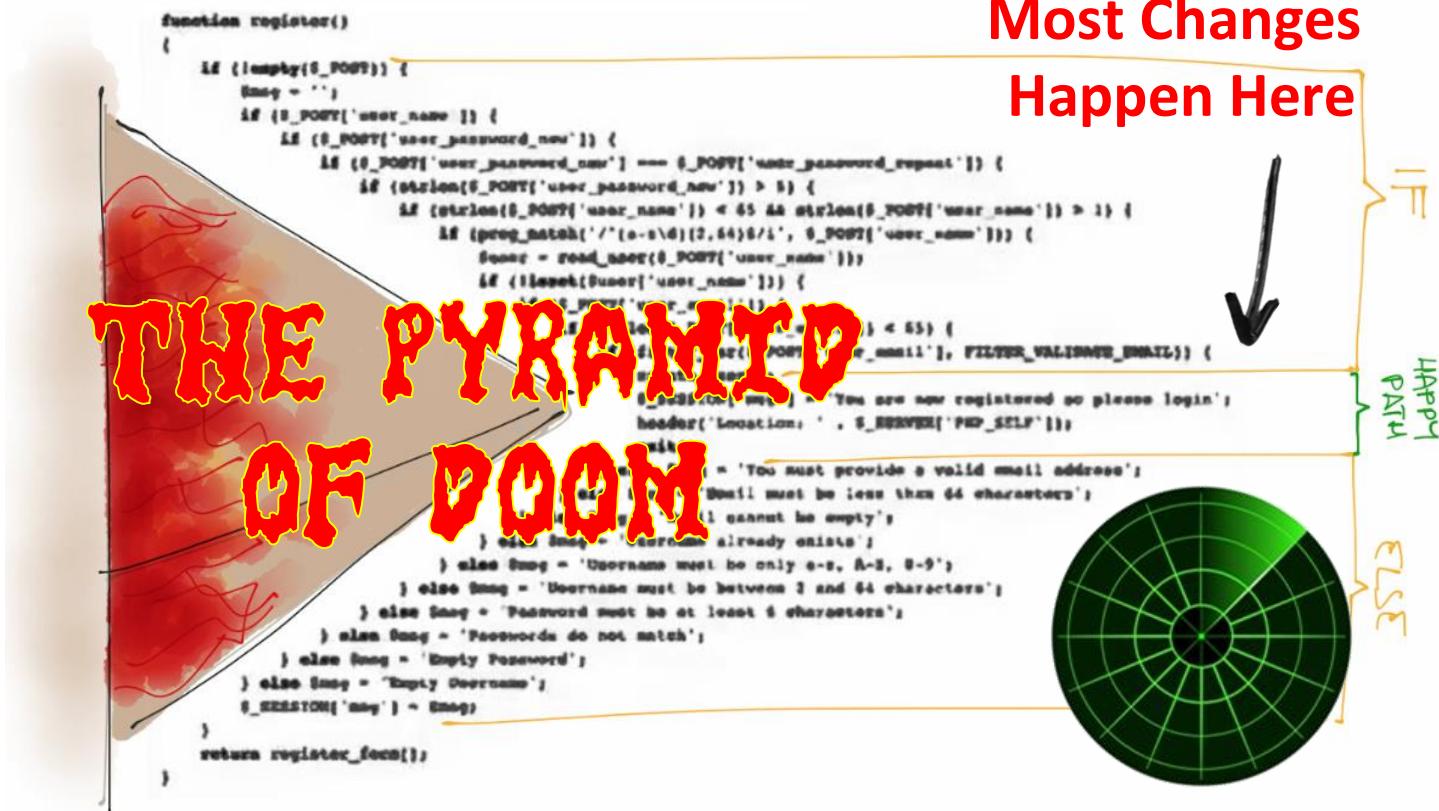
Flat Functions ✓ Are Easy to Break into Pieces



Flat Functions



Deeply Nested Functions



Hard to Break

↑↑ Cognitive Complexity

How to Flatten Functions?

■ if

- Anemic else branch → flip → Guard Clauses (early returns/throws) → next slide
- Comparable heavy then/else branch → Split in 2 functions

■ for

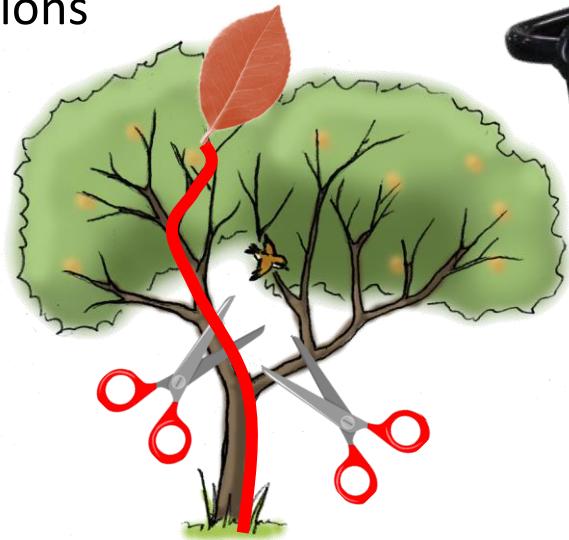
- Split-Loop (careful with side-effects outside of loop)
- Extract pure function from body → ?Streams

■ switch

- The 3 Hygienic Switch Rules (enhanced in Java 17 LTS)

■ try

- Avoid intermediate catch → Global Exception Handler



worse boolean
traverses deep code

Dissociative Identity Disorder Code



An **else** follows...



```
public List<Integer> stringsToInts(List<String> strings) {  
    if (strings != null) {  
        List<Integer> integers = new ArrayList<>();  
        for (String s : strings) {  
            →|→|integers.add(Integer.parseInt(s));  
        }  
        return integers;  
    } else {  
        return null;  
    }  
}
```



How to simplify it ?

"The Pyramid of Doom"

Ever heard about the "**Single Return Point**" principle? They got it wrong! → [Original definition](#)

```
public List<Integer> stringsToInts(List<String> strings) {  
    if (strings == null) {  
        return null; → Early Return  
    }  
    List<Integer> integers = new ArrayList<>();  
    for (String s : strings) {  
        integers.add(Integer.parseInt(s));  
    }  
    return integers;  
}
```



Guard Clauses



Less TABs => easier to read

Fail the build if you detect 3+ TABs in a new method →|→|→|

Can Small Functions be BAD?

Bad Name

= nume prost -> nu ai ce nume sa-i deacat func1() mai bine 10 linii de cod.

Too many parameters/outputs

Return array ?

NU ! return array. DA return structuri.

Fa o clasa! Da poti. Ai voie! Avem voie sa creem clase! Sa gasim noi concepte. sa exprimam noi idei. noi clase.

```
f() {  
    return [a, b];  
}
```

NU array-uri peste tot. De ce. Un array e o structura degenerata. e o clasa ratata.

Daca nu poti sa numesti acea clasa noua-> semn ca nu tre sa introduci 2 lucruri odata.

E ca hartia igienica/masca: de unica folosinta. class PentruFunctiaX {}

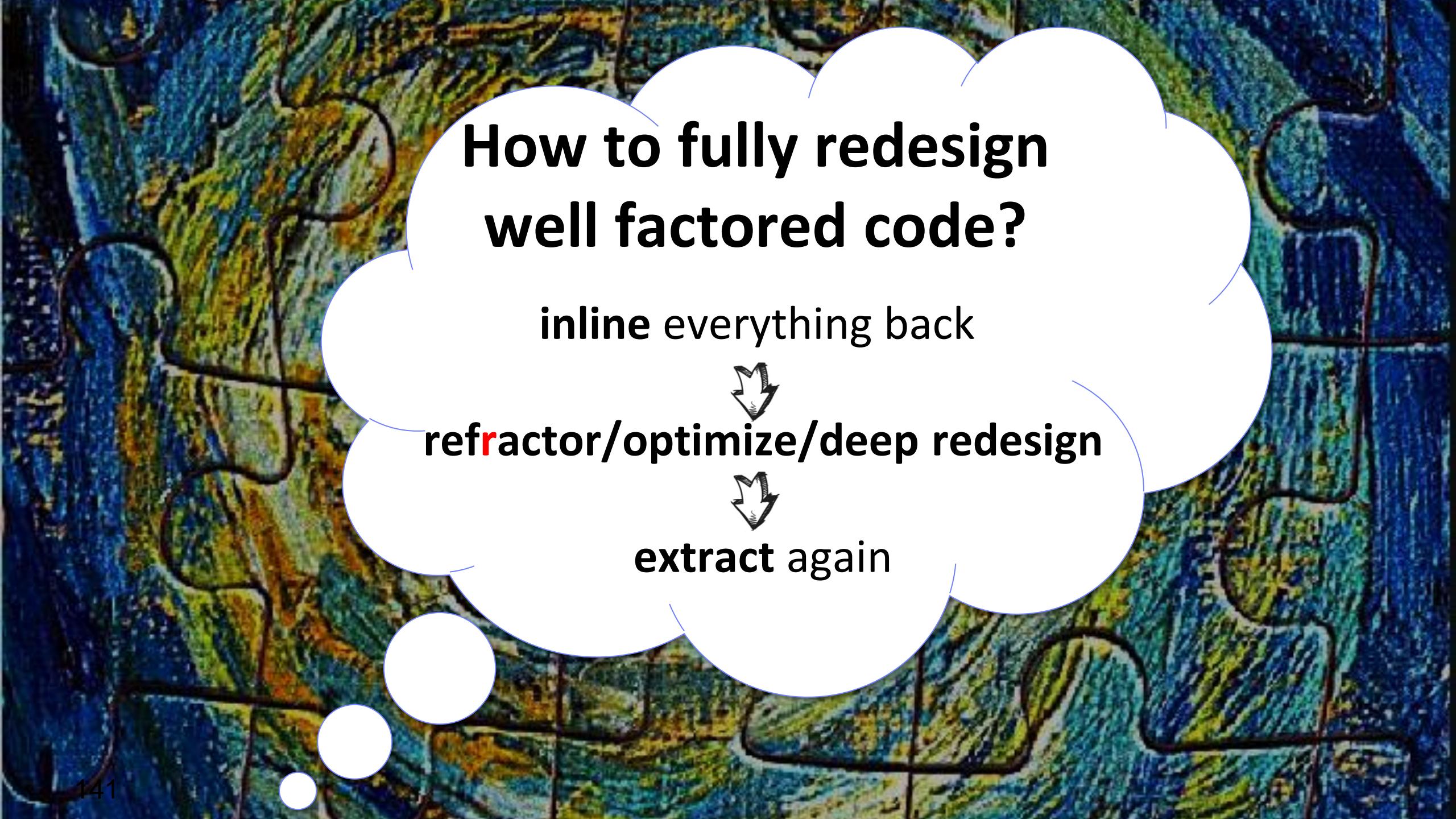
= cand sunt prea simple. cand numele lor nu explica mai mult decat codul in sine. Cand zici "Duuu!hhh!" cand deschizi fct.

Orice fct tre sa ascunda ceva complexitate.



Can too small methods be bad ?





How to fully redesign well factored code?

inline everything back



refactor/optimize/deep redesign



extract again



Max 3 parameters

```
myBadMethod("John", "Michael", "Paris", "St. Albergue");
```

streetName ?!

a) does too many things ?



b) just passes down the args ?



→ Try to introduce a Parameter Object/DTO/vo

```
address.setStreetName("Paris");  
...  
myBadMethod(address);
```

... and see how much code it simplifies

Overengineering?

c) common params -> fields [OOP]

SRP

No boolean params



=> 4

(usually 3)

```
removeOrders(customer, false, true);
```

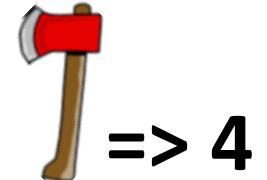
= changed + fear = annoying

No Boolean Parameters



SRP

No boolean params



```
removeOrders(customer, false, true);
```

(usually 3)



What if **null**?

No nullable params



```
if (customer != null) {...} else {...}
```

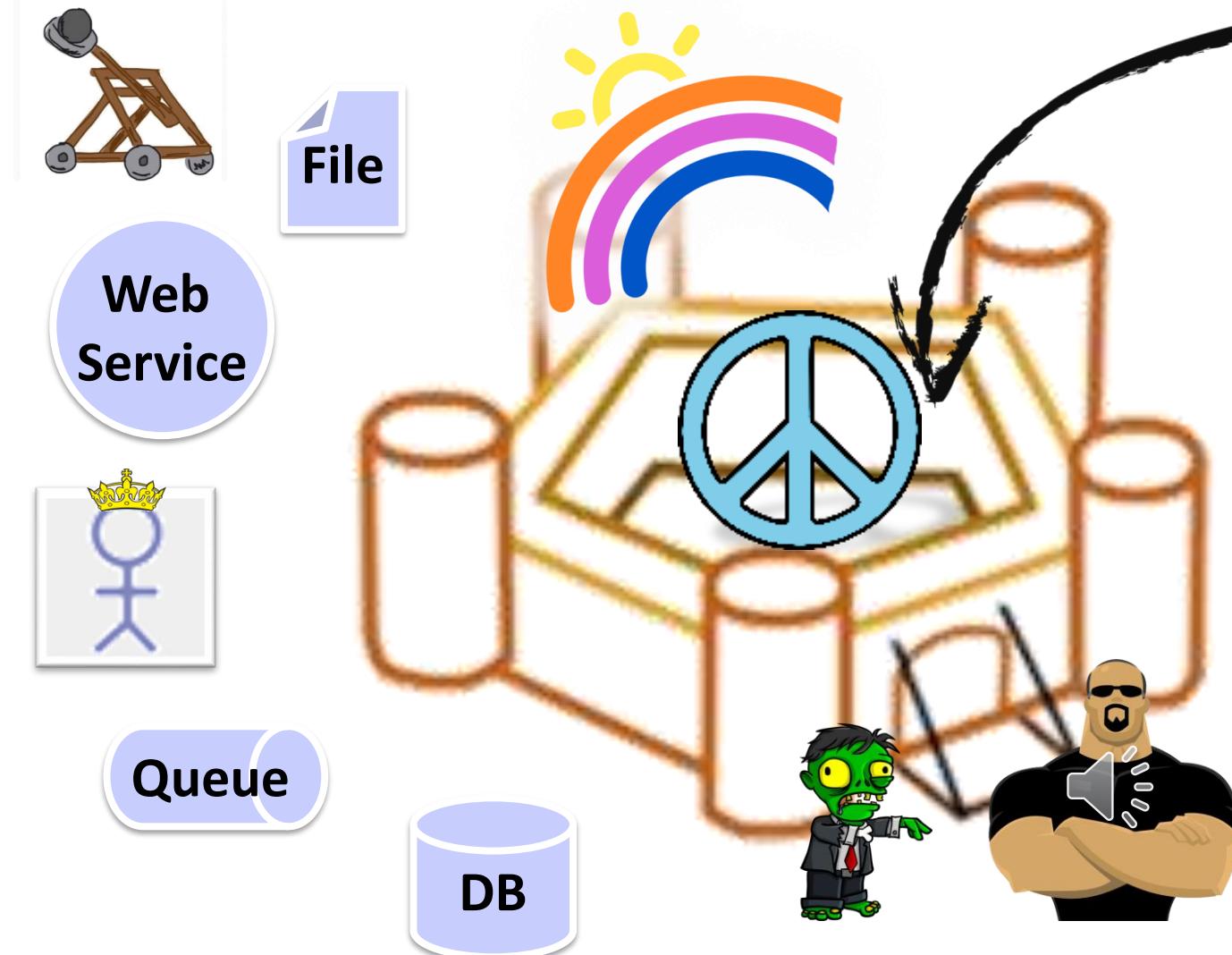


What about invalid data?

,nor **Optional<>**

NULL WARS

(corrupt data)



Avoid returning **NULL**

- Throw exception early
- Wrap it in an Optional<>



null with biz meaning?

vs Null Object Pattern

Defensive Programming

Thoroughly check data
only at the boundaries

Optional vs Early Throw

> 99% cases

`discountService.getDiscount(customer): Discount` (or null when none)

`Optional<Discount>`

signal the possible absence,
and let caller decide what to do for no discount:

`.map()`
`.orElseThrow()`
`.orElse();`

(Disclaimer: hard to tell in large codebases)

Sometimes your caller does `.get()`
`.orElseThrow()` every singletime

`customerRepo.findById(id): Optional<Customer>`

Trick: provide both Optional and throwing variants

`customerRepo.findExactlyOneById(id): Customer` (+ throw when not found)

annoying?



Code: <https://github.com/victorrentea/exceptions-guide.git>

I wrote a series of articles with further details on this topic: <https://victorrentea.ro/blog/exception-handling-guide-in-java/>

The Definitive Guide to Working with Exceptions in Java

Victor Rentea

Java Champion

Simple Design, Refactoring, Unit Testing

Founder of
Bucharest **Software Craftsmanship** Community

Join us on *Meetup*

Blog, Talks, Goodies:

VictorRentea.ro

Independent Trainer



8 years

2000 devs

40 companies

400 days
(100+ online)

Spring

Hibernate

Functional Prog

Design Patterns

Clean Code
Refactoring

Unit Testing

any lang

Reactive

Java Performance

Training for you or your company:
VictorRentea.ro



Posting
Good Stuff on:

Clean Exception Handling



A bit of history

At the beginning

There were no functions

At the beginning

There were no exceptions

40 years ago, in C:

```
int errno = f(...);
```

35y, C++: Invisible exceptions

25y, Java: Checked vs Runtime

The Age of Libraries



This talk is NOT
about library development

In web apps today, when handling exceptions,

We don't Recover

We DIE!



Checked Exceptions,
today

code

Diaper Anti-Pattern

= *catches all the sh*t*

+ Runtime **bugs**

```
} catch (Exception e) {  
    // TODO waste nights  
}
```



a.k.a. Swallowing Exceptions

Shawarma-style
Exception-Handling



FUN FACT!

System.err

might NOT be captured in your log file!



IDE default code block

```
} catch (Exception e) {  
    → e.printStackTrace();  
}
```



RuntimeExceptions won the War !

because they don't annoy us

~~throws~~

~~try~~

~~catch (Exception t) {/*surprise!*/}~~



ABSTRACTION LEAK

Why should I know
about that IOException ?

What to do with Checked Exceptions?

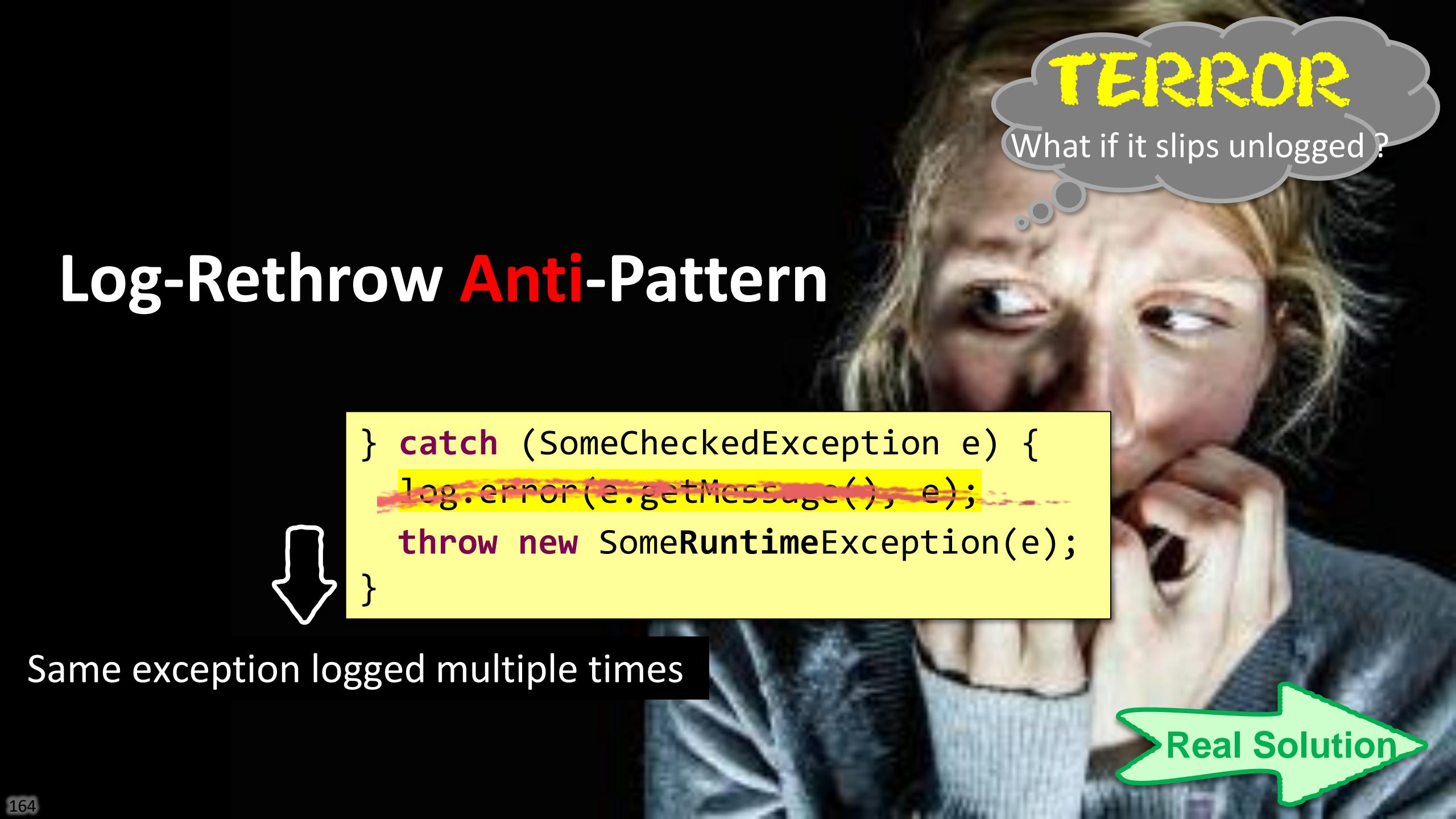
Wrap them in Runtimes

```
} catch (SomeCheckedException e) {  
    throw new SomeRuntimeException(e);  
}
```

Dynamically lower log level via
Spring Actuator or Logback JMX Bean

'log.trace(e.getMessage())'

Ignoring recurrent exception?
(eg. a poller)



TERROR

What if it slips unlogged ?

Log-Rethrow Anti-Pattern

```
} catch (SomeCheckedException e) {  
    log.error(e.getMessage(), e);  
    throw new SomeRuntimeException(e);  
}
```



Same exception logged multiple times



Real Solution

Checked -> Runtime



Core Logic

Only RuntimeExceptions allowed

~~try {~~

Legacy Code

Global Exception Handler

Logs and reports any unhandled exceptions

@ExceptionHandler (Spring)

ExceptionMapper (JavaEE)

manual try – catch (custom threads)

Presenting Errors to Users

What to show them?

Never a Stack Trace: Security

Exception's message

(OK for small apps)

How to unit-test that?

How about MVC?

`int`

Not developer-friendly

Finite values set

`enum`



code

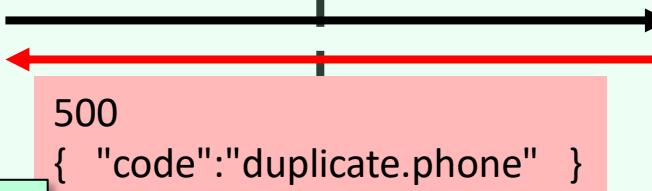
Frontend

Backend

A) Translate on FE



errors.json



Global Exception Handler

Ex

B) Translate on BE



500
{ "userMessage": "There's another..." }

messages.properties



Global Exception Handler

Ex

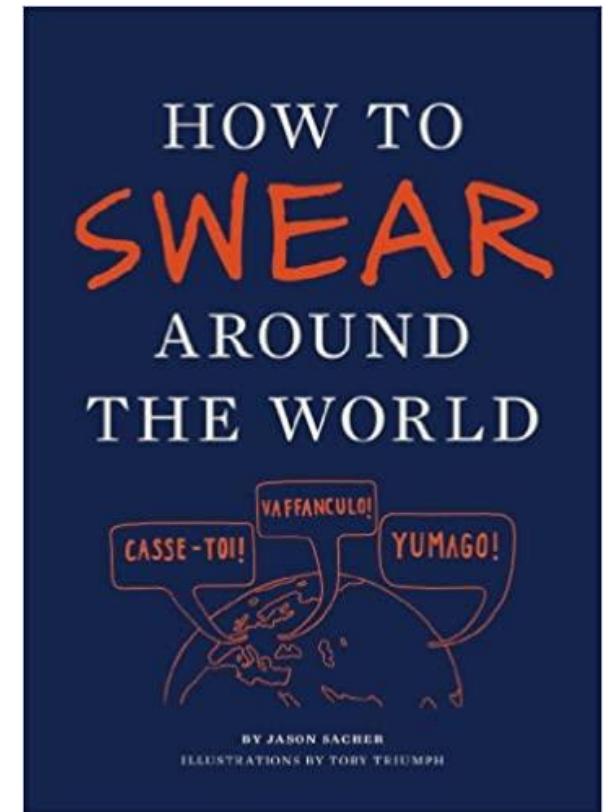
Translating Exceptions

```
class MyException extends RuntimeException {
    public enum ErrorCode {
        GENERAL, // Any case not useful to users
        PHONE_ALREADY_REGISTERED,
        ...
    }
    private final ErrorCode code;
    private final Object[] params;
    ...
}
```

Check at startup

messages.properties

GENERAL = Oups!
 PHONE_ALREADY_REGISTERED = The phone {0} is assigned to another user

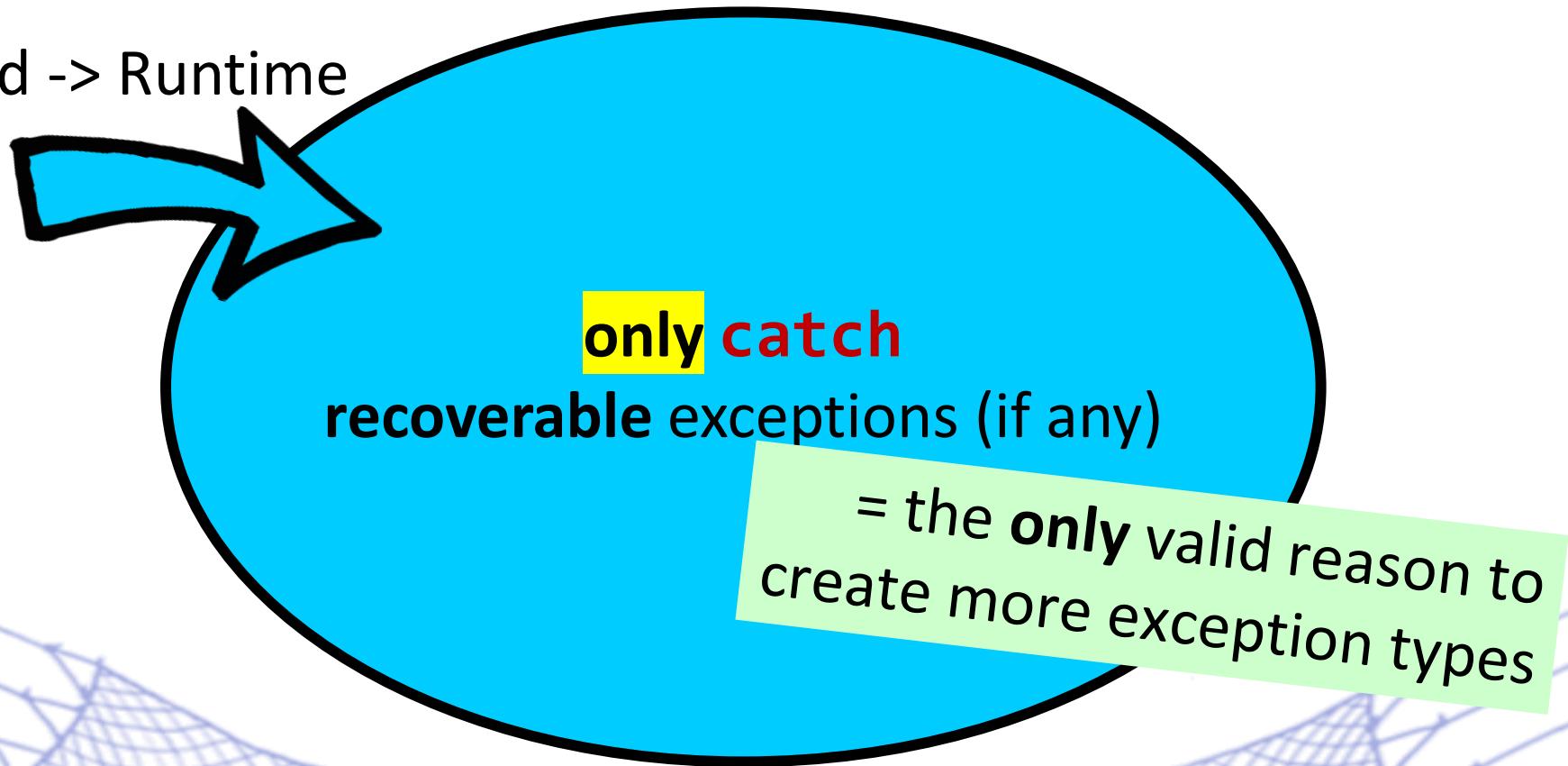


_fr

_ro

_es

Checked -> Runtime



Global Exception Handler

Debugging an Exception

- Check the logs above?
- Breakpoints? 

Why?

→ To inspect some **key variable**

Catch-Rethrow-with-Debug Pattern

```
} catch (AEx e) {  
    throw new BEx("Info " + id, e);  
}
```

AEx(



Must-Have

Never Decapitate Exceptions !

```
    } catch (SomeEx e) {  
        throw new AnotherEx("Oops");  
    }
```



4 Reasons to Catch-rethrow

User-friendly message

(code)

Tests

(code)

Developers

(rethrow with debug message)

Rethrow as Runtime → [@SneakyThrows](#)

```
} catch (SomeEx e) {  
    throw new RuntimeException(e);  
}
```

(Lombok)
for use-case fatal errors

The Queen Of All Exceptions



The Lady In Red

by Chris De Burgh



I've never seen you looking so lovely as you did **tonight**, = on-call bug

I've never seen you **shine so bright**, = P1 production Incident

I've never seen **so many men** ask you if you wanted to dance, = unsolved for years

Looking for **a little romance**, = spend the night with you

given **half a chance**, = although they lack the knowledge/skills

And I have never seen that **dress you're wearing**, = 50-lines stack-trace

Or the **highlights in your hair** that catch your eyes, = the DEBUG logs before it

I have been blind;

N P E

We don't defend against NULL

We **CHARGE** against it!



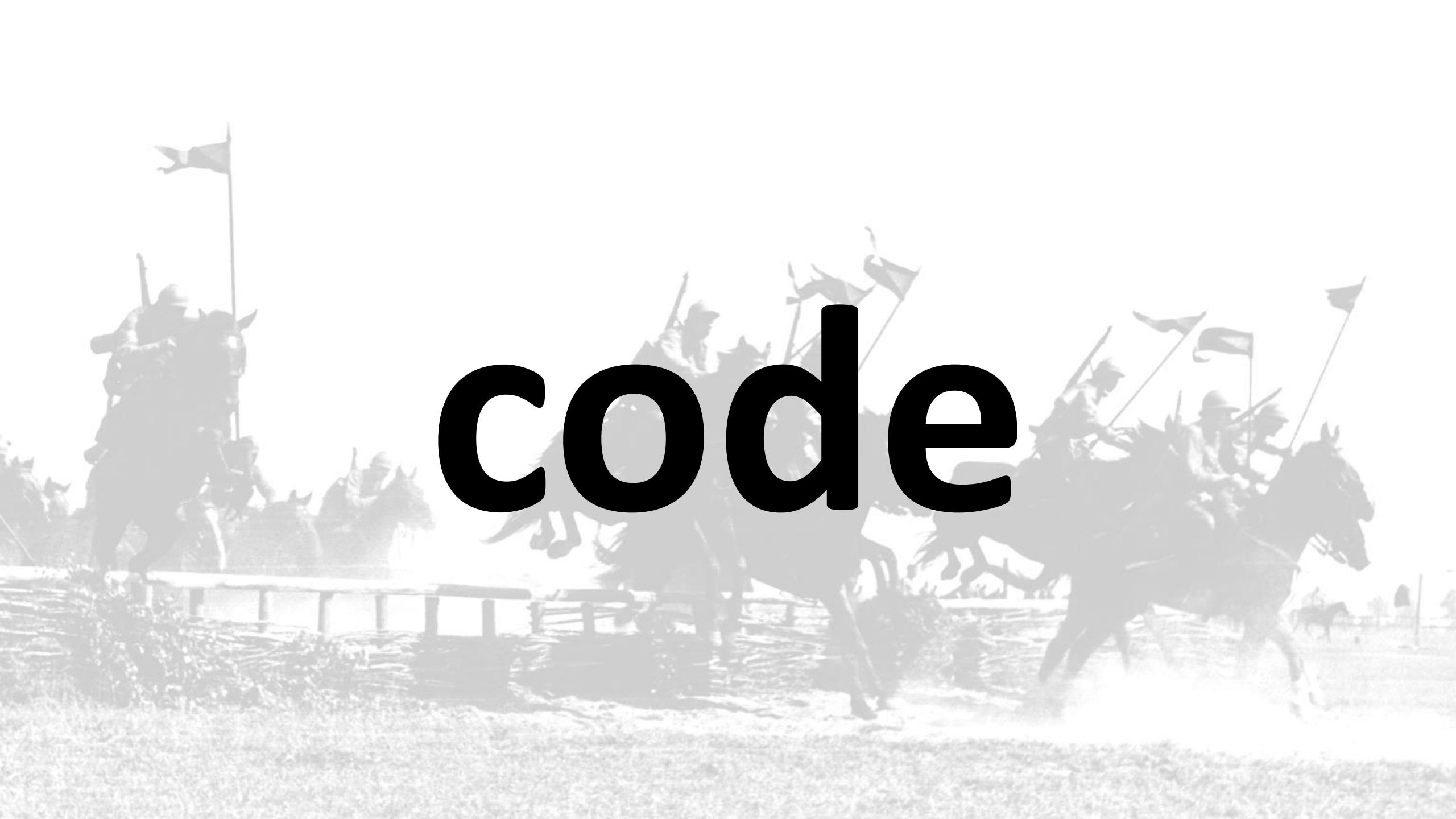


We don't defend against NULL

We CHARGE against it!

Throw early
for constraints

return Optional
for non-required field

A grayscale photograph depicting a medieval-style battle scene. In the foreground, several soldiers on horseback are shown in mid-motion, some carrying lances and others flags. The scene is set against a bright, overexposed background where more soldiers and horses are visible, creating a sense of depth and chaos.

code

Optional<>

`Customer.getMemberCard(): Optional<MemberCard>`

entity

for every nullable column



You, defeating the Null

`Customer.getMemberCard(): Optional<MemberCard>`

Java 8

Java 8 Functional Interfaces

(eg Function, Predicate, ...)

+

checked exceptions

= **pain**

code

Higher-order
Functions!



Wrap checked exceptions in lambdas

```
Function<> Unchecked.function(CheckedFunction<> f)  
          <dependency>
```

```
<groupId>org.jooq</group  
      id>
```

```
<artifactId>jool</artif  
actId>  
</dependency>
```

Try<>



Can hold *either* a result or an exception

Collect both results and exceptions in a single pass

<dependency>

<groupId>io.vavr</groupId>

<artifactId>vavr</artifactId>

Key Points

- Use Runtime; `@SneakyThrows`; Unchecked for lambdas (jool)
- Anti-Patterns: Diaper, Decapitation, Log-Rethrow
- Global Exception Handler
- Enum error codes *for users or tests*
- Catch-Rethrow-with-Debug
- Defeating NPE with early-throw, or Optional
- Try (vavr)

Further Reading, details and comments on this topic: <https://victorrente.ro/blog/exception-handling-guide-in-java/>

Training Topics:

- **Clean Code** + Refactoring
- Design Patterns
- Unit Testing + TDD
- Advanced FP with Java
- Spring
- Hibernate/JPA
- Reactive Programming
- Java Performance
- Pragmatic DDD

Further Reading, details and comments on this topic:
<https://victorrentea.ro/blog/exception-handling-guide-in-java/>



KEEP
CALM

AND

have a nice
exception



Blog ☆, Company Training, Masterclasses, Best Talks,...

!! PLEASE !!
Ask me anything!

Left-Over

- Async exceptions (Executors, @Async, CompletableFuture, Reactive-X)
- finally ~~{throw}~~
- AutoCloseable.close() ~~{throw}~~ -> Suppressed Exceptions – [link](#)
- throw before mutating object state
- Don't use exceptions for flow control (as a GOTO statement)
- InterruptedException – [link](#)
- Sealed classes + records + Switch Template Matching Java 16+

The End

Pure Functions and Immutable Objects



Victor Rentea

Java Champion

♥ Simple Design, Refactoring, Unit Testing ♥

Founder of
Bucharest **Software Craftsmanship** Community

Join us on *Meetup*

Best Talks, Goodies, Blog:
VictorRentea.ro

Independent Trainer & Consultant



8 years

2000 devs

40 companies

300+ days

Spring 

Hibernate

Func Prog in Java

Design Patterns
Pragmatic DDD

Clean Code
Refactoring

Unit Testing

any lang

Reactive-X

Java Performance

Training for you or your company:
VictorRentea.ro



Follow me:
 35K  4K  3K

do side-effects

return void

setActive(**true**):**void**

sendEmail(**Email**):**void**

Command-Query Separation

in 1994, by Bertrand Meyer

search(criteria):List

+INSERT

computePrice(flight):**int**

higher 2nd time

return results

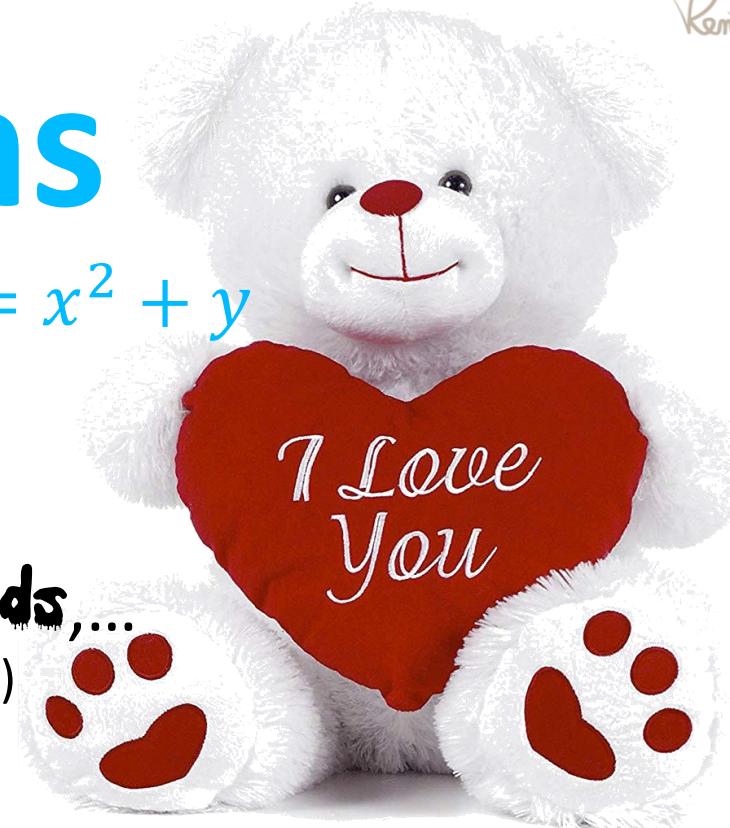
Pure Functions

Pure Functions

= *Mathematical Functions:* $f(x, y) = x^2 + y$

No side effects

No INSERTs, POSTs, queues, files, **Fields**,...
(logging doesn't count)



Referential Transparent ≠ Idempotent

Same Inputs → Same Output

No current time, random, GET, SELECT...



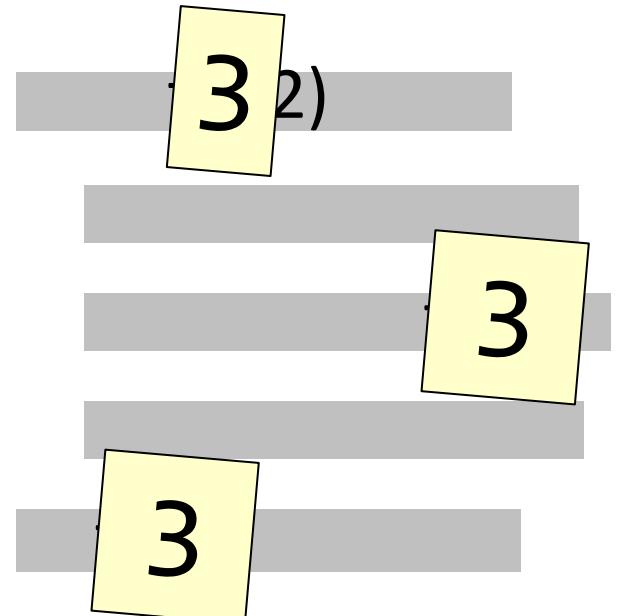
eg: **DELETE FROM X WHERE ID =**

Referential Transparent \neq Idempotent

$$f(1,2) = 3$$



$f(1,2)$ can be replaced with 3 everywhere



After calling f once



Calling it again n times
will not produce any extra changes

Pure Functions : Quiz

✓ **f1(int x) {return x + 1;}**

✗ **f2(Data d) {return ++d.x;}**

Probable side effects

✗ **f3() {d.incrementX(); return d.x;}**

NOT referential
transparent

✗ **f4() {return querySQL(...);}**

🤔 **f5(int y) { return this.x + y; } *is this immutable?***

✓ **f6(Data d, int y) { return d.getX() + y; }**

Expected to be pure

✓ **f7(int i) { if (i<0) throw new WrongInputException(); }**

Why we Love Pure Functions

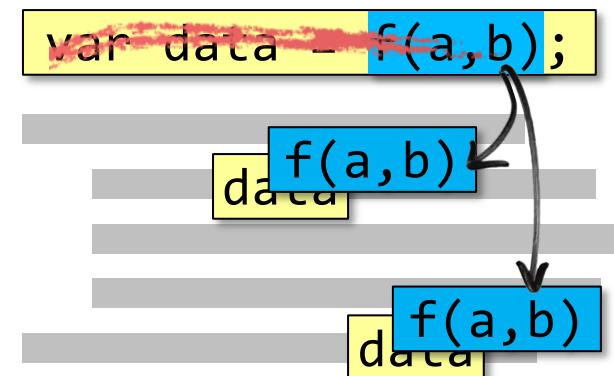
- No hidden inputs, only plain-sight return values and parameters
- No hidden temporal coupling
- **Testable with less (no?) mocks**
- Free to call them any times →
- *Parallelizable*

```
r=f();  
a=g(r);
```

A **pure + fast** function is safe to call multiple times:

they typically are

Inline Variable



"Replace Temp Variable with Query"

**you don't care how many times
(and if) you call a pure function**



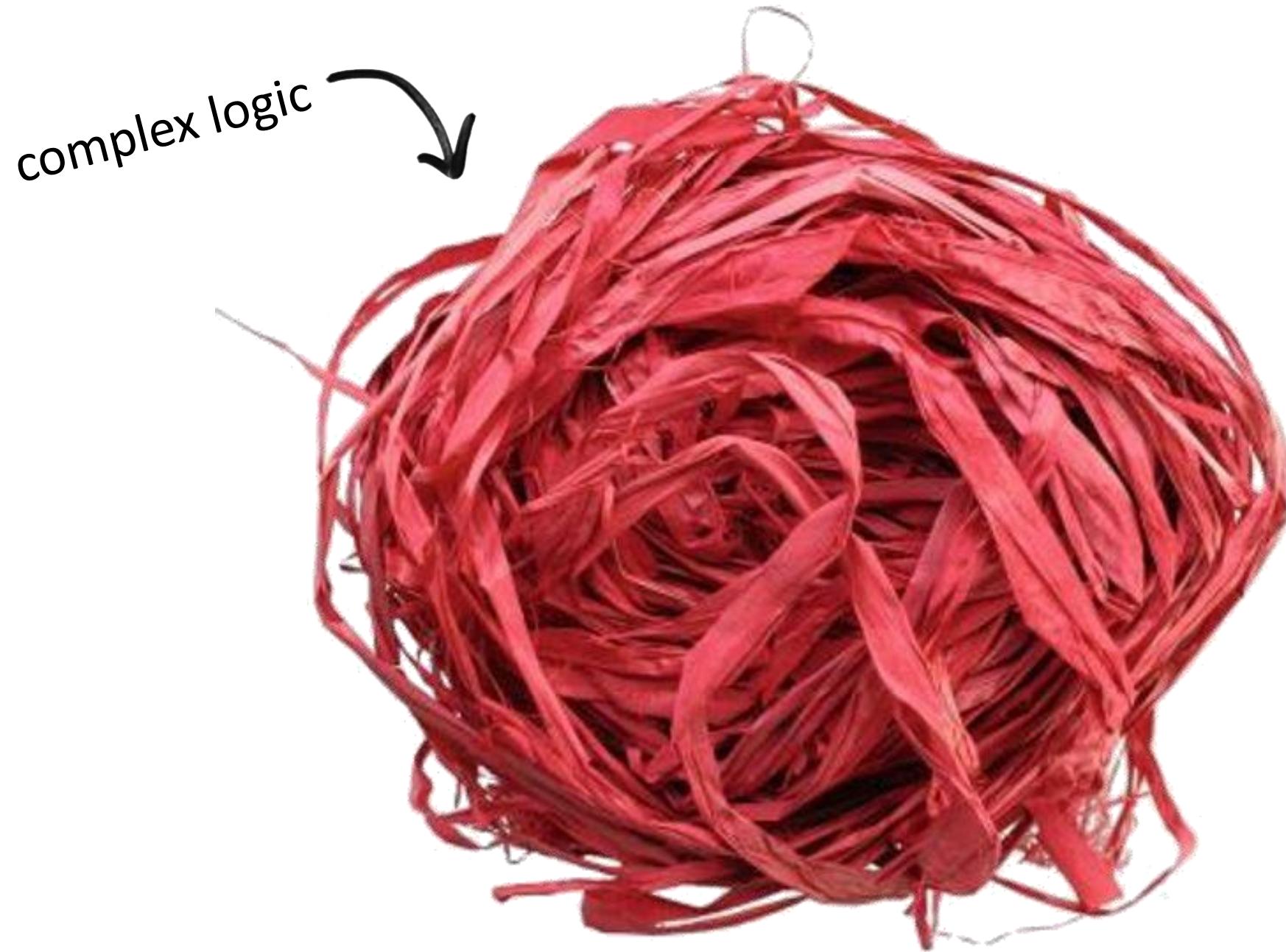


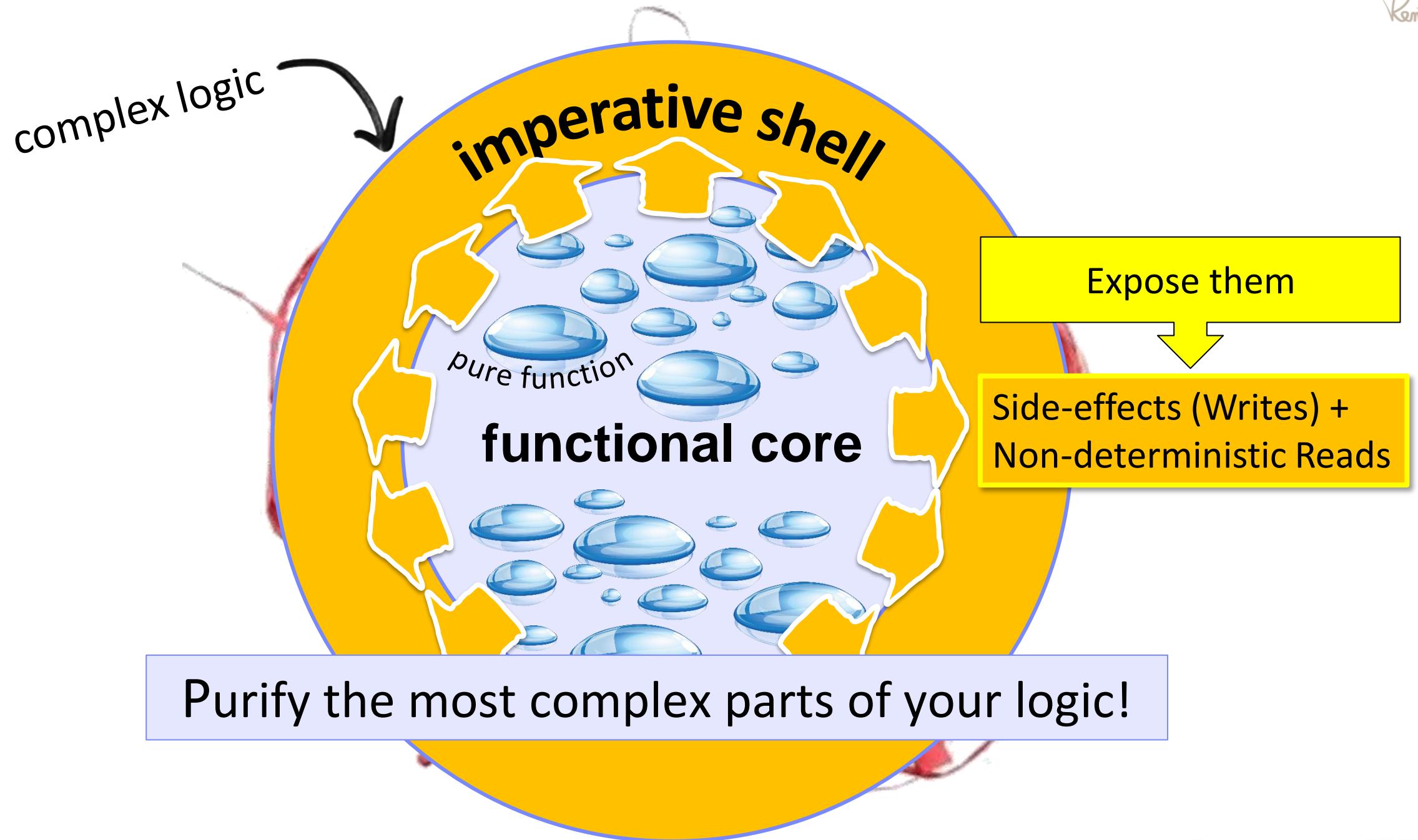
That's it!
I'll make all my functions pure



that's usually impossible

What kind of app doesn't change anything?





Purifying Core Logic:

Not typically causing problems

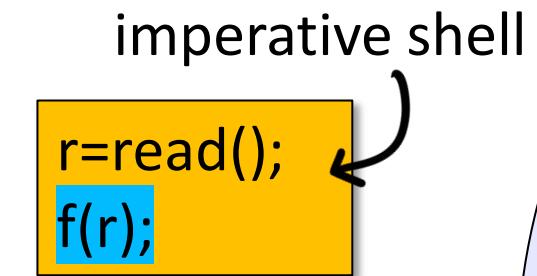
Time and Random

Files **Obvious**

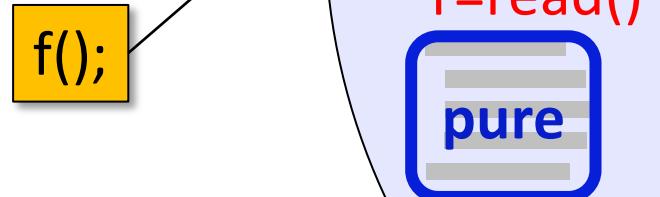
DB and API calls

Expose DB and HTTP calls

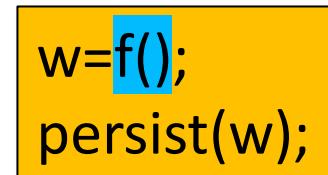
Initial Read → Pass as Parameters



Intermediary
(conditional?) → Split-Phase Refactor



Writing Results → Return Changes



Expose DB and HTTP calls

expose impurity to top-level

- Initial Read** → Pass as Parameters
- Intermediary**
(conditional?) → Split-Phase Refactor
- Writing Results** → Return Changes

imperative shell

```
r=read();
f(r);
```

```
r1=phase1(...)
```

r=read()

```
phase2(r,r1...)
```

```
w=f();
persist(w);
```

Create new classes 

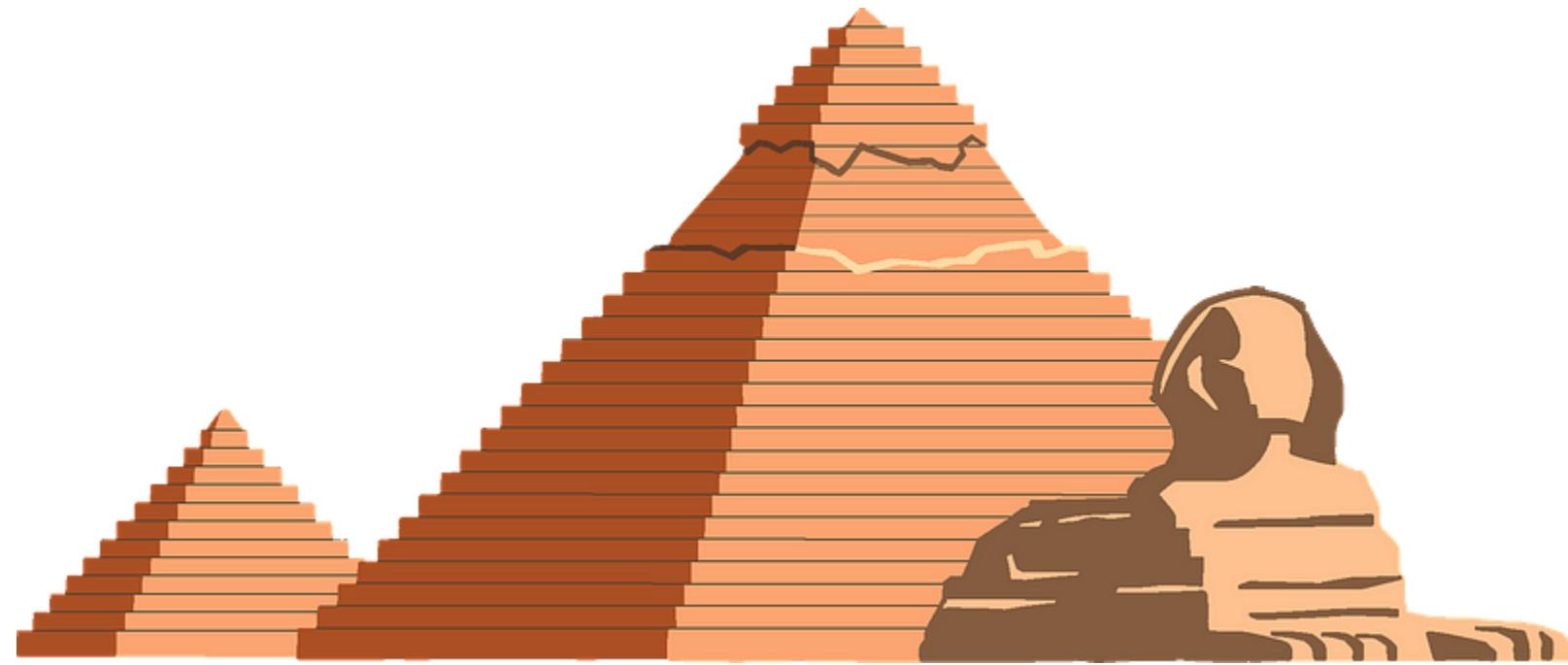
complex logic → pure functions

expose impurity to the surface

Pure Functions don't Change Objects' State

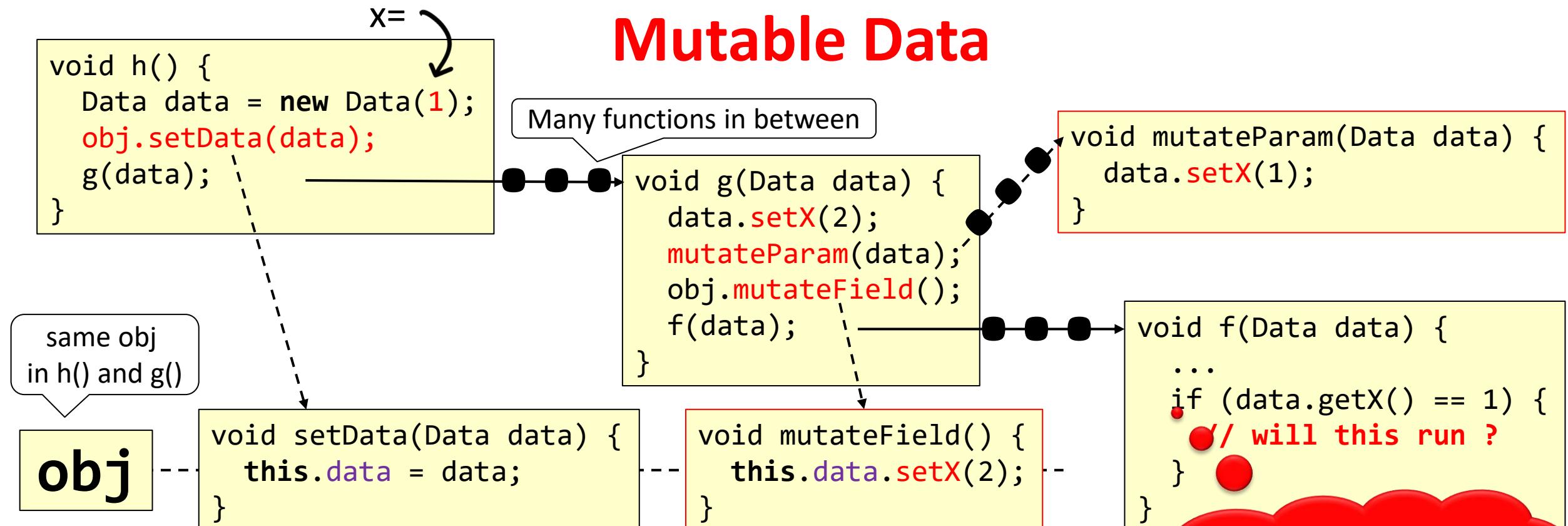


Immutable Objects



Immutable Objects

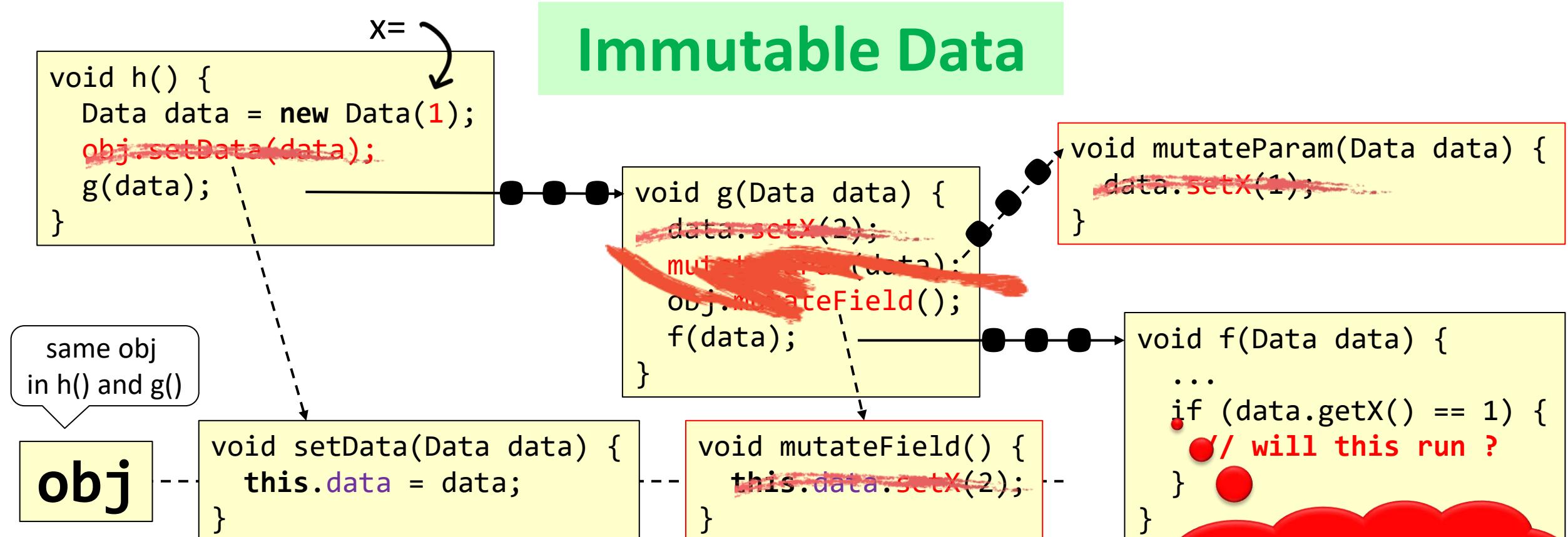
A Code Inspection Session



Long-lived mutable data

What code ran before,
having a reference
to my data instance?!

A Code Inspection Session



Long-lived mutable data

What code ran before,
having a reference
to my data instance?!

A Code Inspection Session

```
void h() {  
    Data data = new Data(1);  
  
    g(data);  
}
```

x= ↗

Immutable Data

```
void g(Data data) {  
  
    f(data);  
}
```

```
void f(Data data) {  
    ...  
    if (data.getX() == 1) {  
        // will this run?  
    }  
}
```

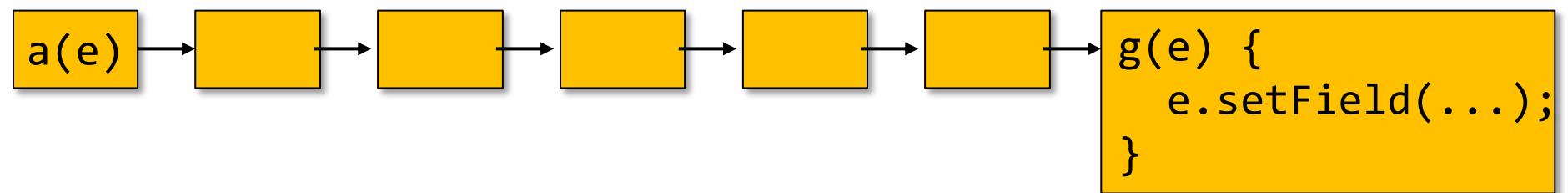
Easier to trace
data changes

Who created
the instance?!

The Big Deal

The Big Deal

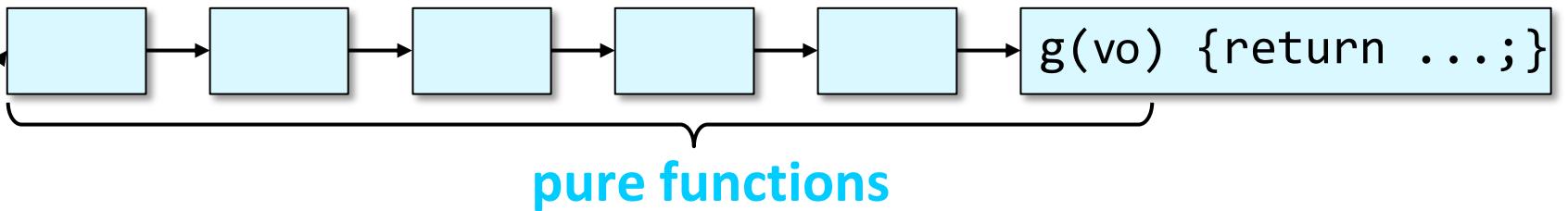
Don't mutate objects on long workflows!



Immutable Arguments

```
a(e) {  
    vo vo=e.extractVO();  
    String s = b(vo);  
    e.setField(s);  
}
```

Return the change, and apply it on the top level



pure functions

Return multiple changes:

1) vavr.Tuple3<String, String, Integer>

2) NewConceptVO

#kudos if you can find a good name!

Extends is dubious anyway

Designing Immutable Classes

final fields,
but not strictly required

final public class A {

private **final** String s;

private **final** B b;

private **final** List<String> list;

if B is immutable
you get true, deep immutability

public A(String s, B b, List<String> list) {

this.s = s;

this.b = b;

this.list = new ArrayList<>(list);

// validation ...

}

Defensive Copying: Stops caller to keep a reference

Overkill, as problem is not the creator but the "man-in-the-middle"

Self-validating data-model

public List<String> getList() {

return unmodifiableList(list);

}

// getters

// hashCode, equals on all fields = Value Object

// bits of LOGIC

public A withS(String newS) {

return new A(newS, b, list);

}

}

return:

Iterable<String>

ImmutableList<String> (guava)

List<? extends String>

record
(java 15)

or, until then...

@lombok.Value

@lombok.With

Java collections are mutable

So we CAN
change them!

"withers" return changed instances



A function changing an immutable object has to return it:

Bad Practice

```
data = updx(data);  
data = updy(data);  
data = updz(data);
```

... every time

Imagine "data"
has 20 fields



Who changed the field X?

The mess is still here!

How to fix?

final variables won't allow this = clutter; Extreme:

Mark only the non-final with @Var
([errorprone Java compiler from Google](#))

IntelliJ underlines
reassigned variables 



```
data = updx(data);  
data = updy(data);  
data = updz(data);
```

The Real Problem

Too Large Immutable Objects → break them

```
data.xyz = createXYZ(...);
```

*If they change together,
they stick together*



```
void f(VO[] arr) {  
    arr[0] = arr[0].withX(-99);  
}
```

```
void f(String str) {  
    if (str.length() > 0) {  
        str.substring(1);  
    }  
}
```

```
void f(Map<Integer, VO> map) {  
    map.put(1, map.get(1).withX(-99))  
}
```

DON'T
EVER
MUTATE
COLLECTIONS!



→ CREATE NEW ONES

State of the art solution:

Immutable Collections with Guava

```
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
</dependency>
```

ImmutableSet

```
ImmutableList.copyOf(list);
```

adding an element = cloning:

```
ImmutableList<Integer> newList =
<Integer>builder().addAll(numbers).add(2).build();
```



Let's Make all our Objects
Immutable!



overkill

Should Entities be immutable?

NO*

instead,

Extract immutable Value Objects from their fields

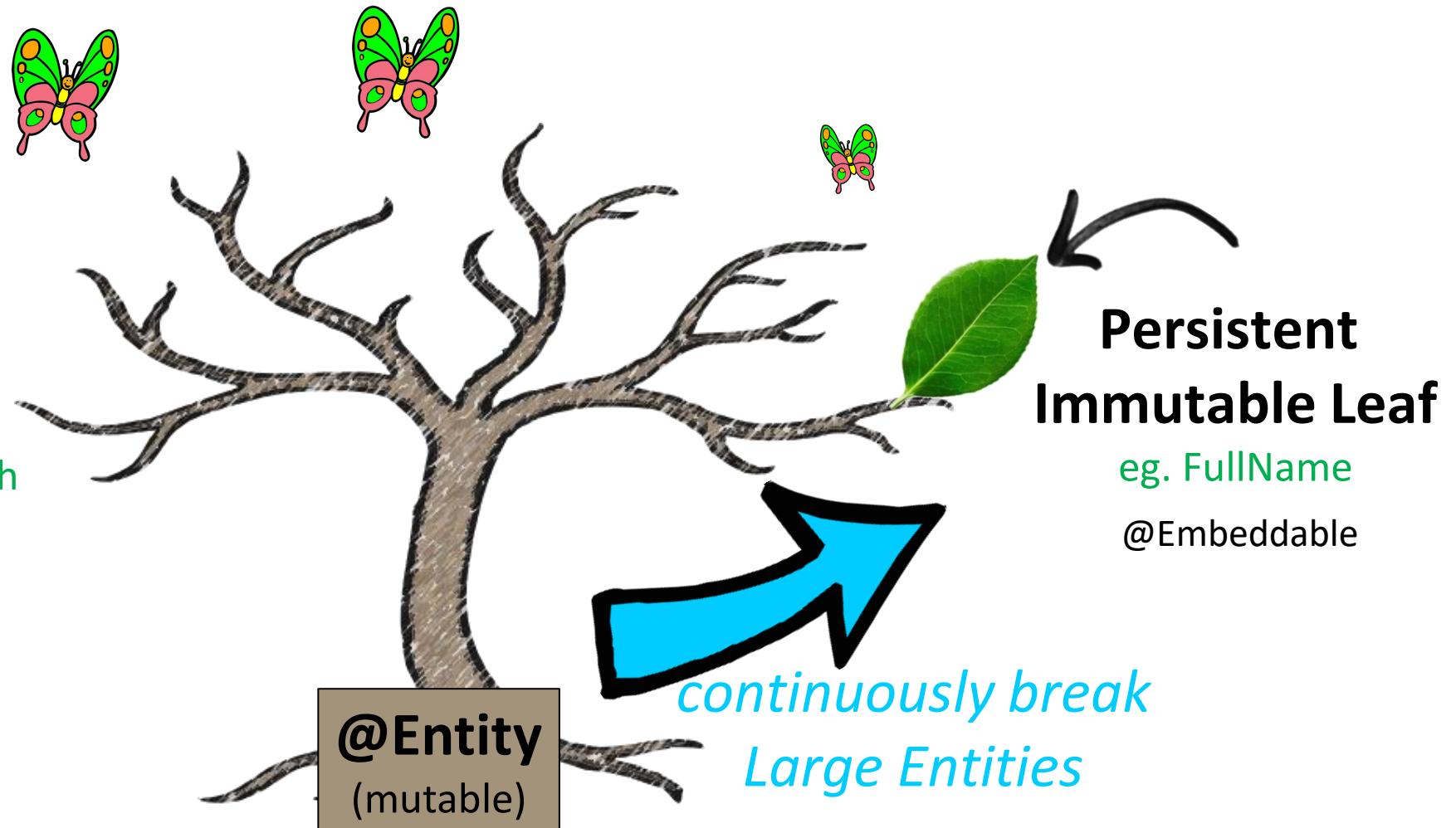
Leaving the root Entity mutable

Immutable Objects in Real Life



**Non-Persistent
Runtime Objects**

that you write heavy logic with



Identifying Value Objects

Conceptual Whole

eg Money

They Change Together

eg Discounts

Screens

eg InvoicingDetails

Input or Output of Complex Flows

eg PriceComputationInput

Always Immutable



**Non-Persistent
Runtime Objects**
that you write heavy logic with

The Big Deal

Is when immutable objects travel lots of code



Performance of Immutability

McDonnell

Performance:

Measure it !

(and you might have a surprise)

Avoid Immutable Objects If

- Trashing millions of instances/second
- Cloning Lots of Collections
- Root Persistent Entities or DTOs
- Trivial logic (overkill) • • •

When it grows complex:
Refactor to Immutability

Take-Aways

- Complex logic → pure functions using immutable objects
- Functional Core / Imperative Shell
- Pull impure remote/DB calls to top level
- ~~We'll change it in there~~ → compute and return
- Immutability is a mindset
- Don't mutate: argument state, variables or collections
- Make Immutable: runtime data or persistent leaves



Thank you!

→ ~~We'll change it in there~~ → compute and return

Return, Don't Change

Pure Functions vs Object-Oriented Programming

Ensure data is not changed
along complex computations
(heavy biz rules, hundreds of if-lines)

Guard state changing invariants,
whenever there's lots of state
(io, protocols, complex state machines)

Use to compute changes
in **functional core**

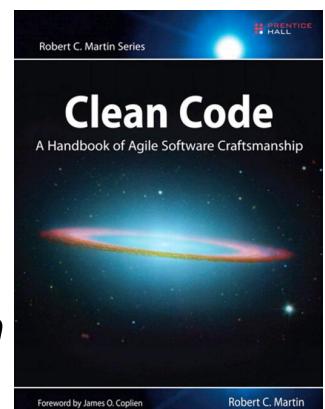
Use to encapsulate data
from **imperative shell**

So many guidelines!

*When I write functions, they come out **long** and **complicated**.
They have lots of **indenting** and **nested loops**.
They have **long argument lists**.
The **names** are arbitrary, and there is **duplicated** code.*

*I then massage and **refine** that code, **splitting** out functions,
changing **names**, eliminating **duplication**.*

*I don't write them **clean from the start**.
I don't think anyone could.* ~Uncle Bob, in



Continuous Refactoring

You are not done when all tests pass !



It is then when you should
Clean Up your code !!

(IDEs help you a lot)



The Hat Metaphor

- Kent Beck



Writing

Make it Work

Copy Pasta



Cleaning

Make it Readable

What can I delete?
How does my code read?



Unit Testing

Crack it

How can I break it?



MONSTER FUNCTIONS

- Mandatory: IntelliJ ☺
- ALT-ENTER on return → Move 'return' closer to computation
- ALT-ENTER all local variables → Move declaration closer to usages
- ALT-ENTER anything colorful (grayed out, yellow, underlined) → Think!
- Introduce Parameter Object, creating or reusing Value Objects
- underlinedVar is reassigned = abused variable → Define more variables; CAREFUL with leaking data
 - case: {
- Flip ifs to define early returns (guard clauses) as early as possible → Reduce 'depth' of function
- Break IFs, and FORs to do a single thing (SRP) → Extract Method
 - Careful, with as much ALT-ENTER as possible
- (try) to section the function at indentation-0 places → Create new classes to pass between
- If using lots of instance fields, Tentatively make it static, to highlight them; instance state = to be improved
- (last resort) Make it a class → Extract Method Object, taking its private satellite methods



Advices for Refactoring Legacy Code

Unknown

You should not understand everything first

Don't fear the lack of tests: use safe tiny steps

Explore the code

(tentatively break compilation/timeboxed exploration)

Be Humble, always ready to CTRL-Z to safety

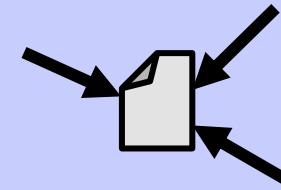
Don't solve it all

Use a notebook to keep you from creeping out

Safest way to unify similar code

0. Move to one file

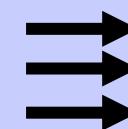
all the similar methods



Introduce Parameters to Replace More Duplicates

1. Extract Variable(s)

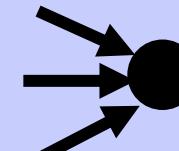
to make the syntax identical



**SUPER
SAFE**

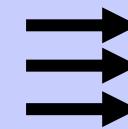
2. Extract Method

The IDE detects duplicates



3. Inline Variable(s)

in the method calls



WARNING

Don't couple unrelated code

How to refactor God Methods of 100-1500 lines ?



A class should have less than 200-300 lines



Extract Method Object refactor

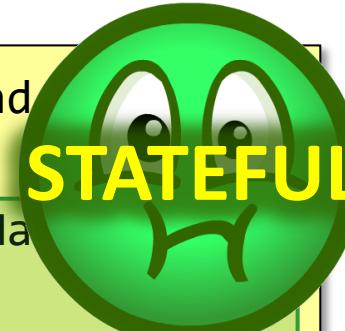
You make it a class

Extract Method Object refactor

```
public String doComplexStuff(String text) {
    Set<String> words=new HashSet<>();
    int totalWords = 0;
    int totalPhrases = 0;
    StringTokenizer tokenizer = ...
    while (tokenizer.hasMoreTokens()) {
        String next = tokenizer.nextToken(); ...
        if (...) {
            for (...) {...}
            ...
        } else {
            ...
        }
    }
} // line 534
```

Extracting a method may require many parameters, output parameters or be even impossible

```
String result = doComplexStuff("bla");
```



```
public class DoComplexStuffCommand {
    private final String text;
    private Set<String> words=new HashSet<>();
    private int totalWords = 0;
    private int totalPhrases = 0;
    private StringTokenizer tokenizer;
}

public ExtractedMethodCommand(String t) {
    this.text = t;
    tokenizer=new StringTokenizer(text, " ");
}

public String execute() {...}
```

These methods use instance fields, and take no/fewer arguments

```
= new DoComplexStuffCommand("bla").execute();
```

then, GC the instance

Clean Functions

Shorter

Many private and flat

Expressive names

Parameters: ≤ 3, no booleans, no nullable, default params

Pure if complex

Throw early or return Optional<>, not null

Agenda

Introduction

Names

Functions

Classes

Formatting & Comments

OOP 900



Data structures

=*classes that expose all their state*

```
public class ImmutableStruct {  
    private final String firstName;  
    private final String lastName;  
  
    public ImmutableStruct(  
        String first, String last) {  
        this.firstName = first;  
        this.lastName = last;  
        // validation ...  
    }  
  
    public String getFirstName() {  
        return firstName;  
    }  
    public String getLastName() {  
        return lastName;  
    }  
}
```

≈ Value Object

We ❤️ Immutable Objects:

- If created valid, remain so
- Easier to debug
- Thread safe
- Safe to put in Tree*/Hash*

Data structures

=classes that expose all their state

```
public class ImmutableStruct {
    private final String firstName;
    private final String lastName;

    public ImmutableStruct(
        String first, String last) {
        this.firstName = first;
        this.lastName = last;
        // validation ...
    }

    public String getFirstName() {
        return firstName;
    }
    public String getLastName() {
        return lastName;
    }
}
```



```
public class PhoneBookEntryDTO {
    public String getFirstName();
    public String getLastName();
    public String getPhoneNumber();
}

public class PhoneBookEn {
    private String firstName;
    private String lastName;
    private String phoneNumber;

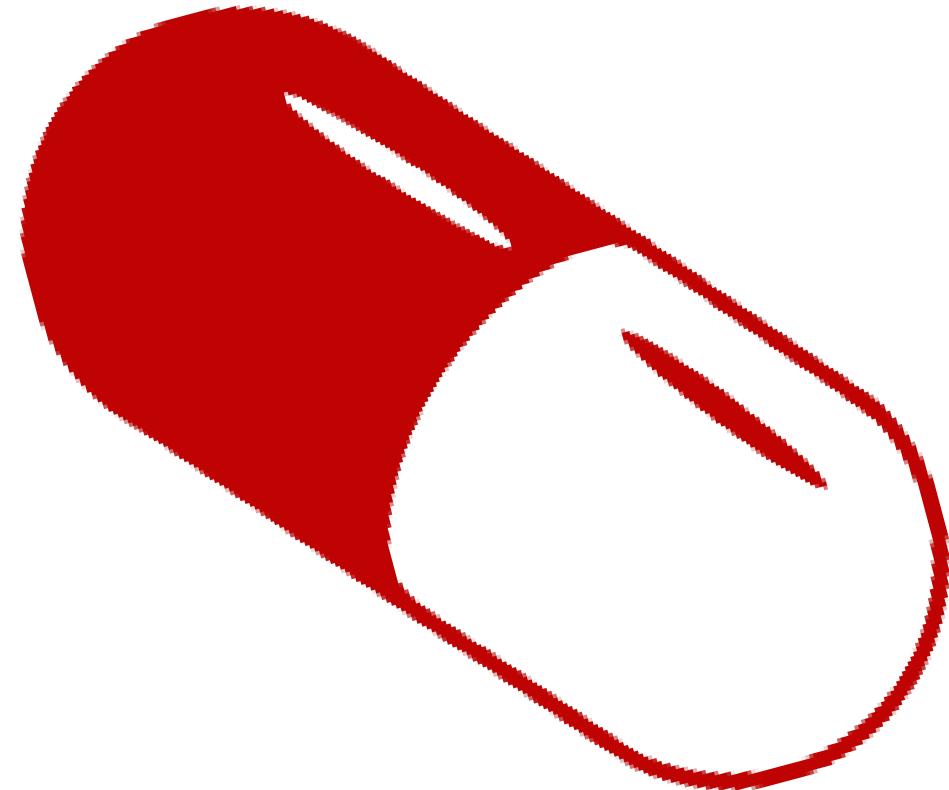
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String first) {
        this.firstName = first;
    }

    public String getLastName() {
        return lastName;
    }
    public void setLastName(String last) {
        this.lastName = last;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }
    public void setPhoneNumber(String phone) {
        this.phoneNumber = phone;
    }
}
```

"Object-Oriented Assembly Language"

This is NOT Encapsulation !



OOP

Expose Behavior, not data

Data is an implementation detail

~~car.engineStarted = true~~

~~car.setEngineStarted(true)~~

car.startEngine()



Information Hiding

Even when asked

~~car.getGasolineInLiters()~~

~~car.getPercentageFuelLeft()~~

car.getEstimatedRemainingKm()



Implementation can evolve
without breaking your clients

MINIMIZE THE API



Protect client code from future changes in implem of a
Library or (mini) Framework

mycorp-commons-**1.0**.jar

API

Decompose complex logic into separate objects

Divide-et-Impera on Complexity

CsvWriter



But at work

we don't do much OOP, do we ?

In our Enterprise Apps, ↪ despite the interview

we write procedural code

WHY ?!

(usually)

We automate existing business processes.
Existing procedures.



Procedural Code

Procedural Code

Lots and lots and lots and lots and lots of it...

How do we
Keep It Short & Simple ?

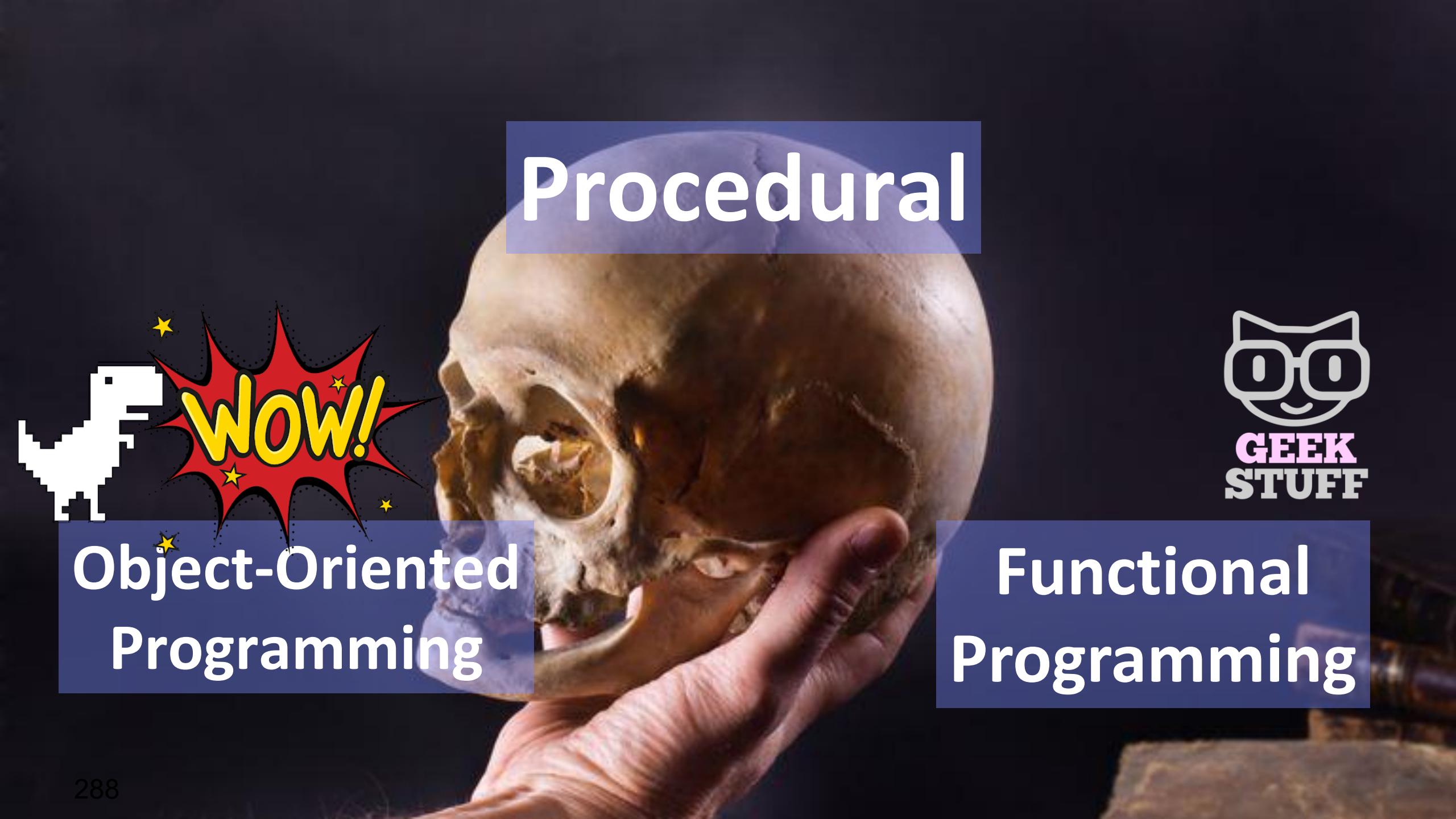


Procedural Code Keep It Short & Simple !!

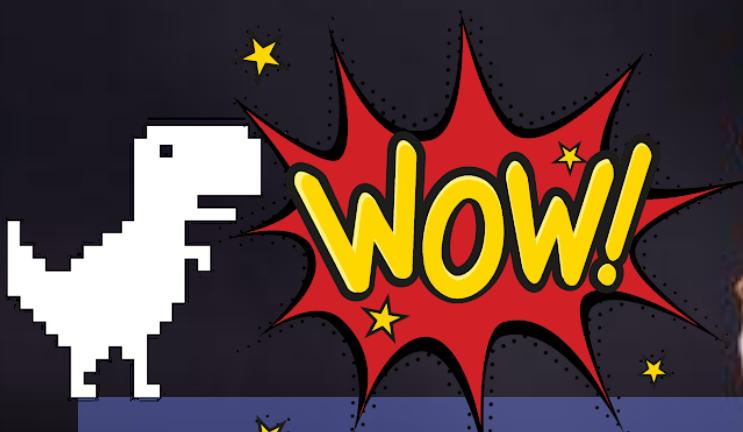
We distribute logic in many classes:



Classes as containers of logic
(in enterprise applications)



Procedural

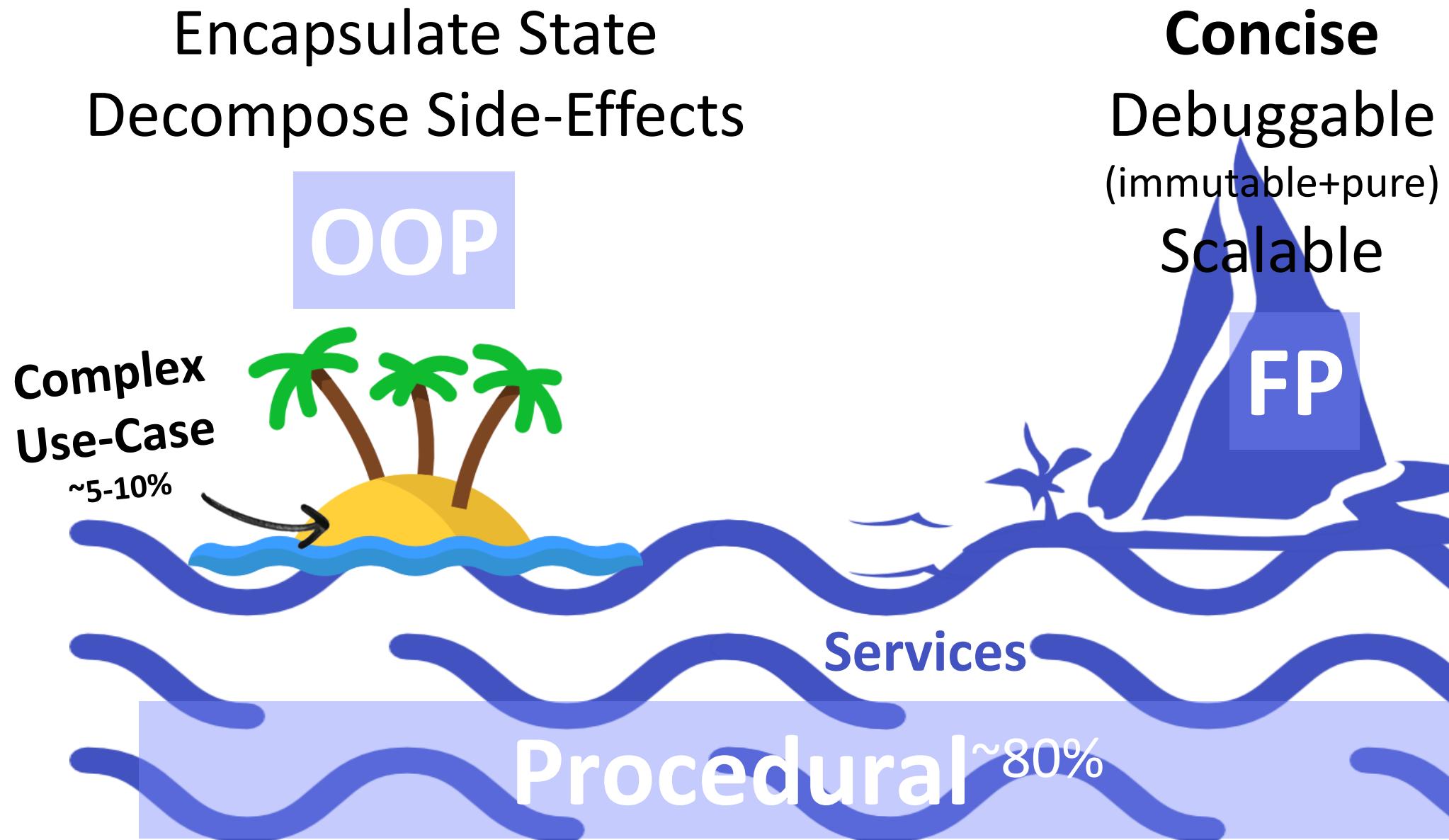
 **WOW!**

Object-Oriented Programming

Functional Programming







Class Types

- **Data Structures** (best if immutable): FullName, Interval
 - Decompose larger structures in immutable leaves
- **Objects** encapsulating state and constraints: CsvWriter
 - Stable API for Libraries
 - Hiding Complexity in Horror Use Cases
- **Bags of Functions**: OrderService
 - Better: PlaceOrderService

Agenda

 Introduction

 Names

 Functions

 Classes

 Formatting & Comments

Hello, World!

(=<`#9]~6ZY32Vx/4Rs+0No-&Jk)"Fh}|Bcy?`=*z]Kw%oG4UUS0/@-ejc(:'8dc

[Malbolge](https://en.wikipedia.org/wiki/Malbolge)

[Brainfuck](https://en.wikipedia.org/wiki/Brainfuck)

Ε ΓΟΕΕ Μ Σ Ρ Ε Δ Γ Α Φ Χ Ε ΓΟΕΕ
Χ Α Ι Α Φ Δ Σ Ω Γ Ι Σ 5 Α Σ Ι Ω Σ Ε Χ Α Ι Α Φ Δ Σ
Ε Ο Χ Κ Ρ Δ Ε 5 Μ Ι Σ Ψ Ι Σ Α Γ Α Ε Ο Χ Κ Ρ Δ Ε 5
Α Α Ε Δ Ι Χ Ι Ψ Σ Ψ Ι Ι 5 Ε Ψ Μ Θ Α Α Ε Δ Ι Ι Ψ
Σ Φ Α Κ Χ 5 Ρ Ψ Ε Ψ Σ Σ Ψ Ψ Δ Σ Α Ο Φ Α Κ Χ 5 Ρ Ψ
Α Δ Ο Ε Ε Ψ Ι Σ Ψ Σ Σ Β Ι Σ Α Ε Ψ Α Δ Ο Ε Ε Ψ Ι
Σ Ψ Α Σ Ι Ι Ο Ε Ε Ψ Σ Σ Α Φ Χ 5 Σ Ψ Σ Ι Ι Π Ο Ε
5 Ψ Σ Π Δ Δ Σ Α Φ Α Φ Δ Ε Σ Ε Ψ 5 Ψ Σ Π Δ Δ Σ Α Φ
Ψ Σ 5 Ι Σ Β Ι Σ Ε Ψ Σ Σ Α Γ Α Φ Ψ Σ 5 Ι Σ Β Ι Σ Ε Ψ
Γ Ψ Σ Σ Ε Σ Ε Ψ Σ Σ Α Γ Α Φ Ψ Σ 5 Ι Σ Β Ι Σ Ε Ψ
Σ Ι Ψ Ε Ι Σ Ε Ψ Σ Σ Α Γ Α Φ Ψ Σ 5 Ι Σ Β Ι Σ Ε Ψ
Γ Σ Σ Ψ Ψ Ι Ι Δ Ι Α Ε Ι Χ Ι Ρ Ψ Ι Ι Ω Σ Ψ Π Ψ Σ Σ
Σ Ι Ψ Ε Ι Σ Ε Ψ Σ Σ Α Γ Σ Ι Ι Δ Σ Ψ Α Γ Σ Ι Ι Ψ Ε Ι
Γ Σ Σ Ψ Ψ Ι Ι Ι Ε Δ Ε Α Σ Ψ Ξ Ε Ψ Ι Ι Η Γ Σ Σ Ψ Ψ
Σ Ι Ψ Ε Ι Σ Ε Ψ Σ Σ Α Γ Σ Ι Ι Δ Σ Ψ Δ Σ Ι Ι Ψ Ε Ι
Ω Ε Γ Ρ Φ Ξ Α Σ Α Θ Ε Δ Ι Ψ Δ Σ Ι Ι Ψ Ε Ι Ι Ψ Ε
Σ Ι Ψ Μ Σ Σ Ε Ψ Ι Ι Σ Α Σ Α Σ Σ Σ Σ Σ Σ Σ Σ Σ Σ

The image consists of a large, dense grid of Greek characters, primarily alpha (Α), beta (Β), gamma (Γ), delta (Δ), epsilon (Ε), zeta (Ζ), eta (Η), and iota (Ι). These characters are arranged in a grid pattern and are colored in black, green, and red. The background is white, and the overall effect is a digital or abstract representation of ancient Greek script.

You're not Neo
It's all about Team Work



WAKE UP!



**Don't Code!
Communicate !**



Code Review

Think about effort + Δ_{better}
+ does it really matter?

Ahem...
Hey, Victor, let's talk about
the code you pushed!

Never
← make it or
take it →
personal



The Best Code Review

Victor Rentea

www.victorrenteal.ro

CONF42

<https://victorrenteal.ro/blog/the-best-code-review/>

Victor Rentea

Java Champion

♥ Refactoring, Simple Design, Unit Testing ♥

Bucharest **Software Craftsmanship** Community

Join us on [meetup](#)

VictorRentea.ro

Best Talks, Blog, Training

Independent Trainer

INTENSE Technical Training

8 years

2000 devs

50 companies

400 days
(100+ online)

Spring 

Hibernate

Functional Prog

Design Patterns

Clean Code

Unit Testing

any lang

Reactive Prog

Java Performance

Training for you or your company:
VictorRentea.ro

Posting
Good Stuff on:
  
@victorrentea





Hey man,
can you come a sec
for a Code Review?



The Best Code Review



Have a seat!





How about this?

Ok.

(typing...)
What do you think
about this ?

(typing....)

Yeah, that's
a great idea!

Neah...
I'm not convinced

Oh yeah!
Just PUSH it and I'll finish it
in a sec



> revert

A portrait of Mike Tyson, a bald man with a grey beard and mustache. He has intricate black tribal markings on his forehead and around his eyes. He is wearing a dark t-shirt. A light blue speech bubble is positioned to the left of his head, containing the text "Now YOU do it !".

Now YOU do it !

Windows

A fatal exception 0E has occurred at 0028:C0034B23. The current application will be terminated.

- * Press any key to terminate the current application.
- * Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue _



Returning to your desk to DIY

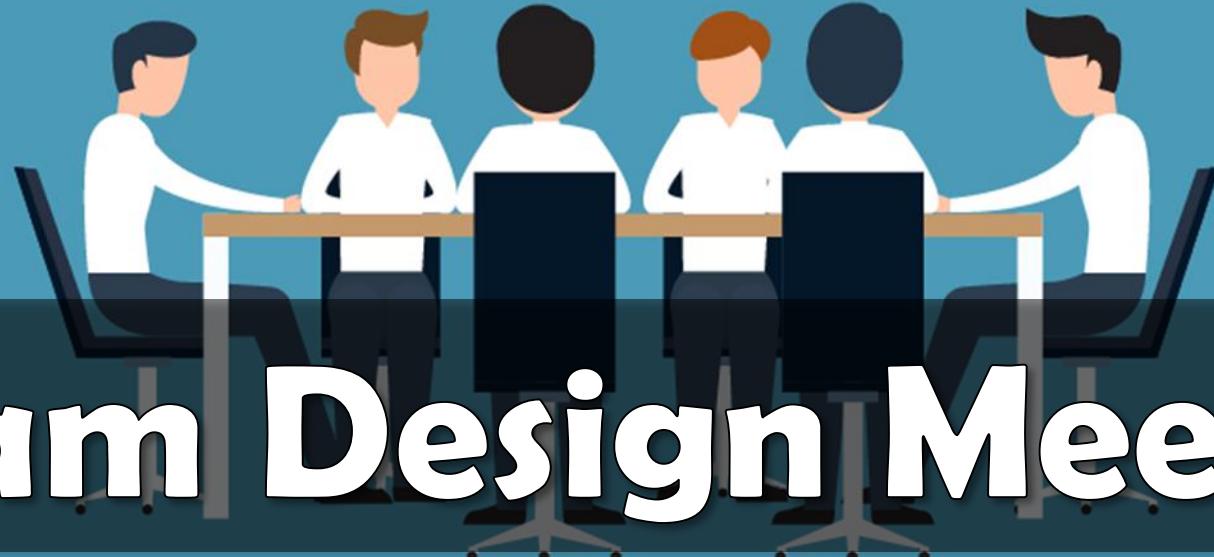


**Convert any Complex Code Review
in a Pair Programming session**

Use Video for Complex Reviews



Recurring Topics



Team Design Meeting



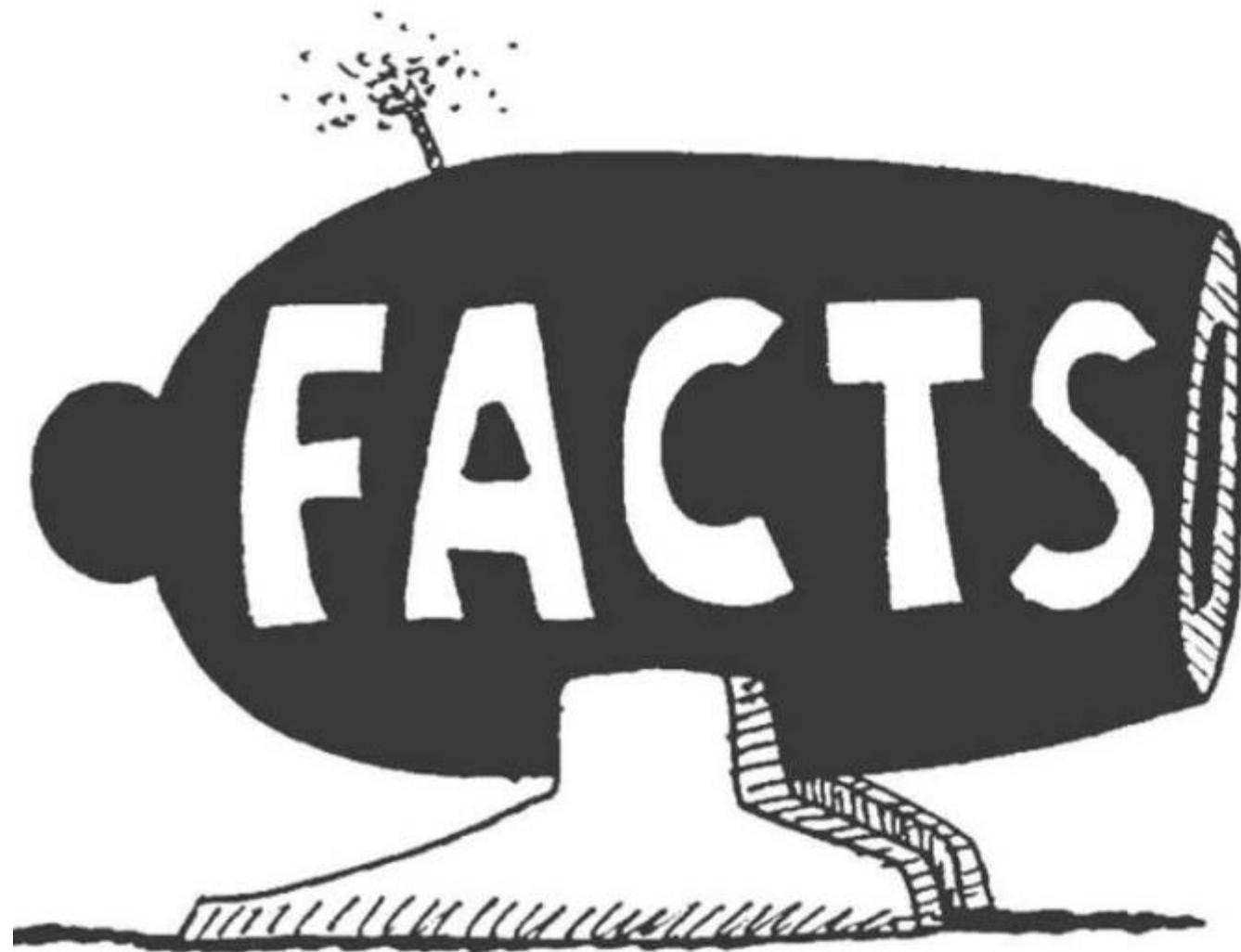
Terrible Issues?

Second

chance

A photograph showing a teacher with glasses and a red shirt smiling while guiding two students, a boy and a girl, who are writing in notebooks. The teacher is leaning over them, providing support.

Guiding Review



Practical Tips for Reviewee

Work Hard

Until you're ***proud*** of your Code

Run SonarLint before commit

Ask for ***in-flight reviews***

Intentional ***micro-commits***

Commit Massive Automatic Refactoring separately

Sketch a ***supporting diagram***

Always offer to provide a walkthrough to the reviewer

Code Review Tips

Be kind and humble, on both sides
Never make it / take it personal

Constructive Attitude
Focus on understanding & learning

Facts vs Opinions
Imperfect code might be better

Pair Programming
is better for **complex stuff**

The Best Code Review

Victor Rentea

www.victorrentea.ro

CONF42

<https://victorrentea.ro/blog/the-best-code-review/>

DRIVER

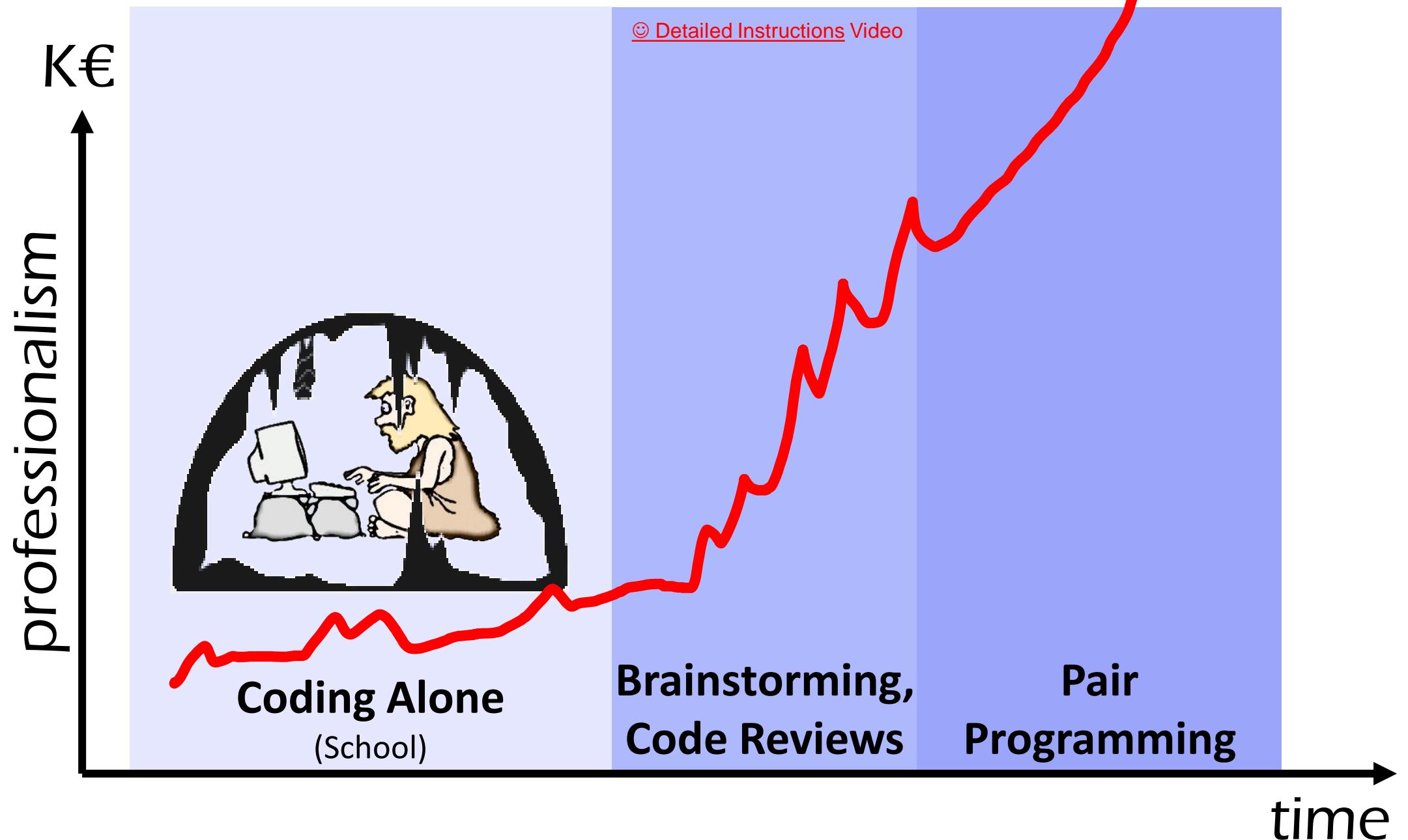
Pair Programming

NAVIGATOR

A photograph of two programmers, a man and a woman, working together at a computer. They are both looking at the screen, which displays a complex software interface with multiple windows and code snippets. The man is on the left, wearing a blue shirt, and the woman is on the right, wearing a grey shirt. A large black arrow points from the text on the left towards the text on the right.

fun++
less bugs
better design
natural learning
team building

faster + cheaper
on the long run



[© Detailed Instructions Video](#)

Don't Code! Communicate !

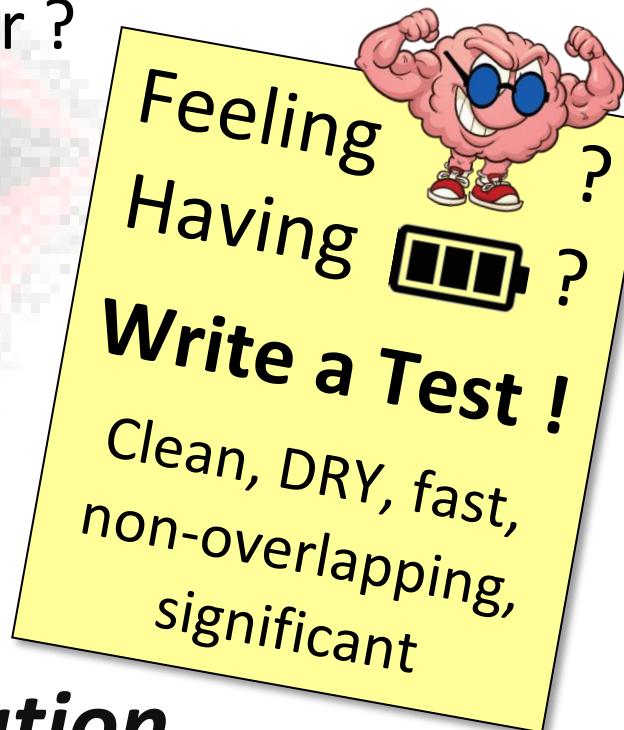


Respect your readers

Details matter: 10x more reading, remember ?

children
✓
Write Literature

The simplest code that works.
Never obfuscate



Simplicity is the ultimate sophistication

- Leonardo da Vinci

You got your Driver License

Your First Day Driving...

without the instructor next to you





Turning Left for the first time



Turnin time



Then your GF starts the radio:

A photograph of a man with dark hair and a beard, wearing a white shirt and tie, driving a car. He has a wide-eyed, shocked expression with his mouth open. His hands are firmly gripping the steering wheel. The background shows the interior of the car and some blurred trees outside, suggesting motion. Overlaid on the left side of the image is large, bold, yellow text.

You explode!

10 years later...



Master your IDE

Learn those **keys!**



**Key Promoter X plugin
will help you a lot !**

Sharpen reflexes for real production-code fights

| Refactoring: Ctrl-Alt-Shift-T | |
|-------------------------------|---|
| Shift-F6 | Rename |
| Ctrl-Alt-M | Extract Method |
| -V | ... Local Variable |
| -F / -C | ... Field / Constant |
| -N | Inline |
| Ctrl-F6 | Method Signature |
| F6 | Move: file, method |
| Alt-Enter | Quick-fix: error, warn, typ, gray, refactor |

| Editing | |
|--------------------------|------------------|
| Ctrl-Y | Delete Line |
| Ctrl-D | Duplicate Text |
| Ctrl-[Shift]-W | Grow Selection |
| Alt-Shift-↑ | Shift Lines up |
| Ctrl-Shift-↑ | Shift Block up |
| Alt-J | Multi-cursor |
| Ctrl-Space ^{x2} | Static functions |

| Navigation: Ctrl-<click> | |
|--------------------------|-----------------|
| Alt-F7 | Find Usages |
| Shift-Shift | Find File |
| Ctrl-Shift-F | Find Text |
| Ctrl-H | Type Hierarchy |
| F12 | Outline |
| Ctrl-Alt-H | Call Hierarchy |
| Ctrl-Shift-A | Search Anything |

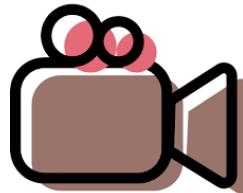


Missing a key? Are you on Mac or Eclipse?

Download editable PPT [here](#)

More Advanced Keys

IntelliJ Productivity Tips:



by Victor Rentea
+ IntelliJ Lead @JetBrains,
at jPoint, Moscow 2021

Learn to move fast
so you aren't sorry to Ctrl-Z

So you can experiment
and find the simplest design

Ctrl-Shift-Enter – Complete Statement

Ctrl-Alt-T – Surround with ...

Ctrl-Alt-Shift-T – Refactor this (cursor/selection)

Ctrl-Shift-Space – Type-aware auto-complete

Ctrl-Space x 2 – Autocomplete static methods

“gr” -> generates getter getResult() { return ... }

Ctrl-E

// language=sqll --> [IntelliJ on DB](#)

error *warn* *typo* *unused*

When it's red, yellow, blue or gray,

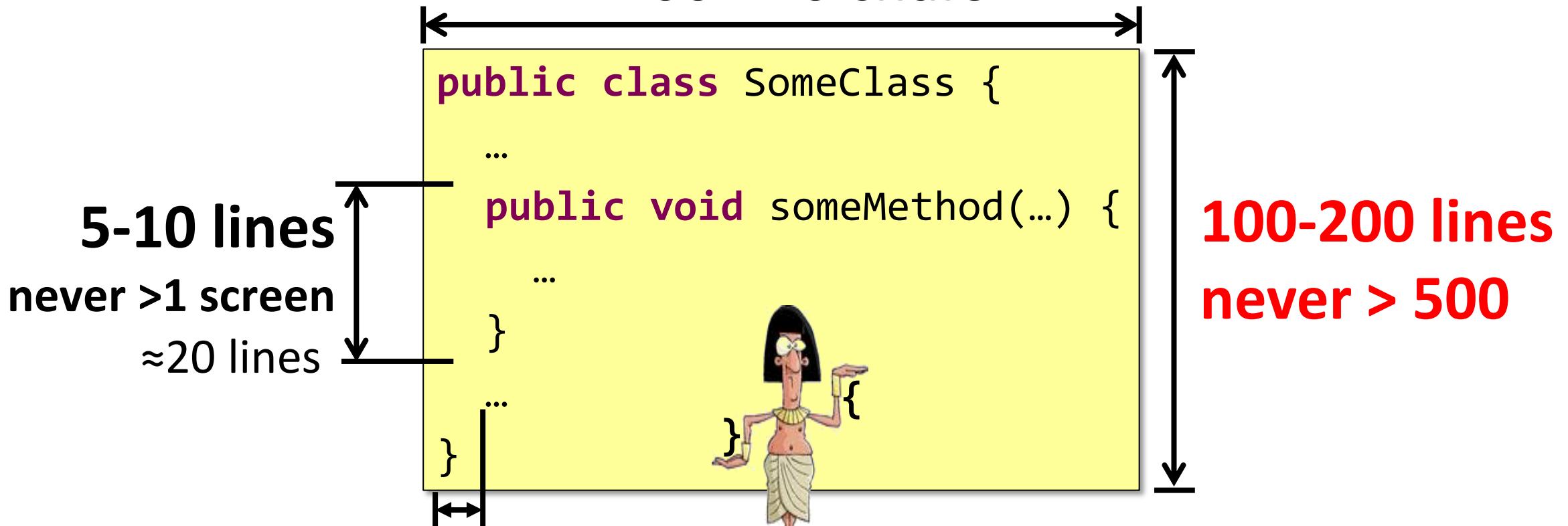


Alt-Enter will save your day.



Suggestions...

100-120 chars



2-3-4 chars / TAB?



Spaces, code style, ...

Team **Rulez**
~~format => ownership~~

Comments = Failures

Written proof of incompetence



Always before dropping a comment, and
Try to express it in code

Why do we hate comments ?

Comments

They inevitably fall out of sync with the code

/ very important comment from 10 years ago that tells that... */*

Comments



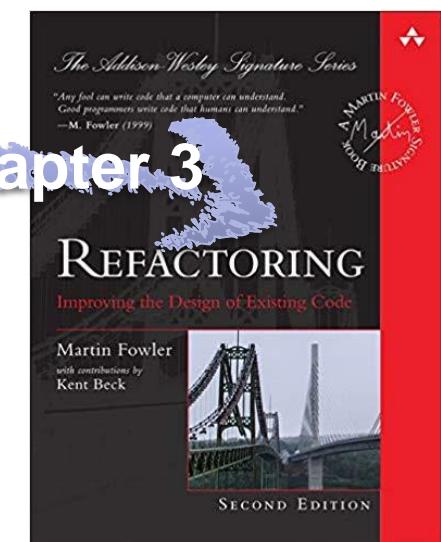
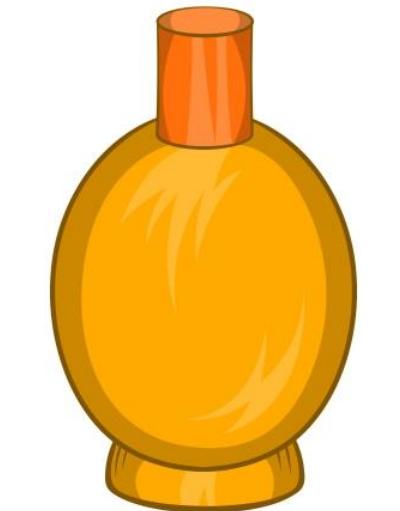
We aren't saying that people shouldn't write comments.

In our olfactory analogy, comments aren't a bad smell; indeed they are a sweet smell.

The reason we mention comments in Code Smells is that comments are often used

as a deodorant

- Refactoring 2nd



**The code should
Speak for itself**

Read This

```
public List<int[]> getCells() {  
    List<int[]> list1 = new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4) list1.add(x);  
    return list1;  
}
```

■ Readable Constants

- Instead of magic numbers

```
public List<int[]> getCells() {  
    List<int[]> list1 = new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4) list1.add(x);  
    return list1;  
}
```

cell[**STATUS**] == **FLAGGED**

■ Readable Constants

- Instead of magic numbers

■ Explanatory Variables

- Name intermediary results

83 **if** (multiline &&
 84 (conditions ||
 85 formulas))

```
public List<int[]> getCells() {
    List<int[]> list1 = new ArrayList<int[]>();
    for (int[] x : theList)
        if (x[0] == 4) list1.add(x);
    return list1;
}
```

```
public List<int[]> getFlaggedCells(){
    List<int[]> flaggedCells = new A...L...<...>();
    for (int[] cell : gameBoard) {
        boolean isFlagged= cell[STATUS]==FLAGGED;
        if (isFlagged)
            flaggedCells.add(cell);
    }
    return flaggedCells;
}
```

■ Readable Constants

- Instead of magic numbers

■ Explanatory Variables

- Name intermediary results

83 **if** (multiline &
 84 conditions ||
 85 formulas)

■ Explanatory Methods

- "Encapsulate Conditionals"

■ Rename

```
return gameBoard.stream().filter(Cell::isFlagged).collect(
```

■ Extract Methods



```
public List<int[]> getCells() {  

    List<int[]> list1 = new ArrayList<int[]>();  

    for (int[] x : theList)  

        if (x[0] == 4) list1.add(x);  

    return list1;  

}
```

```
public List<Cell> getFlaggedCells(){  

    List<Cell> flaggedCells = new A...L.<Cell>();  

    for (Cell cell : gameBoard){  

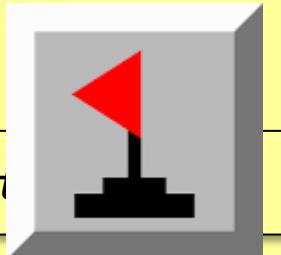
        if (cell.isFlagged())  

            flaggedCells.add(cell);  

    }  

}
```

Live Documentation



Constants

1. Multiple usages of the same business value

```
public static final String SPACE = " ";
```

DELIMITER

2. Even a single literal, to understand logic

```
if (isInternational) {  
    price += 20;  
}
```

Duh!!

~~EXTRA FEE FOR TNTL SHIPPING~~

My Point?

If a literal appears in a single place,

And naming it doesn't make the code easier to understand,

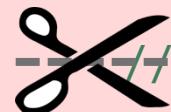
Perhaps don't extract a constant!

Bad Comments

Mumbling

- Unclear!
- Rush/lazy?
- Forgotten?

```
stream.close();
} catch (IOException e) {
    // Give me a break!
}
```



Position Markers

----- Chapter 2 -----

Redundant

```
/** Returns the day of the month
 * @return the day of the month
 */
public int getDayOfMonth() {
    return dayOfMonth;
}

/** Default constructor. */

```



"UCLOC"
- the '90s

Commented-out Code

DELETE IT! (Git rocks!)

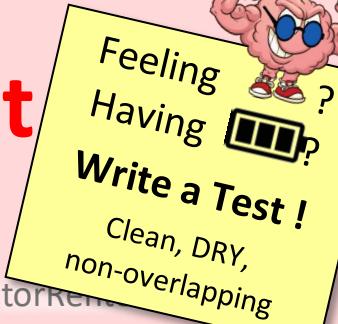


Non-Local

Comments far from the code

Over-involvement

Wiki article syndrome

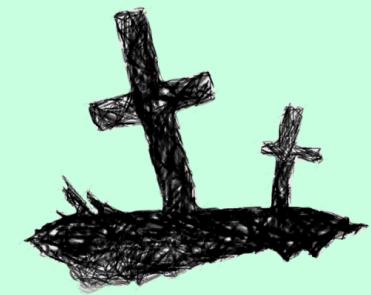




~~WHAT?~~
~~HOW?~~

Good Comments

When the code just can't say it



Algorithms

// http://wiki.../Dijkstra's_algorithm

Warnings

2 DAYS
LOST HERE

//Don't try optimization X, leaks memory

Bug Workarounds

// http://jira.../bug-1252

Performance Tweaks

// int[] occupies less memory

Weird Business Rules

// http://confluence.../1252

TODOs

// TODO vrenteа 2005-03-02 Remove on Mon

~~Survives time and refactoring~~

Foreign API/Format

dto.setFName(name); // fullName!

Library JavaDocs

Keep other devs out of implementation

Opinions?

~~// Register user~~



extract registerUser()

~~// Loop over all entries and remove nulls~~

```
list.removeIf(v -> v == null);
```

~~// *****~~

// Avoids memory leak

~~// *****~~

Duh!!
(don't underestimate
your reader)



∞ chains



```
public static String getFirstUserName(ApiResponse apiResponse) {
    return Optional.ofNullable(apiResponse)
        .map(ApiResponse::getInformations)
        .map(Informations::getAllUsers)
        .map(AllUsers::getAdminOnly)
        .map(AdminOnly::getWithPassword)
        .map(WithPassword::getUsers)
        .map(List::stream)
        .map(stream -> stream.filter(i -> !i.isDisabled()))
        .map(stream -> stream.sorted(Comparator.comparing(User::getCreatedAt)))
        .map(Stream::findFirst)
        .map(Optional::get)
        .map(User::getUsername)
        .orElse("Not Found");
}
```

Break into explanatory
variables/methods

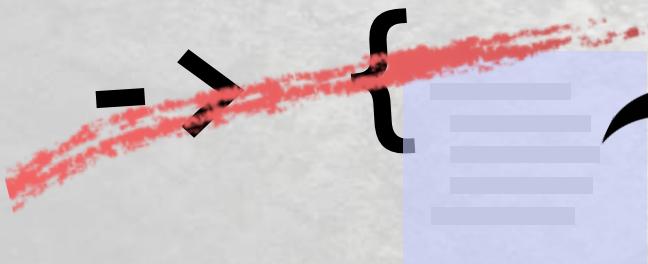
Clean Code in Java8

One-line lambdas only!

x->f(x,y) or ::namedFunction

Avoid .forEach(() -> Logic with Side Effects)

Prefer .collect() or classic for (e : list)



'-> {} - 0 matches in workspace

extract namedFunction()

Keep it Decent
intermediary collect()

Have a library

jOOL, vavr, ...

Unchecked.*

Try<R>
.zip()

Enrich your data objects
With little helper functions, eg. isActive()

Judicious Optional<>



Agenda

- Introduction
- Names
- Functions
- Classes
- Formatting & Comments
- Java 8+

Key Points

- Stop **Refactor** = Start **Legacy**
- Refine **Expressive Names**
- Short methods
- Structs, Objects or Logic Containers ?
- Replace Comments with Expressive code
- **Pair Programming** is the way

**What can I do
to make refactor happen
each day
in my team?**

HOW TO
APPLY ALL
THIS IN MY
LEGACY
CODE
??

LET'S
PRACTICE
!!!

WHERE
CAN I READ
MORE ?

PRAGMATIC
+
PROFESSIONAL



Goal of Practice

Automatic steps

Less Attention on Details

Ability to focus on the big picture

Refactoring

Requires Practice

Myth: Mastery takes 10.000 hours of deliberate practice

Where to Practice?

Test-Drive your Bugfixes

Pet Project (no deadlines)

Synthetic Coding Katas

30 minutes after finishing your task

Coding Kata

Exercises to practice to gain reflexes

Start with this

- **Refactoring:** messy code + tests
→ NO FEAR
- **Legacy Code:** messy code
- **Test-Driven Dev:** requirements



Let's Play a Game ...

"What's not to like about this code?"

- Goals:
 - Articulate what's wrong
 - Separate facts from opinions
 - What's better/worse
 - Articulate exceptions to the rules
 - What do **WE** care (or NOT) about?
 - What do **WE WANT** to change?

Chunking in Refactoring

- **nanosteps**: extract, inline, rename, move "free agents", IDE/editor keystrokes, commit to version control
- **microsteps**: introduce delegate, push details up the call stack, optimize nanosteps based on IDE's affordances
- **moves**: collapse layer, replace inheritance with delegation, replace side-effect with event/listener
- **strategies**: isolate technology integration, separate domain behavior from application behavior, route all events through single source

Challenge Yourself

<https://williamdurand.fr/2013/06/03/object-calisthenics/>

No rush

Safe environment

Perfection

Reflect on HOW you work





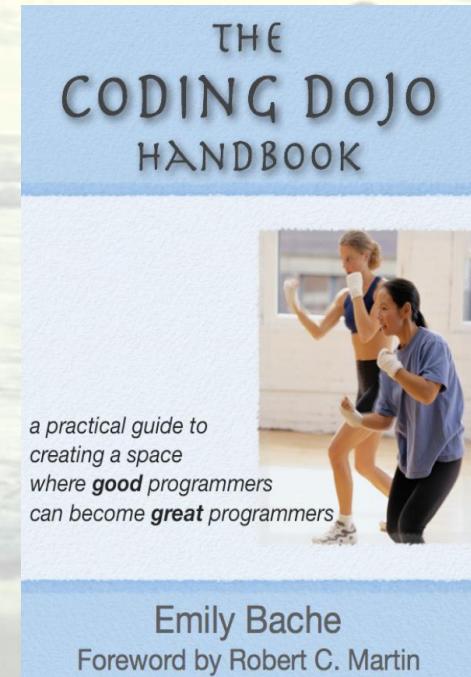
KEEP
CALM
AND
CODE

A classic Uncle Sam recruitment poster from World War I. The central figure is Uncle Sam, depicted as an older man with white hair and a goatee, wearing a top hat with a blue band and stars. He has a stern, commanding expression and is pointing his right index finger directly at the viewer. He is wearing a dark blue jacket over a white shirt. The background is a plain, light color.

Do It Yourselves!!

FEEL:

Pair Programming
Refactoring
TDD



DON'T FEEL:

Rush
Fear
Competition

TDD Kata Restriction:

HORROR: Baby Steps:

Ai 5 minute sa scrii si
testul si sa-l faci sa treaca
Daca nu->REVERT

DO:

Play
Challenge Yourself
Do it 'the other way'

Object Calisthenics

- 1.Only One Level Of Indentation Per Method
- 2.Don't Use The ELSE Keyword
- 3.Wrap All Primitives And Strings
- 4.First Class Collections
- 5.One Dot Per Line
- 6.Don't Abbreviate
- 7.Keep All Entities Small
- 8.No Classes With More Than Two Instance Variables
- 9.No Getters/Setters/Properties

<http://codingdojo.org/KataCatalogue/>

<https://www.codewars.com/>

~~Blamestorming~~

Retro

Some learning key points

Gold Plating



Premature Refactoring

When To Stop?

When To Stop?

When Readable by Colleagues

Talking about Code

Code Review
Pair Programming

What Code should you Refactor First?

Code that annoys the team

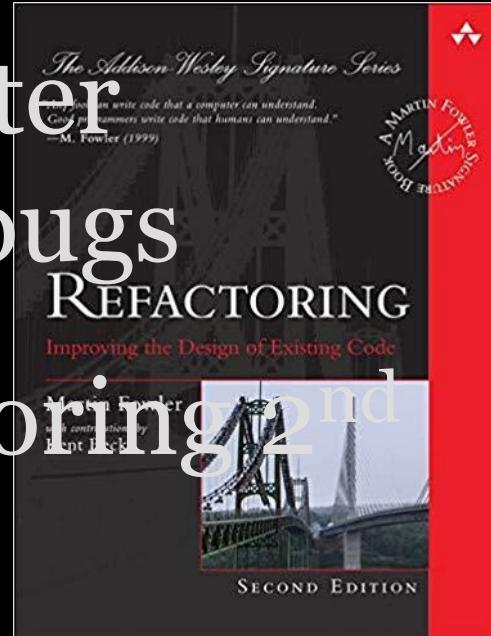
Frequent Bugs or Difficult Changes

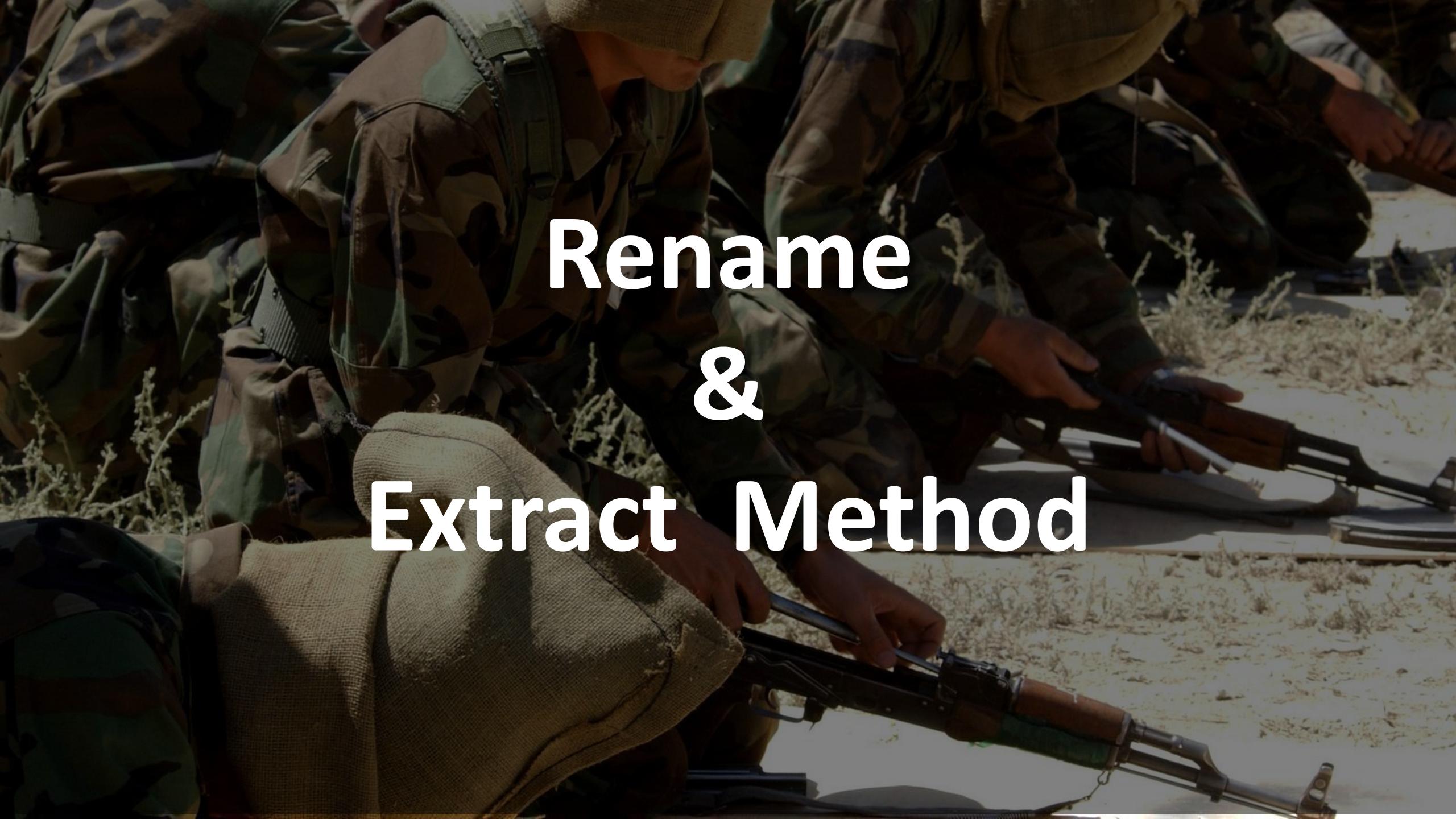


The point of refactoring isn't to show how sparkly a code base is—it is purely economic.

We refactor because it makes us faster
—faster to add features, faster to fix bugs

- Refactoring 2nd



A photograph showing a group of soldiers in camouflage uniforms sitting on the ground in a desert-like environment. They are holding rifles and looking towards the right side of the frame. The scene is set outdoors with dry grass and shrubs in the background.

Rename
&
Extract Method

Refactoring Levels

Level 0: Formatting: autoformat on save based on team stylesheet

Extract/Rename/Inline Private Function

Rename Local Variable/Parameter

Extract/Inline Local Variable

Watch out for repeating impure function calls

100% SAFE

You don't estimate

You don't talk about

You don't ask permission

Change public/global/API stuff

= Life-Style

Introduce abstraction (class)

Break class

Edit code with your brain (!Booleans)

Refactoring Opportunities

<https://martinfowler.com/articles/workflowsOfRefactoring>

Boy-Scout Rule (Litter-Pickup): Leave the code cleaner **[100% safe]**

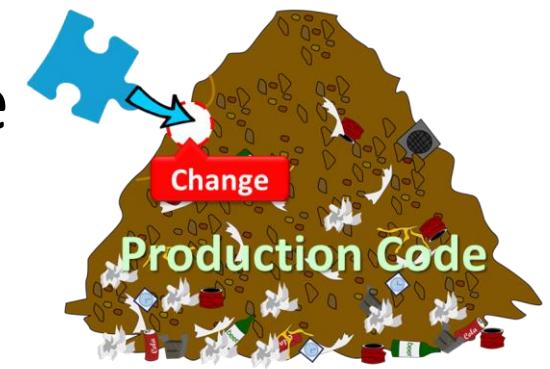
Comprehension Refactoring: Frustration → Aha! → express it

Preparatory Refactoring: before a complex change

Planned Refactoring: in POCs, prototypes?

Long-Term Refactoring: legacy 10K LOC class → set design goals

TDD Refactoring

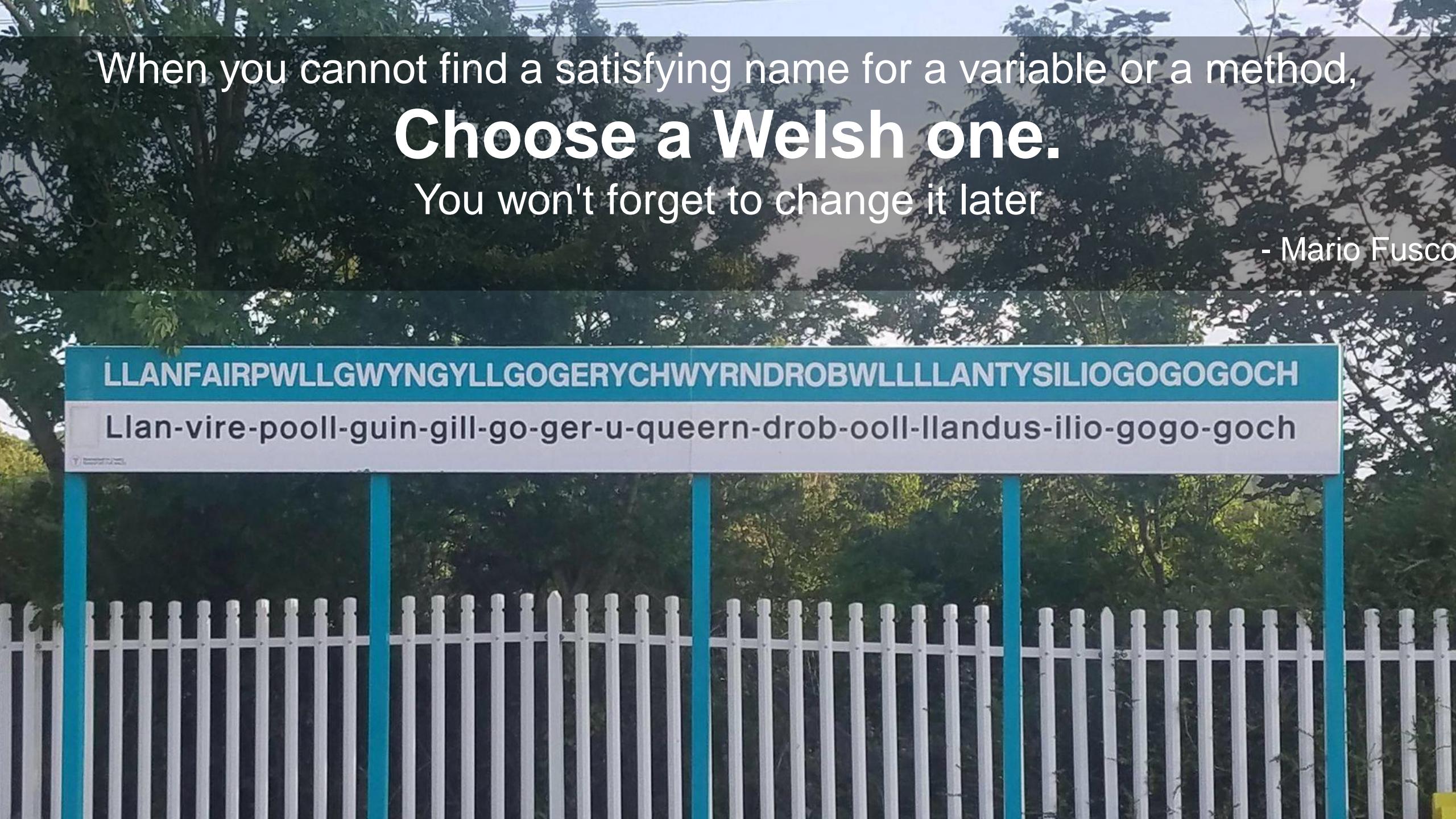


When you cannot find a satisfying name for a variable or a method,

Choose a Welsh one.

You won't forget to change it later

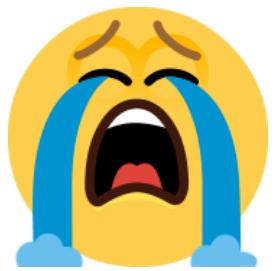
- Mario Fusco



LLANFAIRPWLLGWYNGYLLGOGERYCHWYRNDROBWLLLANTYSILIOGOGOGOCH

Llan-vire-pooll-guin-gill-go-ger-u-queern-drob-ooll-llandus-ilio-gogo-goch

Most Powerful Forces in Universe



2. Revert



3. Ctrl-



Do you love switches?

Switch Hygiene Rules

is the only instruction in a method

default: throw new **JDD**

One-line per case

```
case X: return func();  
case Y : func(); break;
```



Coming soon...

return switch (enum)

[JEP 325: Switch Expressions](#)

Java 17 LTS (sep 2021)

Switch vs Polymorphism vs Enum

```
enum Code {
    A { f(){...}, g(){...} },
    B { f(){...}, g(){...} };
    abstract f();
    abstract g();
}
```

Abstract Enum

- Logic in enums
- Effectively static

```
enum Code {
    A(F::fa, G::ga),
    B(F::fg, G::gb);
    Function<F, Integer> f;
    Supplier<Integer> g;
}
```

Enum + Function Ref

- Per-Function (F, G)
- + Static/Instance funcs
- + n-m matching

```
enum Code {
    A(new ALogic()),
    B(new BLogic());
    ILogic logic;
}
```

Enum + Strategy

- Per-Code logic (A, B)
- Static, no container

```
enum Code {
    A(ALogic.class),
    B(BLogic.class);
}
```

Enum + Strategy with Spring

+ spring.getBean(...)

Good ol' switch

- When repeated: forget to add a case
- Cases tend to grow => Extract
- + Easy

```
enum Code{A,B}

switch(code) {
    case A: fa();
    case B: fb();
}
```

Abstract Class

- Code can't change /time
- + Allows fields/code [OOP]
- + Compile-check ~ enums

```
abstract class X
{ abstract f(); g(); }

class XA extends X
{ f(){...} g(){...} }
```

Enum Map

- null vs default: throw

```
Map<Code, Function<...>>
Map<Code, ILogic>
Map<Code, Class<...>
```

Switch expression

- + Compile-check for enums

```
return switch(code){...};

— in Java14
```

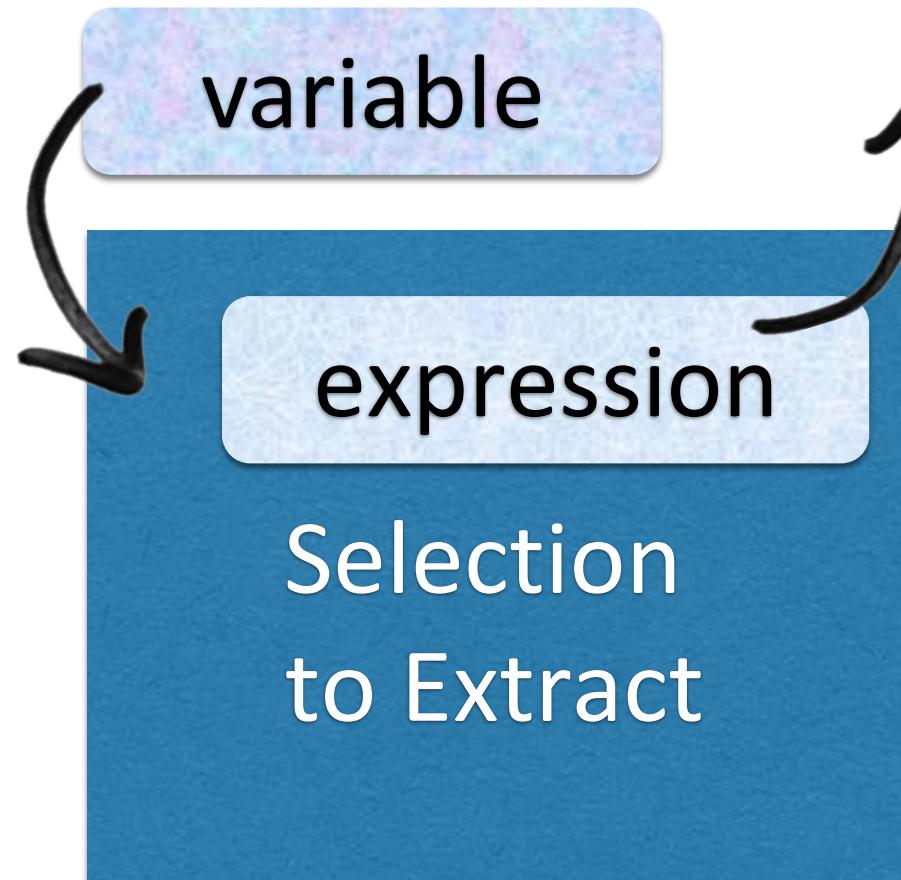
Filters

- + Complex matching
- Tiny classes
- Decentralized

```
interface GFilter {
    boolean accepts(o);
    void process(o)
}
```

Extract Method

Expand Selection
for less parameters



Extract Variables
to generify/reuse
... then extract method
...then inline var back

Feature Envy

=code that works with too much data of another class

...probably belongs inside that other class



$f(\alpha)$ \rightarrow $\alpha.f()$

Switch Structures

```
public double func(Movie movie) {
    switch (movie.getType()) {
        case NEW_RELEASE: ... break;
        case REGULAR: ... break;
        case CHILDRENS: ... break;
    }
}
```

😊 Easy to add functions

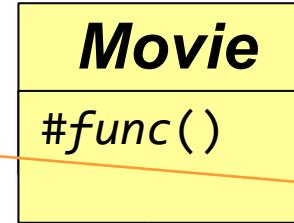
- +1 type-specific functionality
= a new **switch**
e.g. getMaxLoanDays()

😢 Dangerous to add types

- +1 type = a new **case** in all the existing switches

VS

Polymorphism



```
public class NewReleaseMovie
    extends Movie {
    @Override
    public double func(...) { ... }
}
```



abstract methods need to be defined by each concrete type

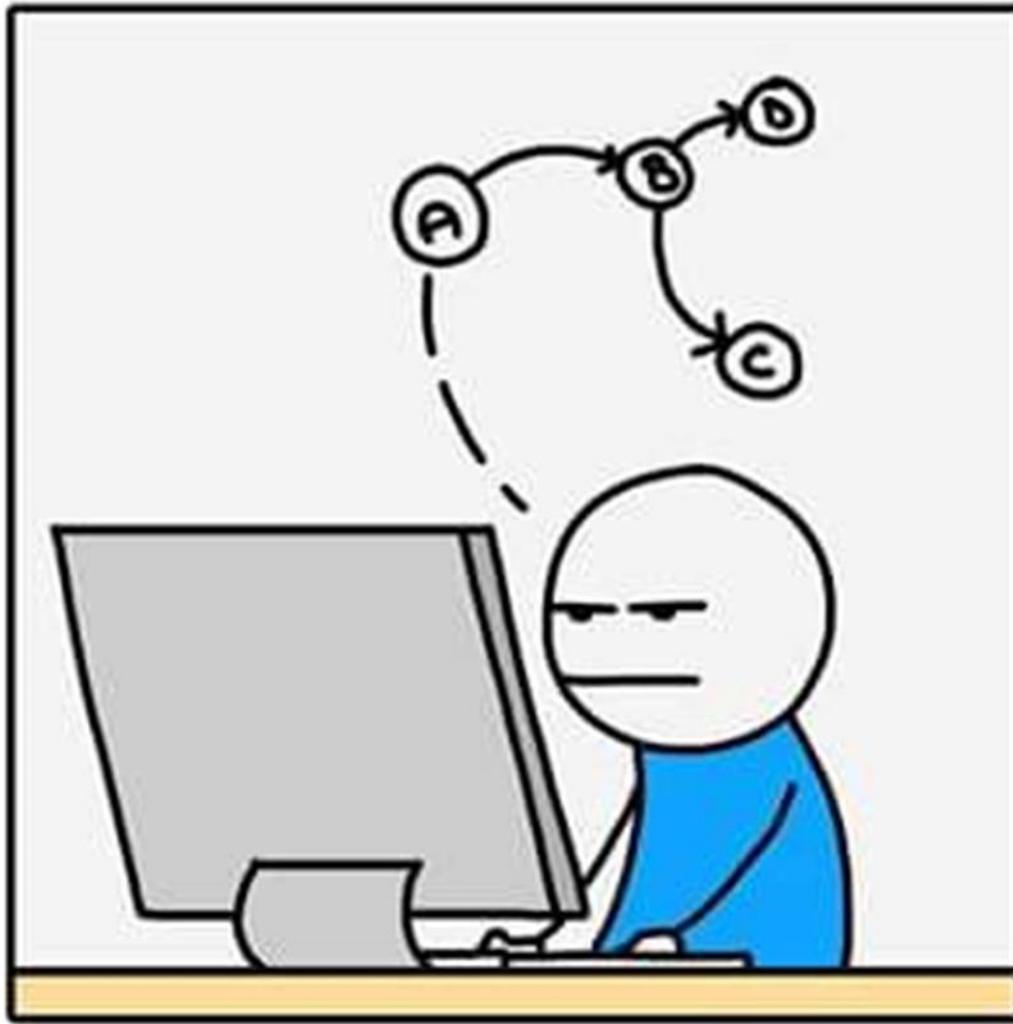
😊 Safe to add types

- Java forces us to implement all the necessary methods

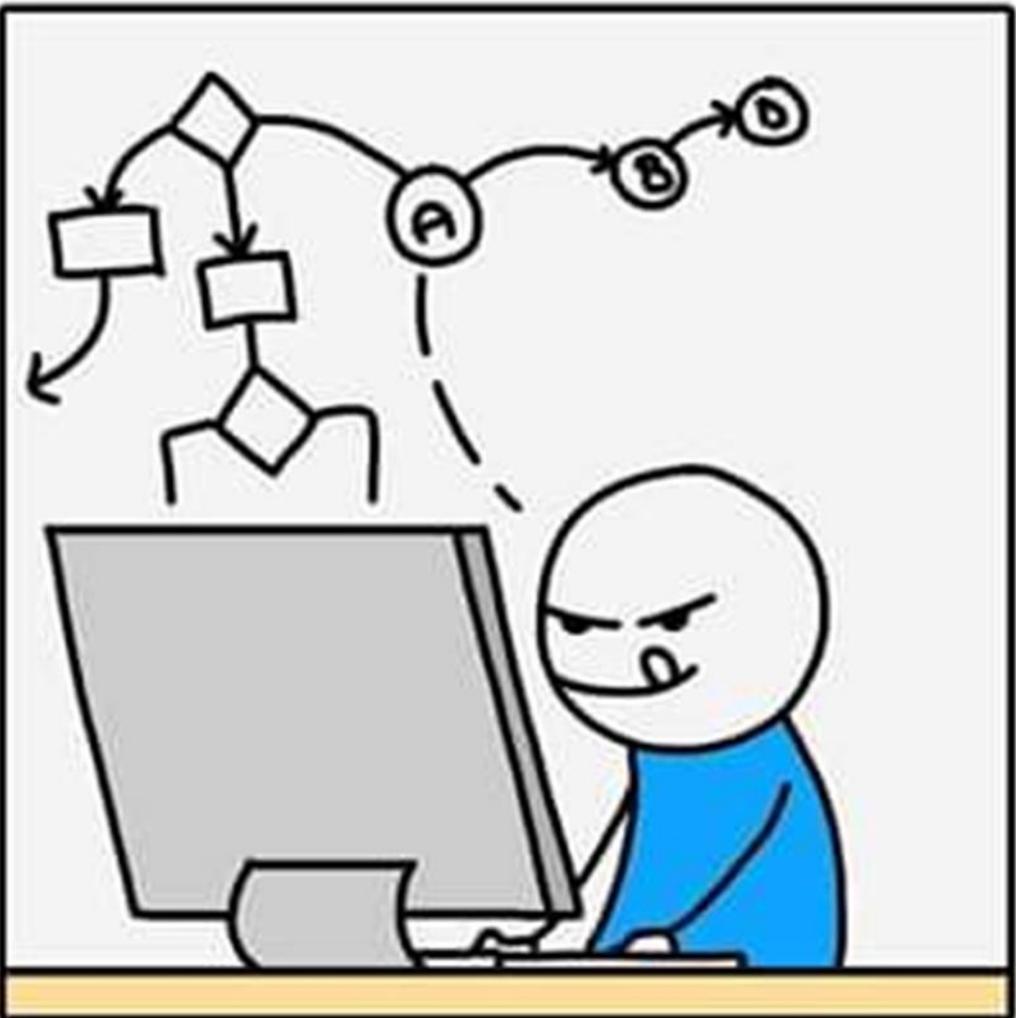
😢 More artifacts: + 3 classes

FOCUS

FOCUS



Focus



Focus



Focus



Focus

FOCUS





Please do not tap on
glass.

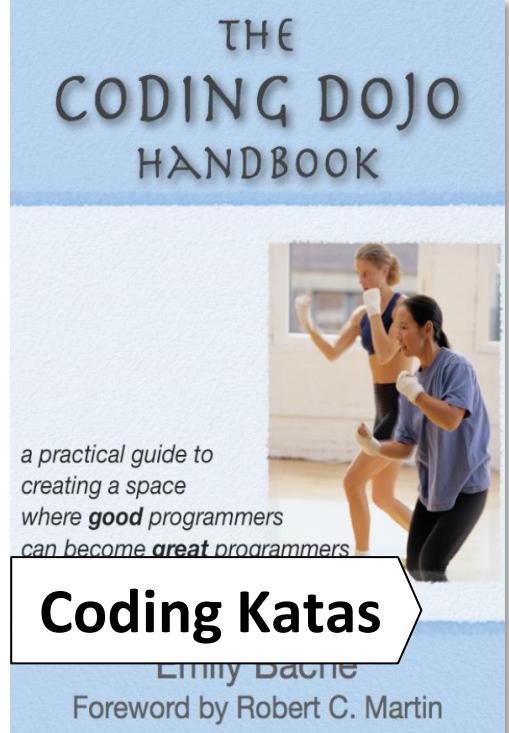
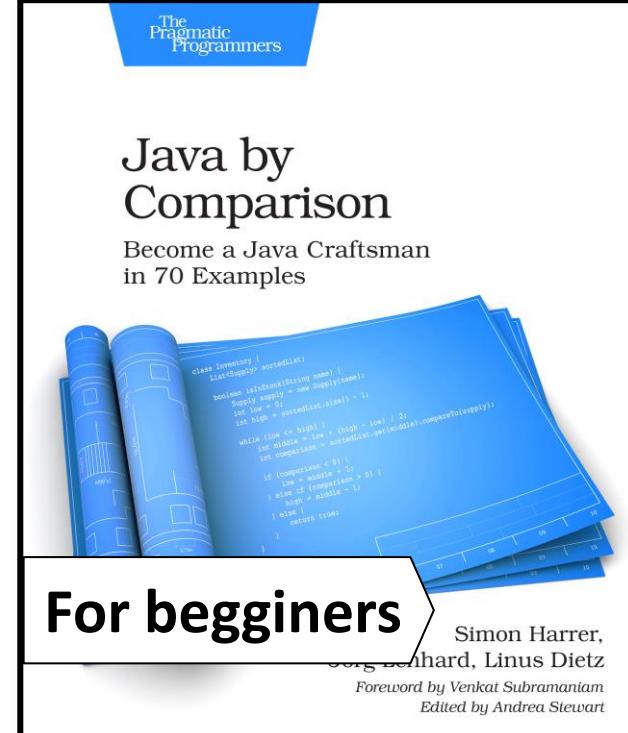
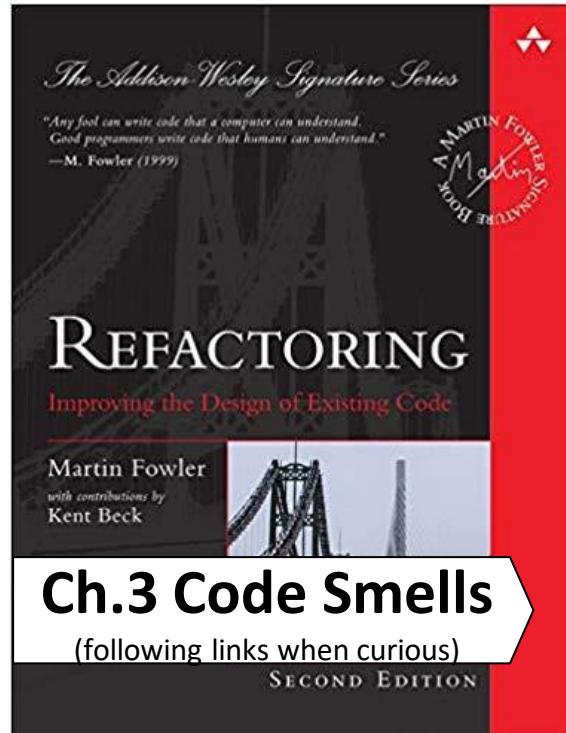
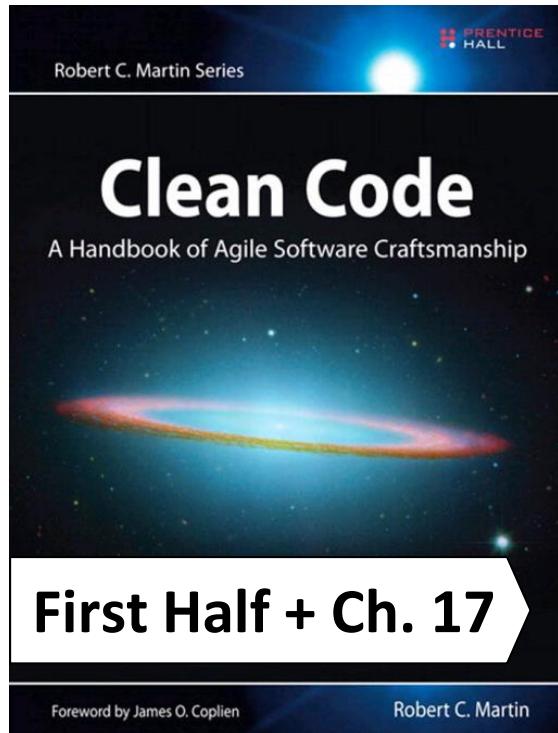
Programmers are easily
scared and will start to
cry. Violently.

Please enter SLOWLY while
singing 'Ave Maria' to avoid
an incident.

Thank you.

Clean Code - Reading Guide

A time-efficient way to get up to speed with Clean Code and Refactoring



cleancoders.com

sourcemaking.com/refactoring

refactoring.guru

CLEAN CODE TODO LIST

- ✓ Review every commit you push until you're **proud** of it (goal: readable, not geek)
- ✓ After finishing a task, spend 30 minutes to *try* to improve it; then 1h, 2h, 4h... until complaints ([link](#))
- ✓ Talk about code. Learn Code Smells. Perform detailed Reviews.
- ✓ Try Pair Programming 1h-2h/week, 25 mins rounds. At least weekly.
- ✓ Install plugins: SonarLint (IJ: run before commit), KeyPromoter X (IJ)
- ✓ Print and learn the shortcut cheat sheet IntelliJ Tricks Video on [victorrentea.ro](#)
- ✓ Practice a Refactoring Kata: <https://kata-log.rocks/refactoring> (GildedRose, Trivia, Yatzi)
 - ✓ Alone, Pairing, Mobbing
 - ✓ Lacking ideas? See other coding > google "coding kata victor rentea"
 - ✓ Too easy? Try some constraints: <https://gist.github.com/asierba/5028e63991ce787fe383>
- ✓ Always reflect on **how** you work.

Hunt me for questions!

Thank You!

<https://github.com/victorrentea/clean-code-dive>

This talk was a sequel of the first episode:
“The Art of Clean Code” at Devoxx Poland 2017

come take 1 sticker or shortcuts sheet

Stay into
The light

Functional Party
Activist

I'm available
a statement of seniority



I use both hemispheres

Speaking Romanian? join me

victorrentea.ro/community

Trainings, talks, goodies

VictorRentea.ro

Download and customize from [victorrentea.ro#stickers](http://victorrentea.ro/stickers)



Refactoring: Ctrl-Alt-Shift-T

Shift-F6 Rename

Ctrl-Alt-M Extract Method

-V ... Local Variable

-F / -C ... Field / Constant

-N Inline

Ctrl-F6 Method Signature

F6 Move: file, method

Alt-Enter Quick-fix: error, warn, todo, gray, refactor



Editing

Ctrl-V Delete Line

Ctrl-D Duplicate Text

Ctrl-[Shift]-W Grow Selection

Alt-Shift-↑ Shift Lines up

Ctrl-Shift-↑ Shift Block up

Alt-J Multi-cursor

Ctrl-Space^{x2} Static functions

Navigation: Ctrl-Shift-C

Alt-F7 Find Usages

Shift-Shift Find File

Ctrl-Shift-F Find Text

Ctrl-H Type Hierarchy

F12 Outline

Ctrl-Alt-H Call Hierarchy

Ctrl-Shift-A Search Anything

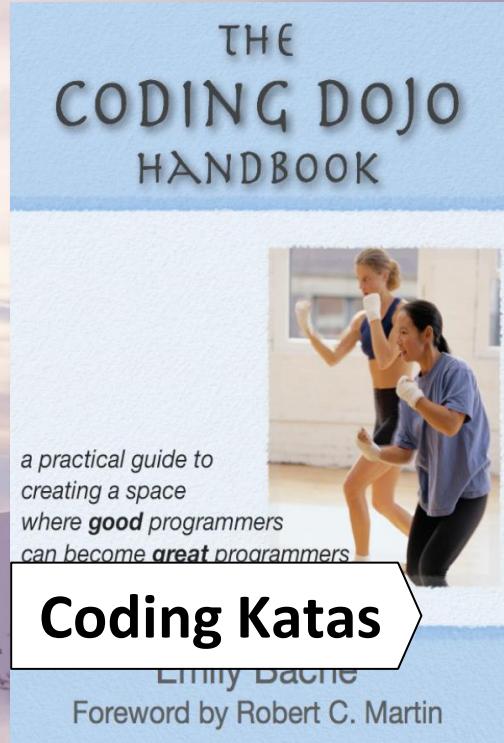
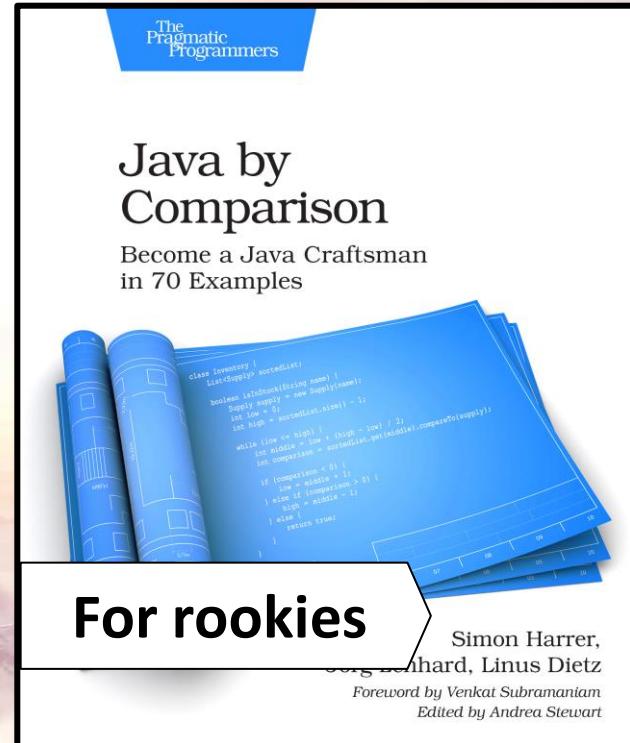
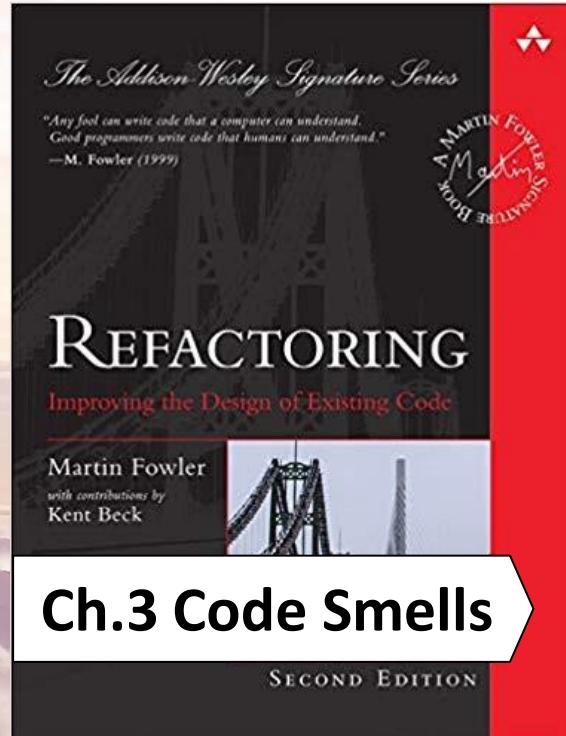
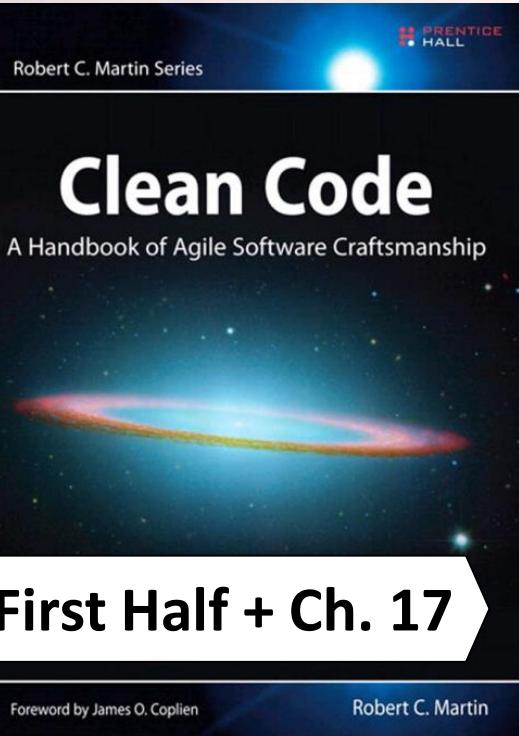
Training. With Personality. VictorRentea.ro



Tough meetings?
Abused estimates?



The End ?



Join me:

victorrentea.ro/community

blog, best talks, training offer

VictorRentea.ro

Share your thoughts



@VictorRentea

Refactoring Quiz

When this happens...

... we can use this refactoring ↓

| | |
|---|---|
| You want to reuse logic or to document its intent: | Extract Method |
| Push a literal or expression from your function into whoever calls it: | Introduce Parameter |
| Move logic into the class holding all the data required by that logic (Feature Envy smell): | Move Method |
| Replace a local variable with its value or expression: | Inline Variable |
| You want to write a second implementation of your class' public API: | Extract Interface |
| Replace the (multiple?) calls to a function with the function body: | Inline Method |
| Extract an expression into an explanatory variable: | Extract Variable |
| Extract <i>similarly</i> looking code as a common function (sequence of 3 steps): | Extr Variable > Extr Method > Inline Variable |
| Long loop doing many things: | Split Loop |
| Group several parameters from a function signature into a new class: | Introduce Parameter Object |
| Lots of variables defined at the start of a long function: | "Move closer to usages" (IntelliJ) |
| A variable 'result' assigned in many places + ending return result; | "More return closer" (IntelliJ) |
| Anemic else branch with no or trivial logic | Early Return (flip IF + remove else after return) |
| Direct external access to [public] fields: | Encapsulate Fields |

Clean Code in Javascript / FE

- - TypeScript or at least babel / ES6;
- > const rocks !
- - var x = 1, y = 2 <<< don't !
- - install SOnarLint plugin (WebStorm/VSC/Eclipse) connect to sonarqube on pipeline > AND > jslint (gradle run on pipeline failing the build) > ESLINT
- - brainstorm best practices of using chosen frameworks; do workshops/ coding katas together:
- "let's create a simple screen with a search and a grid, or an edit screen, together in 1-2 hours"
- - establish code conventions : the shape of FE code matters more than in BE as the refactoring tools are not as powerful.
- - DRY : aggressively eliminate any repetition
- - extract large callbacks into separate functions
 - > in general, tolerate larger functions than in Backend, especially if they handle presentation
 - > or ng,react,vue reduce the boilerplate
- - pass "this" as an argument to extracted functions to avoid closure swap
- - Build a non-invasive framework AFTER 3-6 months of development of a new project; retrofit already written code (planned refactoring sprint); establish the coding guidelines; target the usual bugs, annoying code or complex problems; HELP the developers, don't CONSTRAIN them.
- > don't ever dream it will be reused in the next project: don't make it overly generic
- > DO tolerate a certain degree of heterogeneity between screen implementations
- - In front of lots of code, extract the different responsibilities in separate functions, especially if they repeat. eg validation, presenting errors, updating totals
- - Explanatory variables: eg
- var selectCallback = function (event, ui) {
- var serviceProviderData = ui.item; <<< TypeScript ROCKS, did I say that ?
- - Nice idea to extract constraints in dedicated data structures !
- - Become EXPERT in Array functions