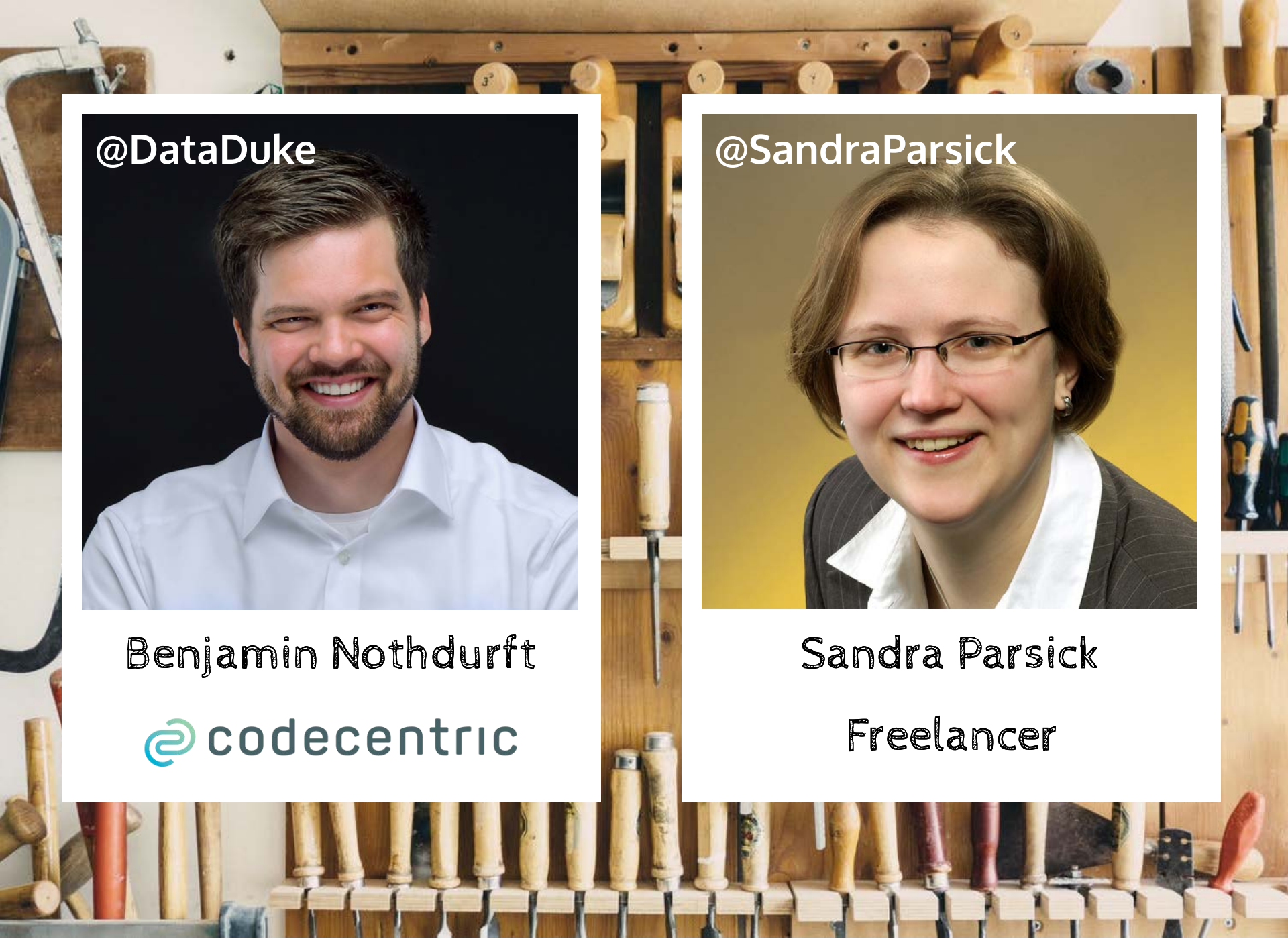
Give the aspiring developer a code kata!

What the heck is Software Craft(smanship)?

Benjamin Nothdurft
Sandra Parsick

2019-01-17 at jugsaxony.org

Martin-Luther-Universität Halle-Wittenberg



Benjamin Nothdurft

codecentric

Sandra Parsick

Freelancer

Agenda

Brief History - 10"

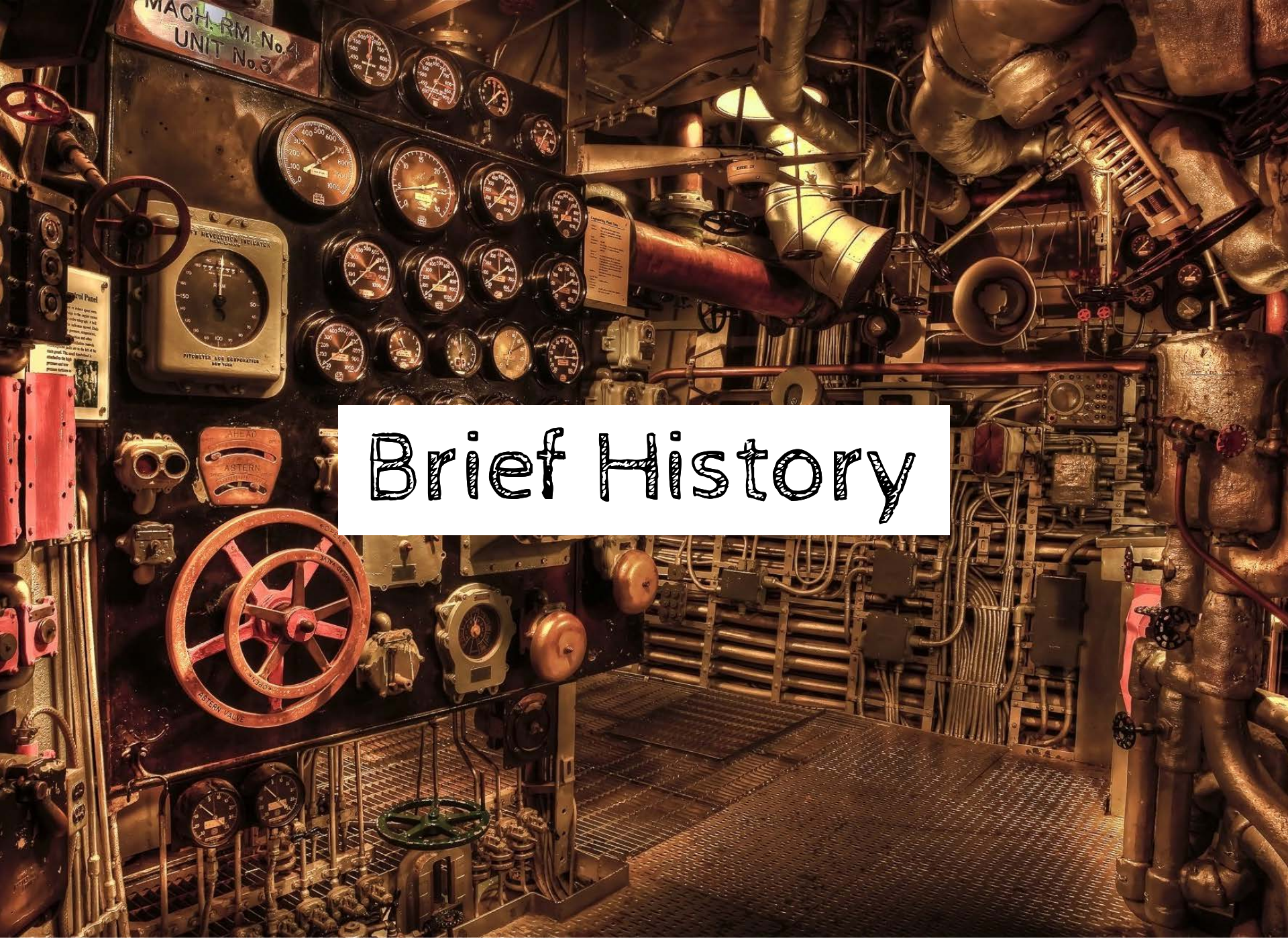
Current State - 5"

Dojo & Katas - 25"

Clean Code - 5"

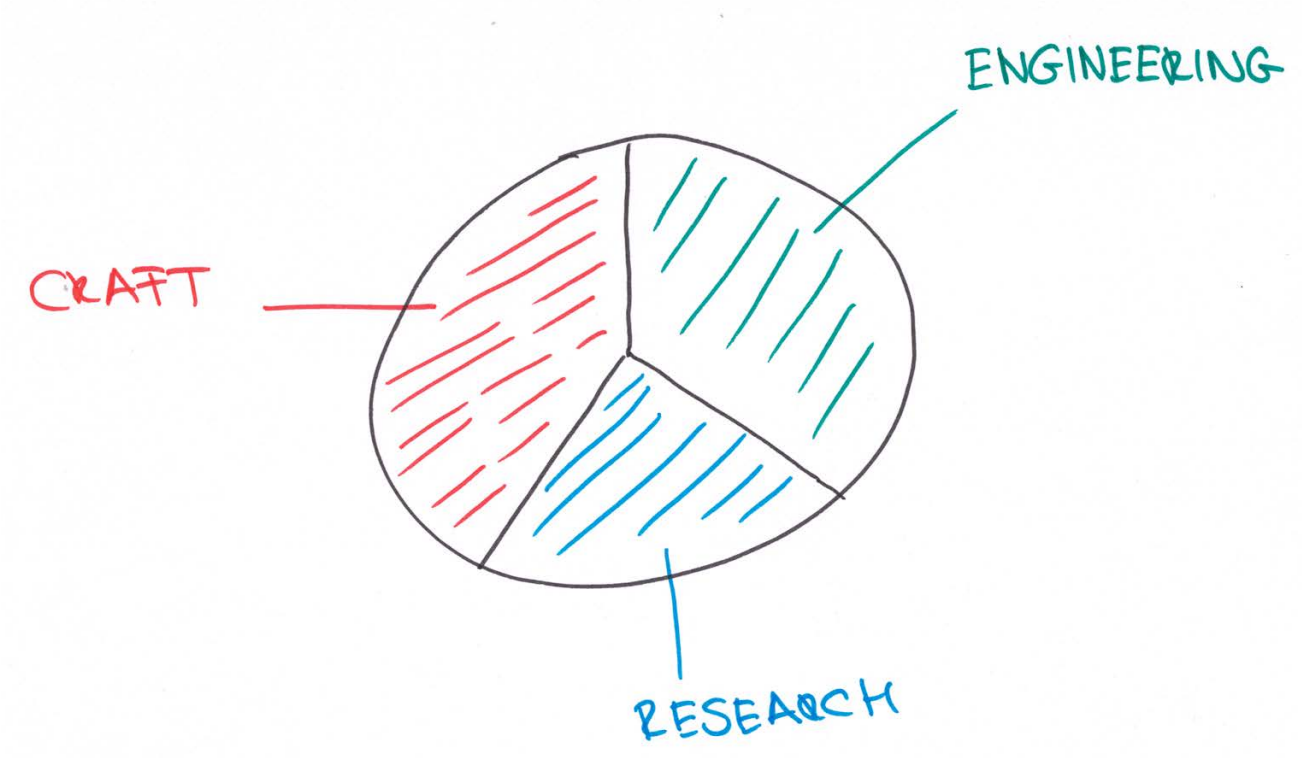
Disclaimer: Thanks to our fellow crafters!

@MarcoEmrich / @DavidVoelkel



Brief History

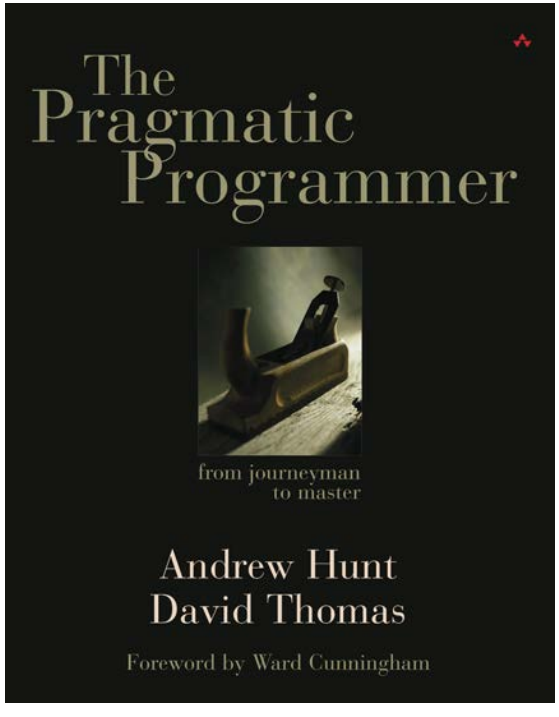
Software Development

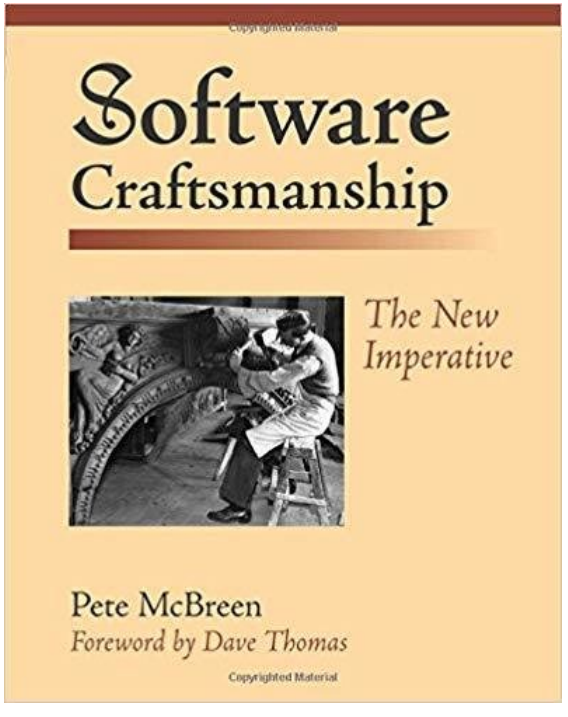


Today we know **software development** is a mixture out of **craft, research and engineering**!

However it was not always like this...

1999-2001



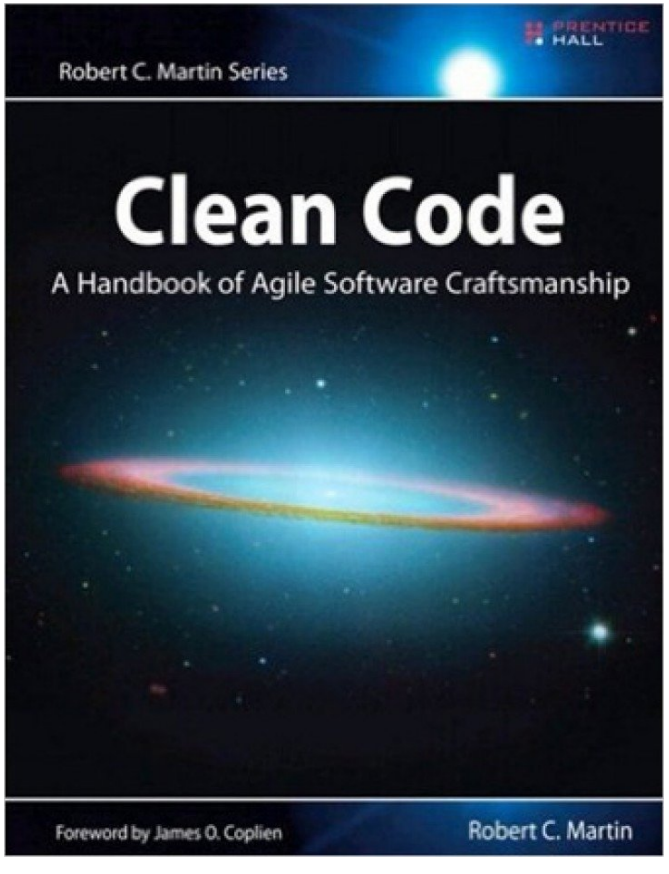


- Idea in the 90s: **software factories** for automated software development
- First try: **Engineering as craft!**

2001 - Birth of Agile

- Agile reaches from **Scrum** (project view) to **XP** (technical view)
- Focuses however strongly on **project process!**
- **Technical excellence** largely neglected.

2008 - Agile Hangover



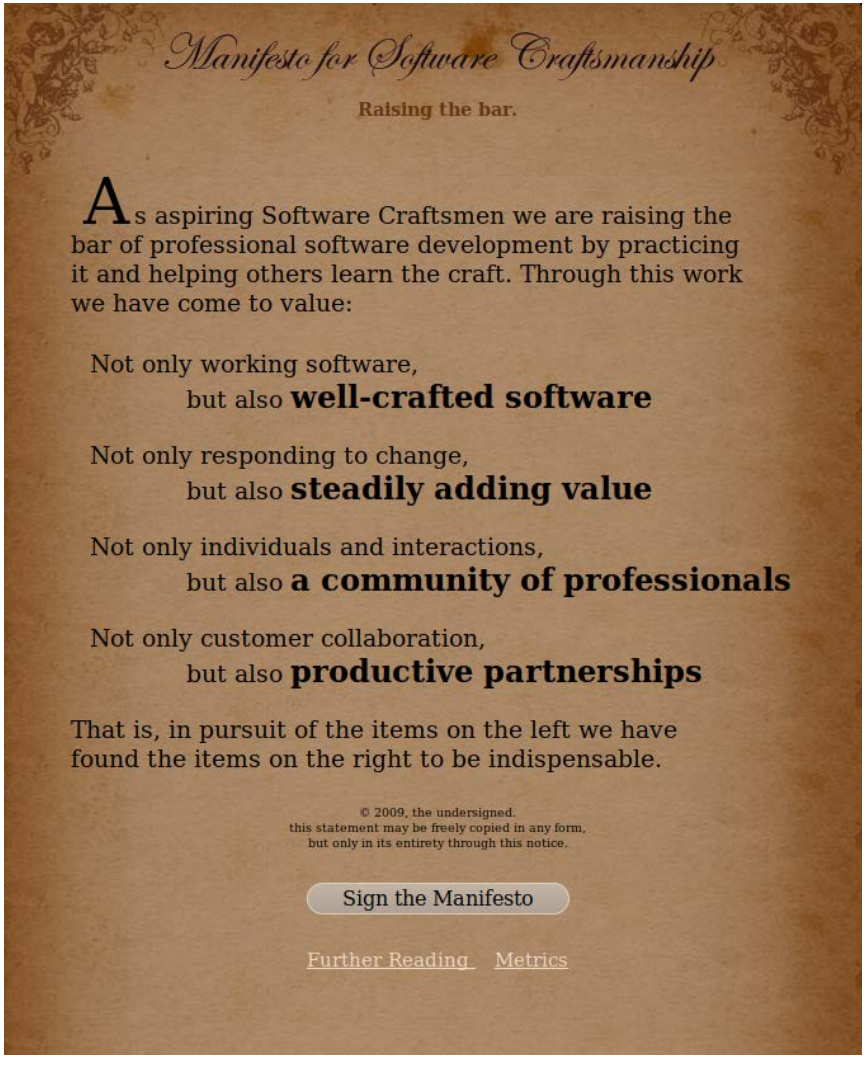
Keynote from Uncle Bob:

- 5 principles to **"craftsmanship over crap"**

...later transformed to:

- **craftsmanship over execution**

2009 - Manifesto for SC



Manifesto for Software Craftsmanship

Raising the bar.

As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

That is, in pursuit of the items on the left we have found the items on the right to be indispensable.

Sign the Manifesto

Further Reading Metrics

2002 - Software Apprenticeship Summit:

- no outcome!

2008 - SC Summit: **Micah Martin** gave a session

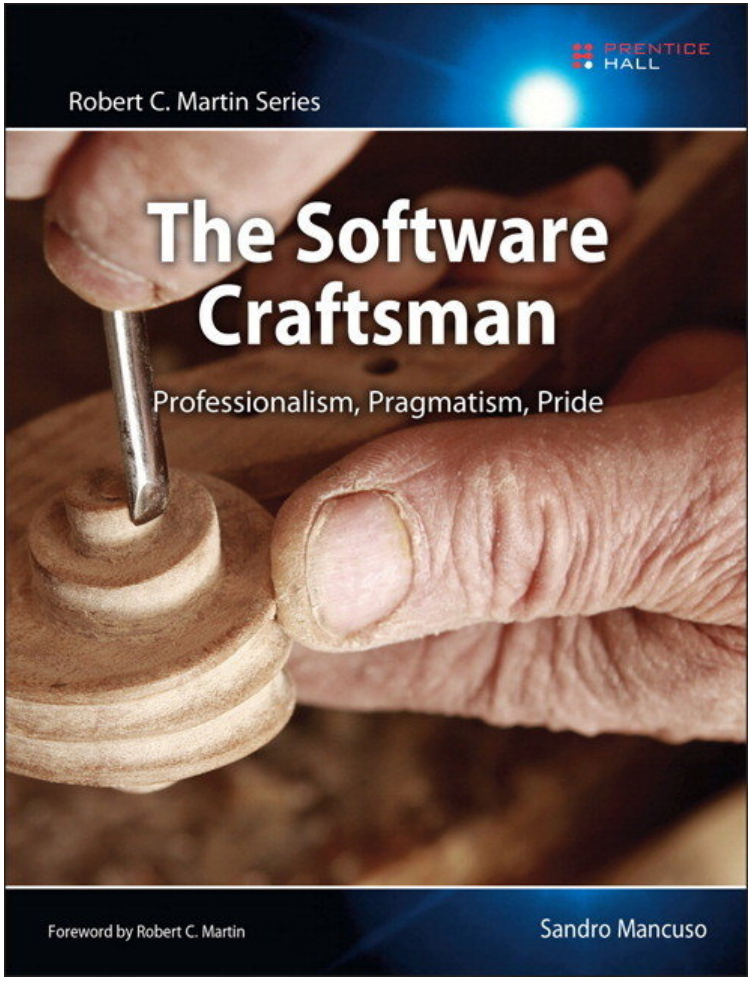
- many ideas as outcome
- whiteboard was signed by everyone

2009 - **Doug Bradburry** wrote in SC Google group "The New Left Side" vs. **Scott Pfister** "Right Side, Revisited"

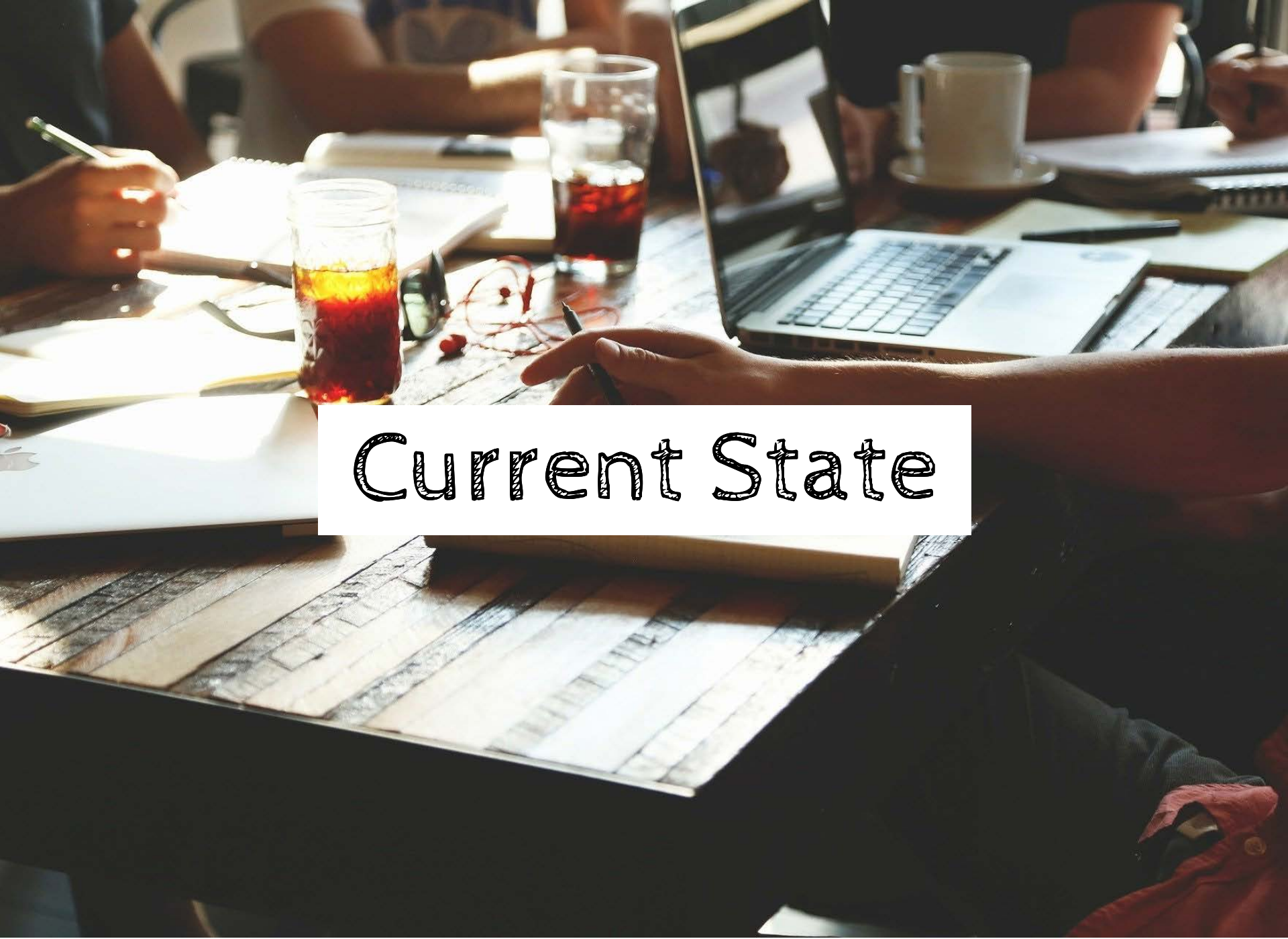
Why a manifesto?

1. vocal community
2. create visibility
3. establish principles
4. develop schools
5. guidance for new devs

2014 - The Software Craftsman



- Ideology and Attitude
  - History
  - Professionalism
  - Practises
  - ...
- Full Transformation
  - Recruitment
  - Interviews
  - Culture
  - Pragmatism
  - Career


Current State

Conferences

2009 - First SC conferences in USA, UK

2009 - Israeli SCC was founded

2010 - London SCC was founded

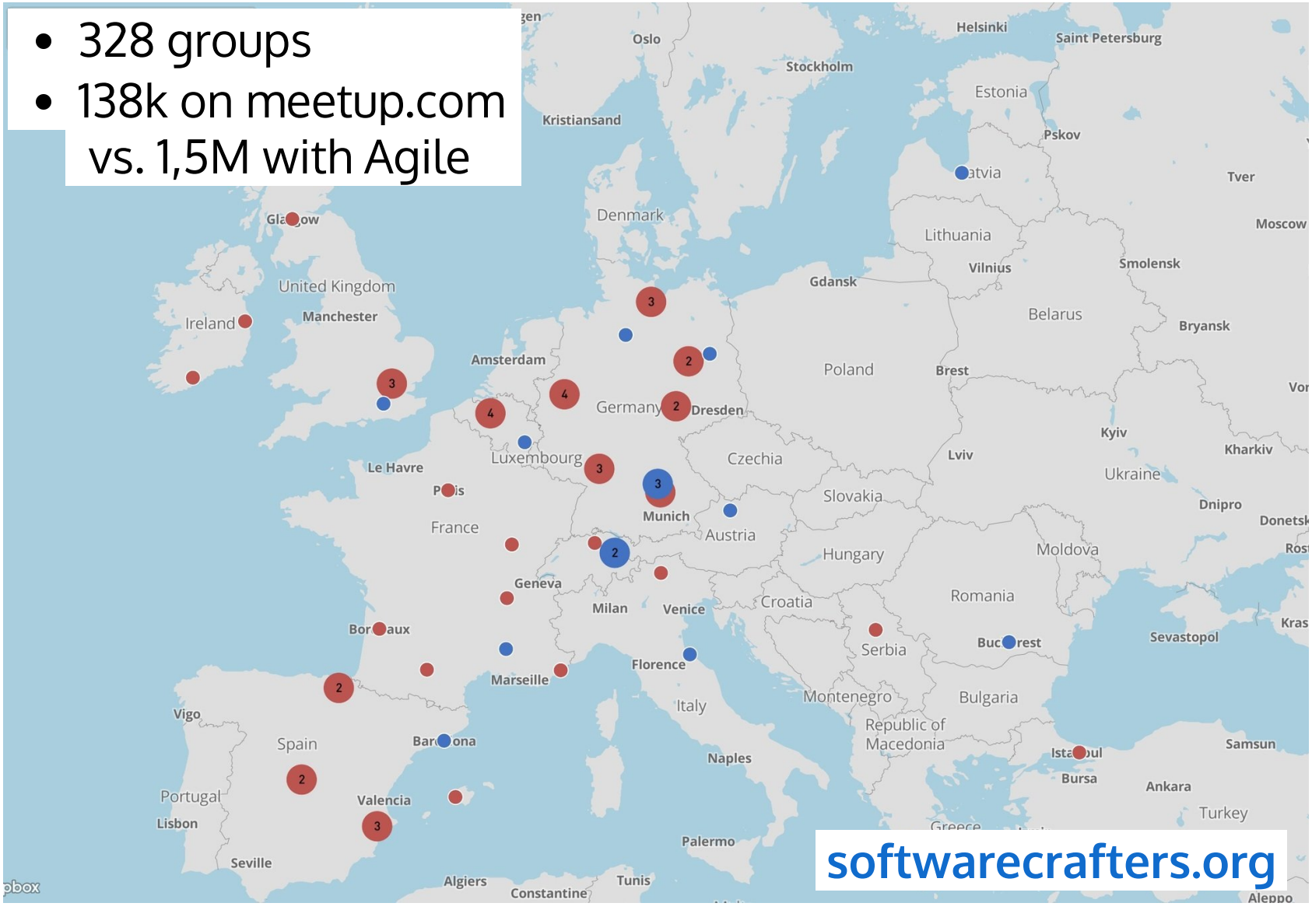
2011 - First SoCraTes in Germany

Today - SoCraTes (partner) conferences/days in:

Germany, Chile, Canaries, Italy, UK, USA, Switzerland, France, Austria, Belgium, Finnland, Romania...

SCC Communities

- 328 groups
- 138k on meetup.com
  vs. 1,5M with Agile



Communities in DACH region

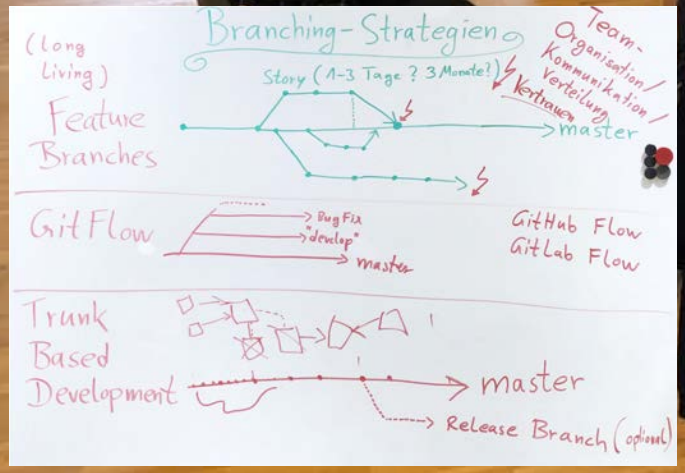
Members:

- 29 regional groups
- 2k on website
- 9k on meetup.com

softwerkskammer.org

Activities

- Open Space
- Birds of Feather (BoF)
- LeanCoffee.org
- Hackergarten.net
- CodeRetreat.org



..but we need to go back to the roots!

- many **new aspiring devs**
- **principles got lost again down** the road while doing other activities
- IT market is booming and we **need technical excellence** to tackle our software products
- arising lack of broad **TDD knowledge**

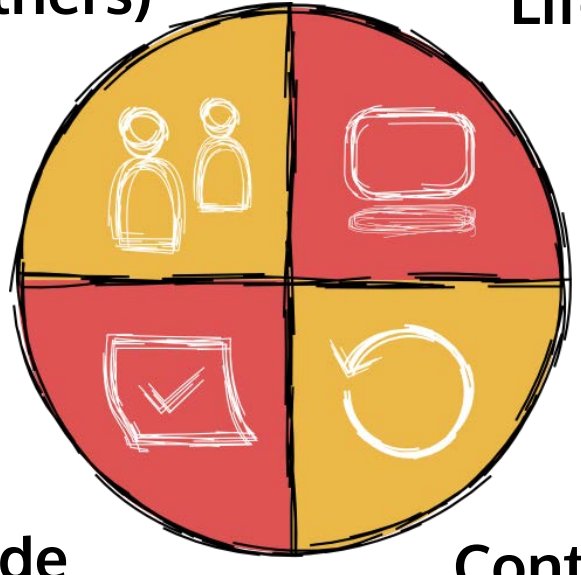

Dojo & Katas

Craftsmanship Principles

Individuals & Interactions

(Learning from each others)

Lifelong Learning



Clean Code

Continuous Improvement

(Practice)

What is a coding dojo?

" A bunch of coders get together, code, learn, and have fun. It's got to be a winning formula! – Emily Bache

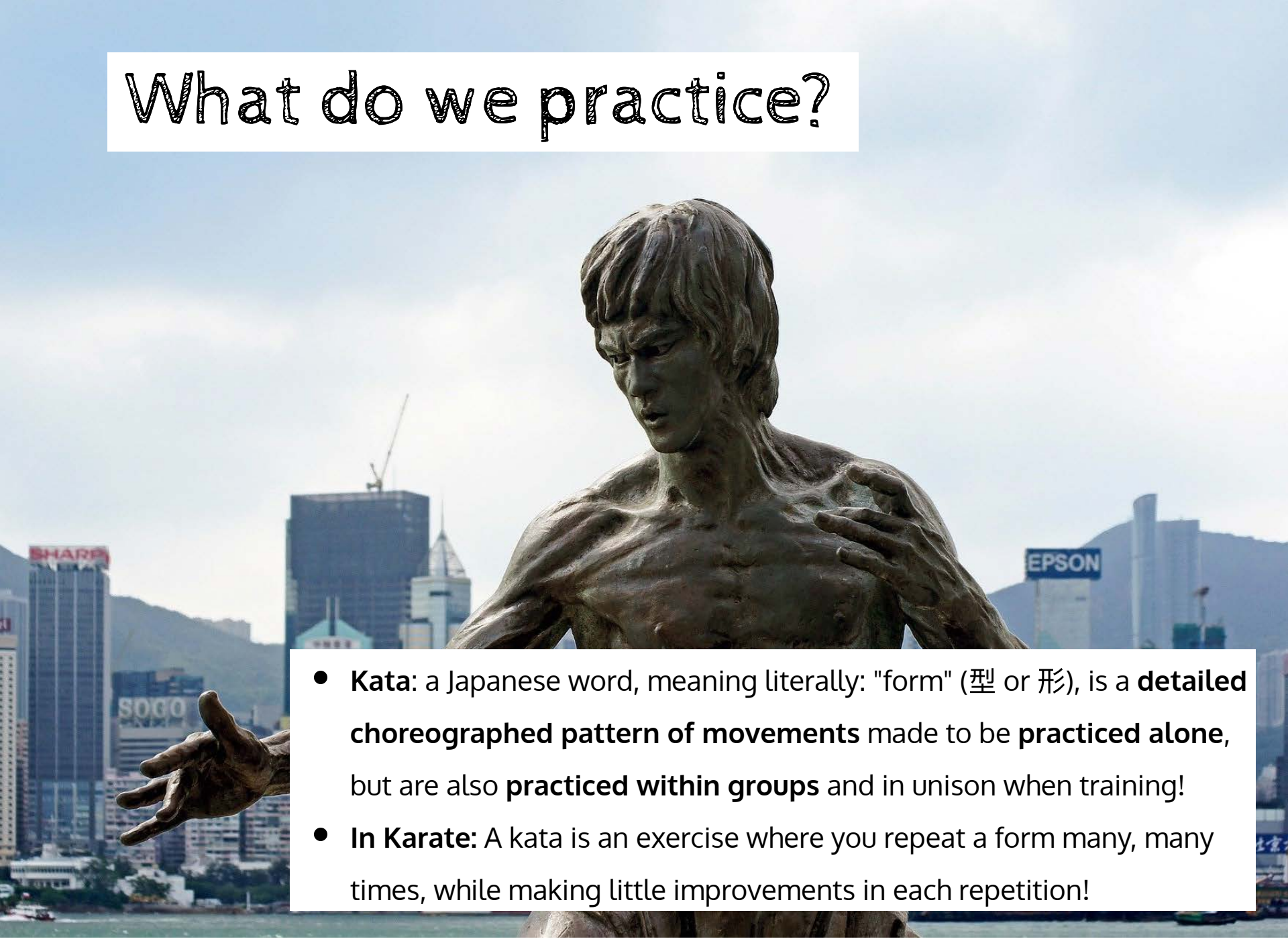
Why do we need a coding dojo?

- no managers, no deadlines
- **safe environment**
- **all professionals need to practice!**
- not all forms of practice are equal
- special way to practice
- designed to **emphasize skills** that are hard to aquire and easy to lose!

Coding Dojo Principles

- **First Rule:** *Design cannot be discussed without Code, Code can not be shown without tests.*
- **Come with your relicts**
- **Learning Again**
- **Slow down**
- **Throwing yourself in**
- **Finding a master**
- **Subjecting to a master**
- **Mastering a subject**

What do we practice?



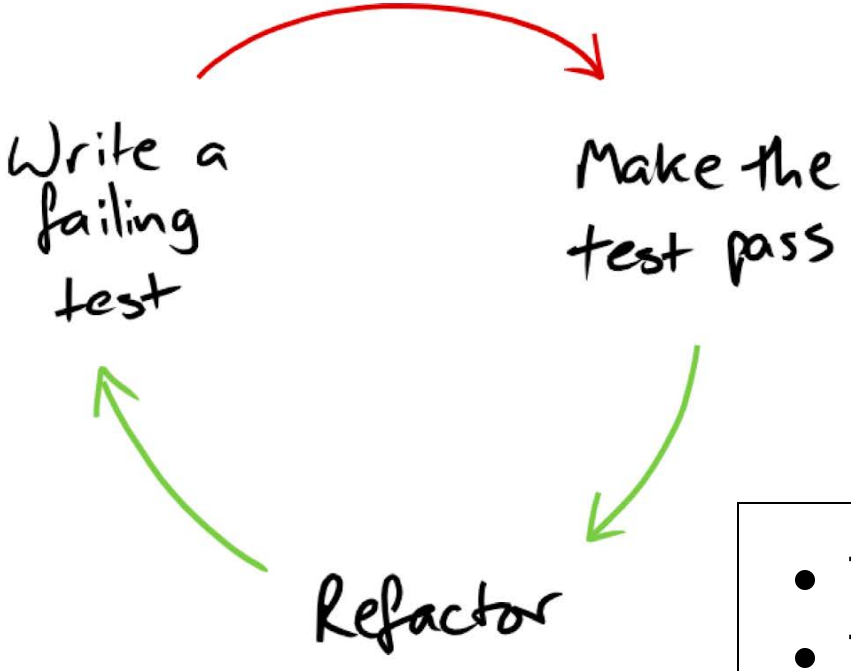
- **Kata**: a Japanese word, meaning literally: "form" (型 or 形), is a **detailed choreographed pattern of movements** made to be **practiced alone**, but are also **practiced within groups** and in unison when training!
- **In Karate:** A kata is an exercise where you repeat a form many, many times, while making little improvements in each repetition!

Characteristics of a Code Kata

- **Definition:** A kata is a defined solving flow of a **code exercise** made to be practiced many, many times **alone**, in **pairs** or as **groups** (e.g. MOB Programming) while making little improvements.
- **Duration:** Most exercises are quite short (~ **30 minutes to 1 hour**) so that one can incorporate them as routines in daily life!
- **Content:** Some involve **programming**, and can be coded in many different ways. Some are open ended, and involve **thinking** about the issues behind programming, e.g. architecture katas.
- **Focus:** The point of the kata is not arriving at a correct answer. The point is the stuff you learn along the way. **The goal is the practice, not the solution!**

> *" TDD is used as a default pattern for coding!*

What is TDD? Why is it so hard?



Goals:

- Higher dev speed
- Better code quality
- Patterns: AAA

- TDD is not about testing!
- TDD = specs/design
- QA is minor point
- TDD is living documentation
- Isolation, Focus
- Test new behaviour in babysteps

FizzBuzz Kata

Task:

- Write a program that prints the **numbers from 1 to 100 but:**
- ...for multiples of 3 print **Fizz**
- ...for multiples of 5 print **Buzz**
- ...for multiples of both 3 and 5 print **FizzBuzz**

Example:

- 1, 2, **Fizz**, 4, **Buzz**, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, **FizzBuzz**, 16, 17, Fizz, 19, Buzz, Fizz, 22, 23, Fizz, Buzz, 26, Fizz, 28, 29, ...

*first described in the essay "Fizz! Buzz!" (~1987) by David Langford as a **drinking game** of his teenage years in the 1960s*

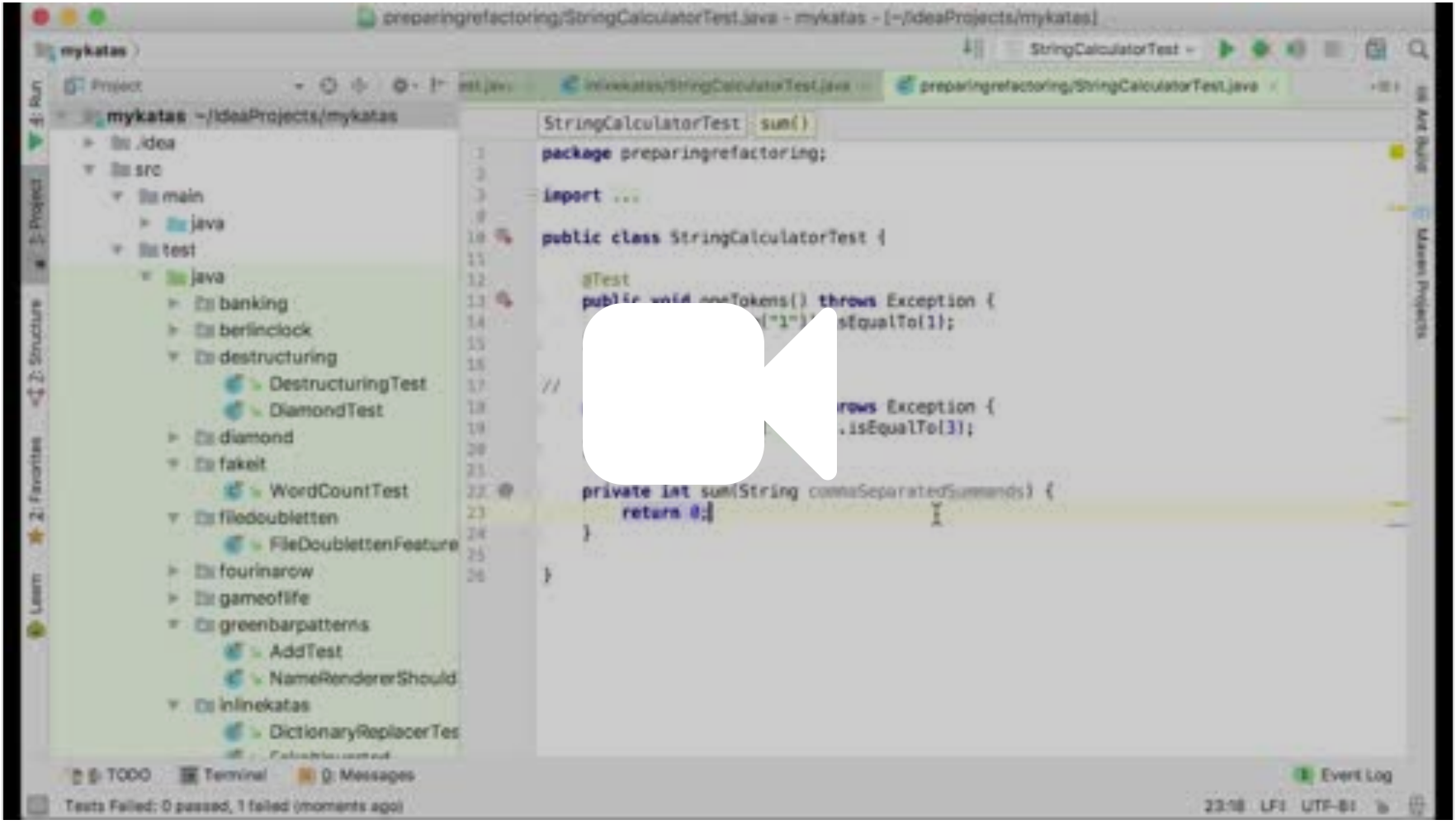
String Calculator Kata

1. Create a simple String calculator with a method **int add(string numbers)**
    - can take **0, 1 or 2 numbers** and will return their sum, e.g. “” or “1” or “1,2”
    - Start with the simplest testcase of an empty string and move to 1 and 2 numbers
    - Remember to solve things a**s simply as possible**
    - Remember to **refactor after each passing test**
2. Allow the **add method** to handle an **unknown amount** of numbers
3. Allow the **add method** to handle **newlines between numbers** instead of commas.
4. Support **different delimiters** with pattern: //[delimiter]\n[numbers…] , e.g. “//;\n1;2”
5. Calling add with a **negative number** should **throw an exception** “negatives not allowed”
6. Ignore **big numbers**, e.g. boundary is 1000 then 1001 + 2 = 2

…

Idea by Roy Osherove

String Calculator Kata (Video)



Steps 1. + 2. = Solved with preparatory refactoring

Discussion points for retro

- Did you ever write **more code than you needed** to make the current tests pass?
- Did you ever have **more than one failing test** at a time?
- Did the tests **fail unexpectedly** at any point? If so, why?
- How much did **writing the tests slow** you down?
- Did you write **more tests** than you would have if you had coded first and written tests afterwards?
- Are you **happy with the design of the code** you ended up with? Should you have refactored it more often?

How do I facilitate a dojo meeting?

Upfront:

- Book a room, Invite people, Print copies of kata description, prepare some slides for dojo introduction, inspect the chosen kata upfront

Start:

- Line people up by experience and match people with the most with the ones with lowest etc. (folding queue)

During:

- Facilitator needs to create good/healthy atmosphere, prompt interesting discussions, keep the code growing,
- Try not stop people when they mess up with TDD, let them learn from mistakes, wait until retro before saying anything!

Code Retreat Day

09:30 **Intro/Talks**

10:45 **Round 1**

11:45 **Round 2**

12:45 Lunch

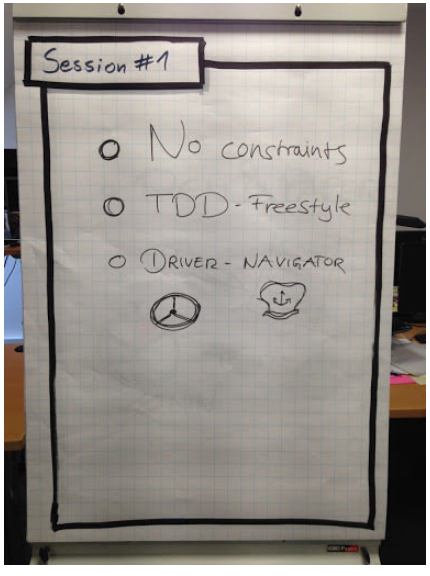
13:30 **Round 3**

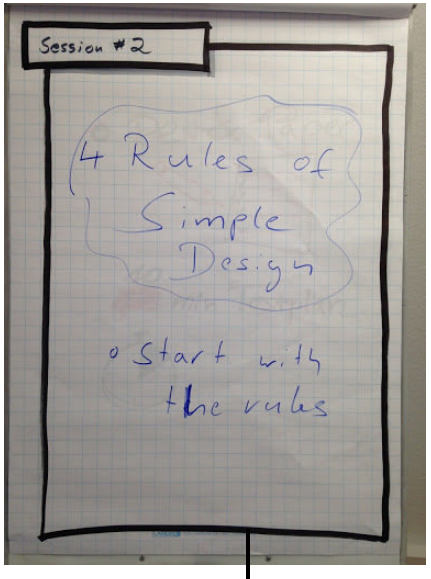
14:30 **Round 4**

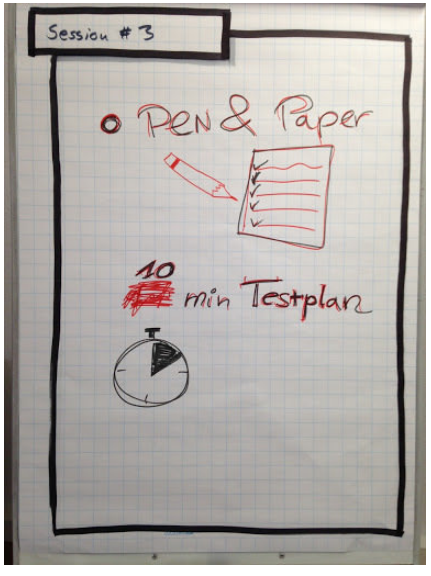
15:30 **Round 5**

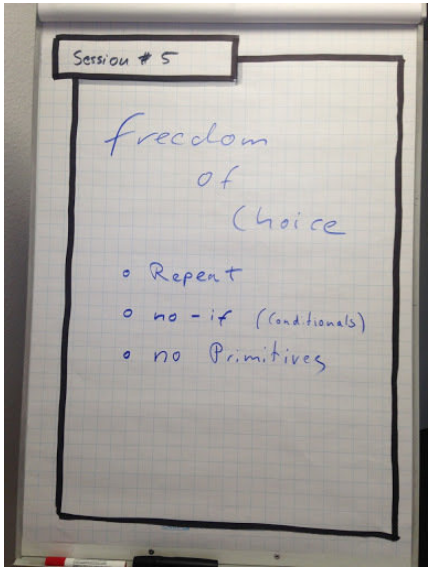
A format popularized by Corey Haines

45" code + 10" retro + 5" break









Kent Beck:

- passes all tests
- maximizes clarity/intention
- minimizes duplication (DRY)
- has fewer elements

Corey Haines: https://leanpub.com/4rulesofsimpledesign

Constraints

- **Basic Activities**
  - Ping Pong
  - Navigator-Driver
- **Missing Tool Activities**
  - No Mouse
  - Text Editor only
  - Paper only
- **Missing Feature Activities**
  - No naked primitives
  - No conditional statements
  - No loops
- **Quality-Constraint Activities**
  - Only four lines per method
  - Immutables only, please
- **Stretch Activities**
  - Verbs instead of Nouns
  - Code Swap
  - Mute with find the loophole
  - TDD as if you meant it

...more Constraints

- Baby Steps
- Silent Coding (Mute)
- No If
- No IDE
- No Mouse
- Only One-Liners
- Every Cell is a Microservice (at Game of Life)
- ...

...more selected Katas

- Bowling Game Kata (by Robert C. Martin)
- Prime Factors Kata (by Robert C. Martin)
- FizzBuzz Kata
- BankOCR Kata
- Ordered Jobs Kata
- Roman Numerals Kata
- Kebab Kata
- ...

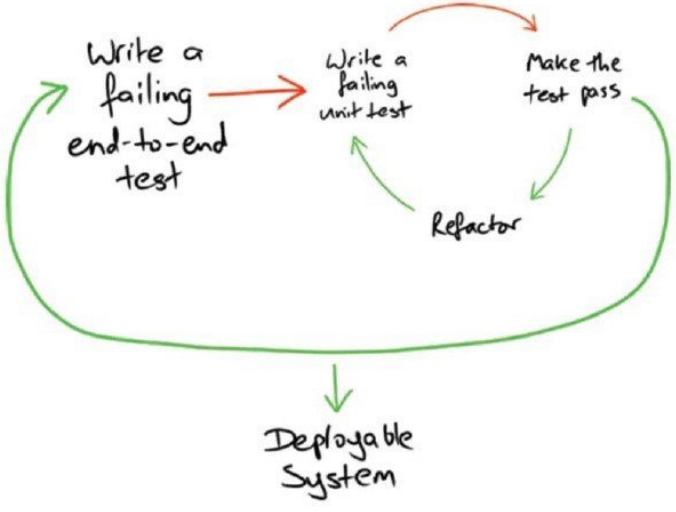
Katalogues

- http://kata-log.rocks
- https://leanpub.com/codingdojohandbook
- https://codingdojo.org/kata
- http://ccd-school.de/coding-dojo
- http://codekata.com
- http://www.thesoftwaregardener.com/agile/dojo-code-katas
- http://cyber-dojo.org
- http://es6katas.org
- https://www.codewars.com
- https://exercism.io
- http://katas.softwarecraftsmanship.org


Clean Code

Different TDD schools

- **London School (Mockist)**
  - Double Loop ATDD ⟶
  - Outside-In Design



- **Detroit School (Classicist)**
  - Kent Beck, Uncle Bob...
  - front-door testing
  - state verification
  - only mock the process boundary (DB, 3rd party)
  - design emerges bottom-up / inside-out
  - "TDD as if you meant it"
- ***"Munich School"***
  - *Fake-it Outside-In Design*

"TDD as if you meant it"

1. You are not allowed to write **any production code** unless it is to **make a failing unit test pass**.

2. You are not allowed to write **any more of a unit test** than **is sufficient to fail**; and compilation failures are failures.

3. You are not allowed to write **any more production code** than is **sufficient to pass the one failing unit test**.

Three rules by Robert C. Martin / Keith Braithwaite

SOLID Principles

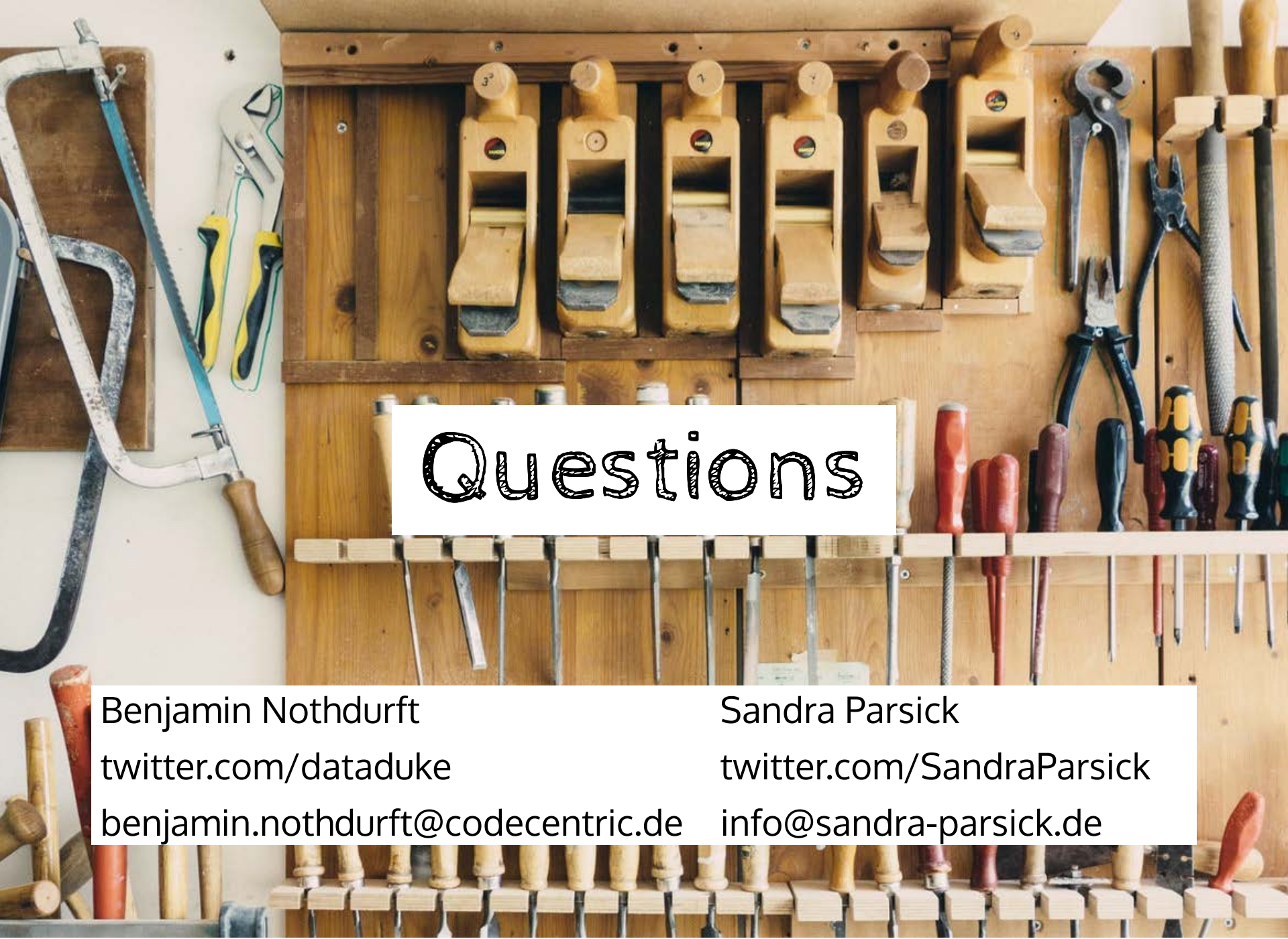
- **Single responsibility (SRP)** "a class should have only a single responsibility"
- **Open/closed** "software entities... should be open for extension but closed for modification"
- **Liskov substitution** "objects should be replaceable with instances of their subtypes without altering the correctness of that program"
- **Interface segregation** "many client-specific interfaces are better than one general-purpose interface"
- **Dependency inversion** "depend upon abstractions, not concretions"

by Robert C. Martin / Acronym by Michael Feathers

Other Principles

- **KISS** - "keep it simple, stupid"
- **DRY** - "Don't repeat yourself"
- **YAGNI** - "You aren't gonna need it"
- **DTSTTCPW** - "Do the simplest thing that could possibly work"
- ...
- ...
- ...

...and many more principles can be practiced with Katas!



Questions

Benjamin Nothdurft
twitter.com/dataduke
benjamin.nothdurft@codecentric.de

Sandra Parsick
twitter.com/SandraParsick
info@sandra-parsick.de