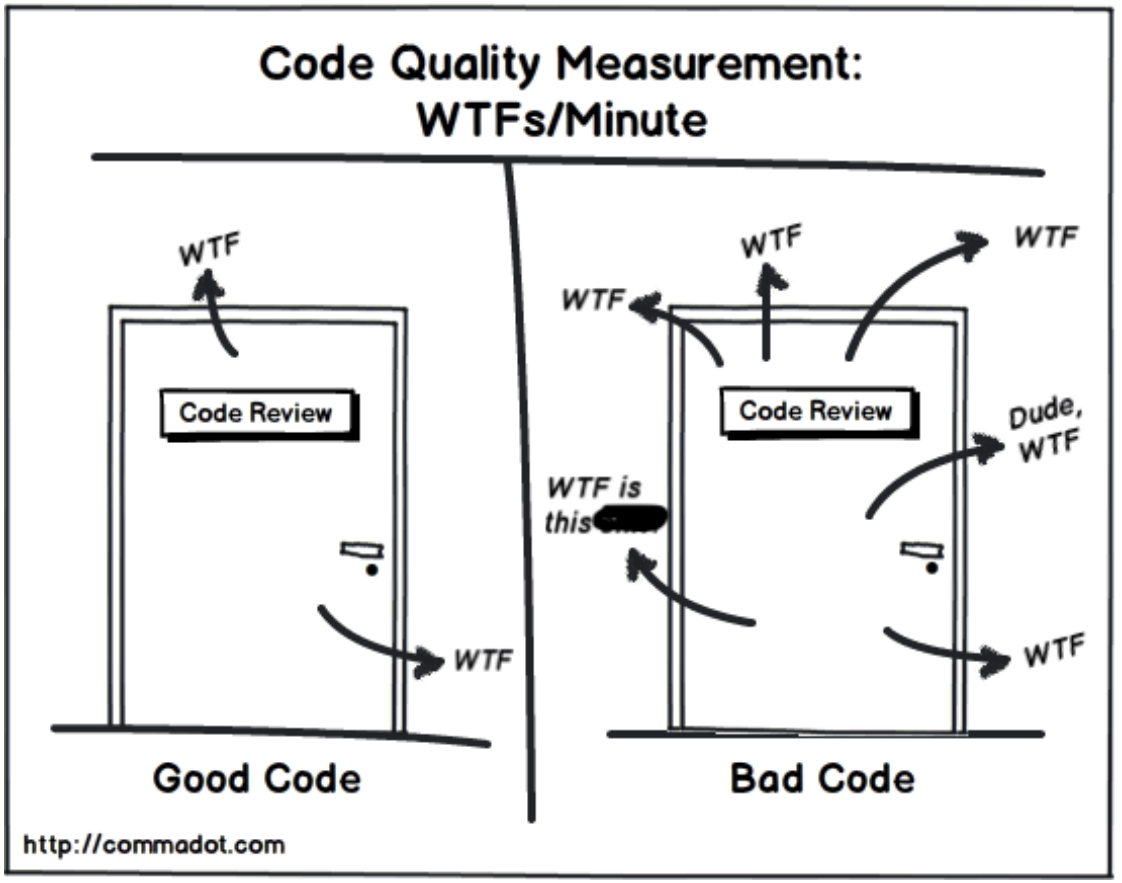
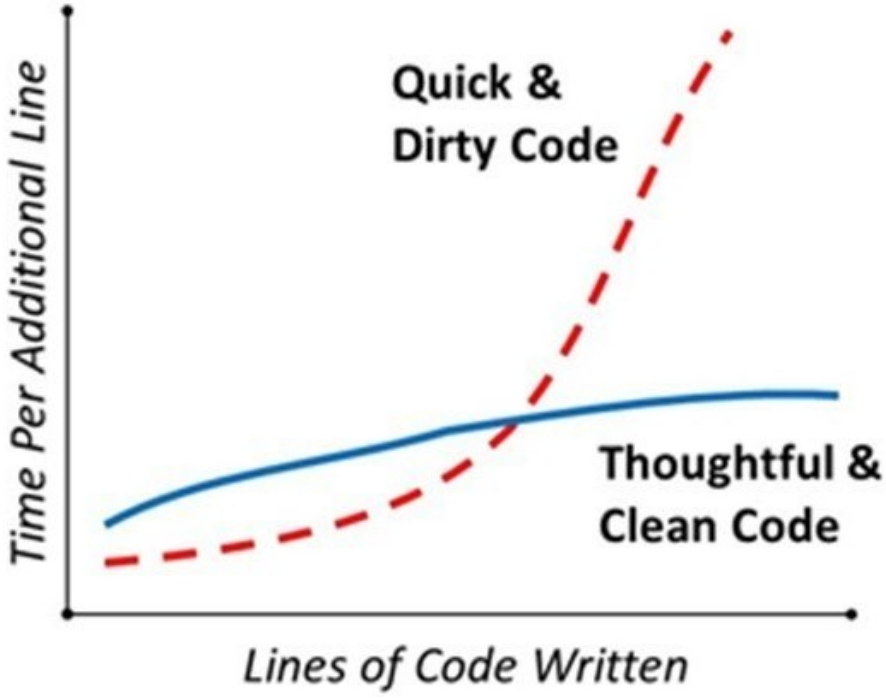
Clean Code


Code Quality Measurement:
WTFs/Minute
WTF
Code Review
WTF
Good Code
WTF
WTF
WTF
WTF
Code Review
Dude,
WTF
WTF is
this
WTF
Bad Code
http://commadot.com

Warum Clean Code?

- Chaos ist teuer
- Verhältnis Code schreiben zu lesen (1:10)
- Bugs können sich schlechter verstecken
- Konsistente Team Produktivität
- Weniger cognitive Load



Was ist Clean Code?

> “ Any fool can write code that a computer can understand.
> Good programmers write code that humans can understand. ”
>
> – Martin Fowler

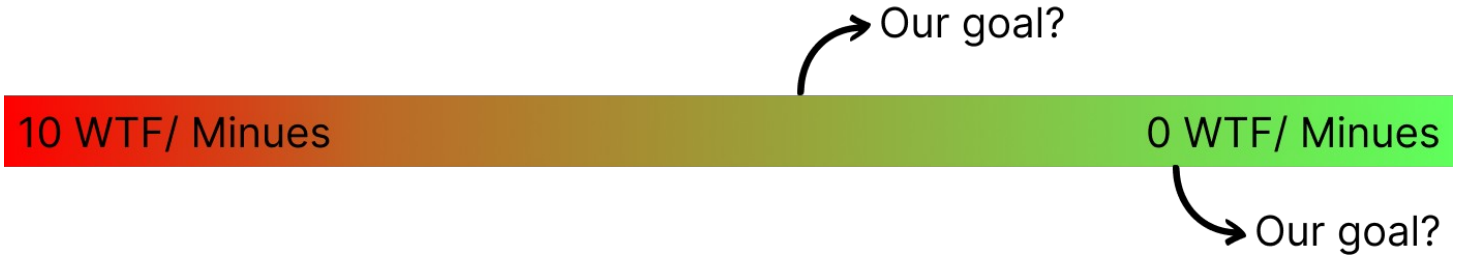
Selbsterklärender Code

wartbar, anpassbar und skalierbar

konsistent und standardisiert

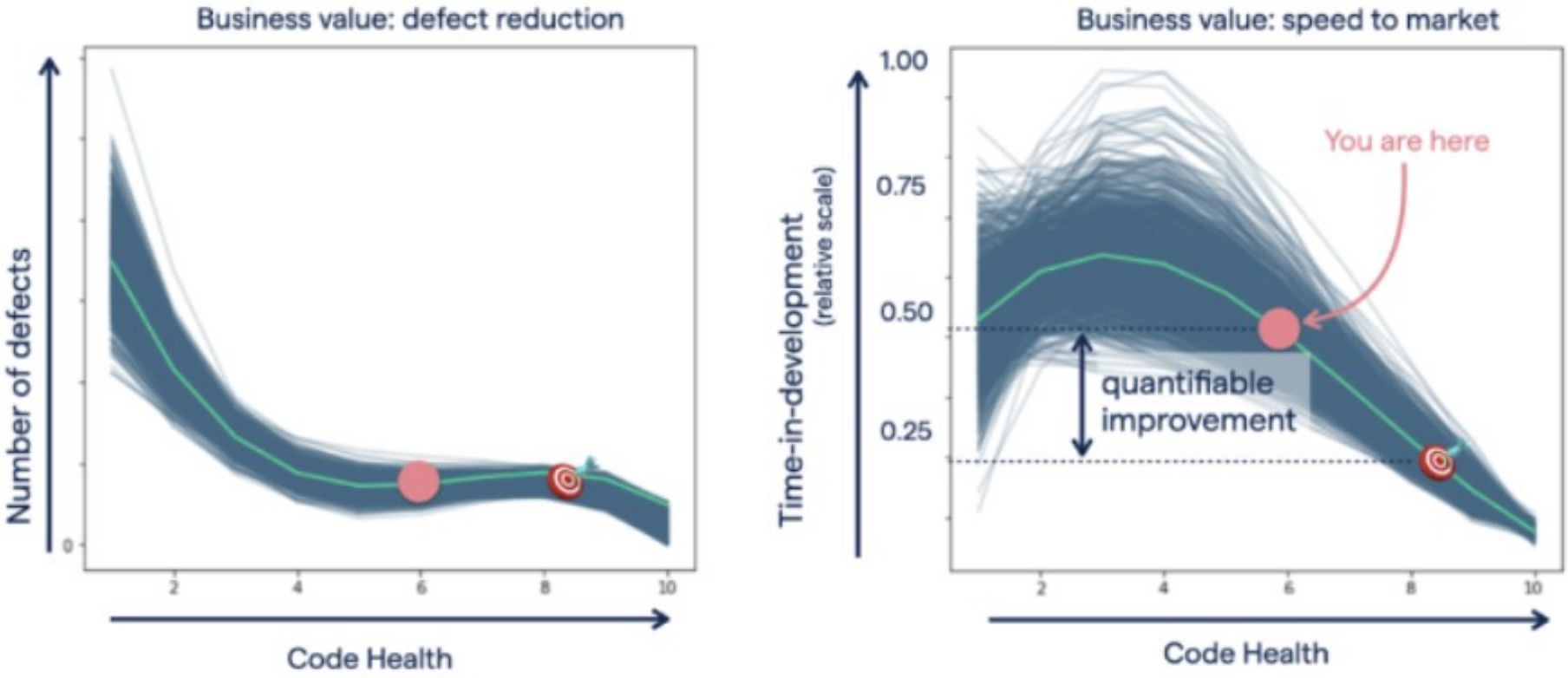
professionell und branchenüblich

Geht es um persönliche Vorlieben?



- Wenn du das klügste/erfahrendest Mitglied in deinem Team bist und komplexen Codes problemlos verstehen kannst, ist der Code dann verständlich genug für das gesamte Team?
- Kann der Code modifiziert werden, ohne die Angst, etwas zu zerstören?

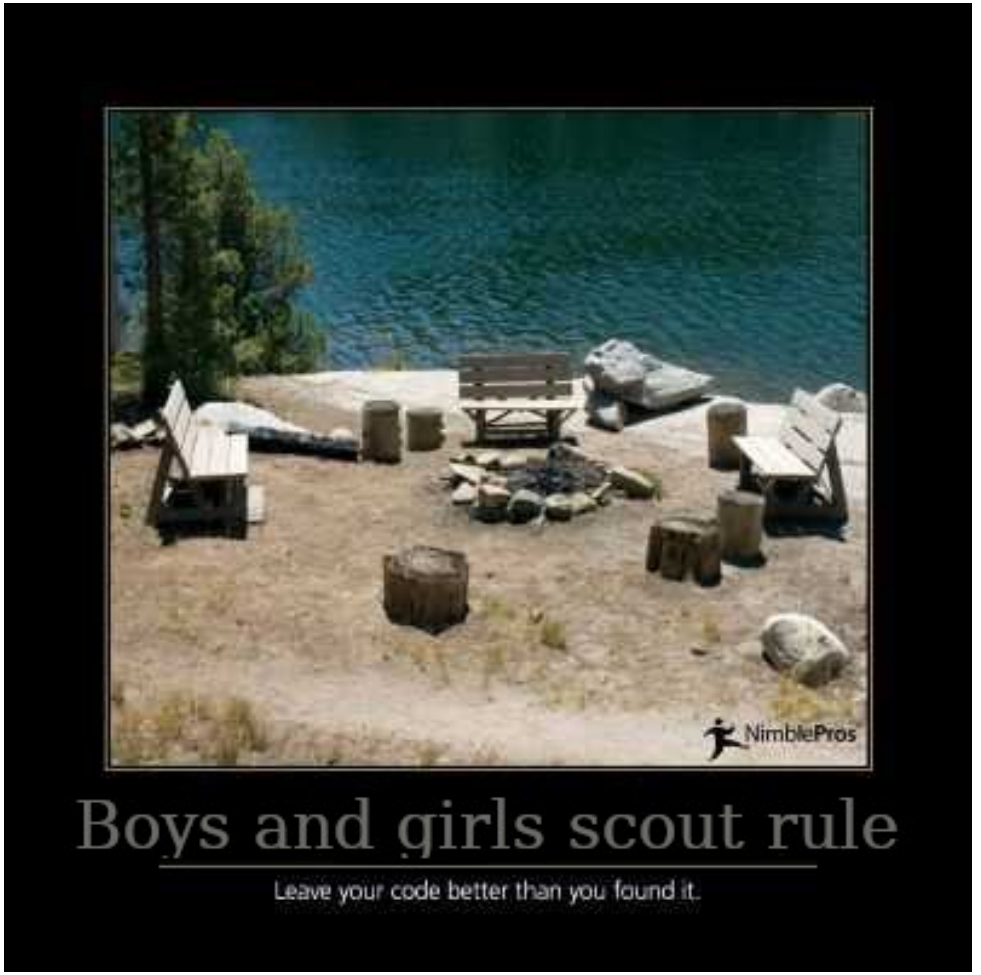
Ein paar objektive Daten



Quelle:https://codescene.com/blog/code-quality-debunking-the-speed-vs-vs-quality-myth-with-empirical-data

Wer ist verantwortlich?

- Es ist eine Entscheidung sich verantwortlich zu fühlen.
- Pfadfinder*innen-Regel: “Leave your code better than you found it”
- Schätzung sollten so gemacht werden, dass sie es zulassen sauber zu coden.



Grundlegende Prinzipien

- SOLID
  - Single Responsibility Principle: Es sollte nie mehr als einen Grund geben, eine Klasse zu modifizieren.
  - Open–closed principle: "software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification".
- DRY: Don’t repeat yourself
- KISS: Keep it simple and stupid

Ein paar generelle Regeln

- Komponenten in überschaubarer Größe
  - Reduzierung der zyklomatischen Komplexität
- Prinzip der geringsten Überraschung
- Verwendung von Abstraktionsebenen und Code auf der richtigen Ebene

Naming

- Aussagekräftige Namen
- Aussprechbare und Suchbare Namen
- Codierungen vermeiden
- Ein Wort pro Konzept und ein Konzept pro Wort

Zur Vertiefung: Ottinger's Rules for Variable and Class Naming

Funktionen

> “ The first rule of functions is that they should be small. The second rule of functions is that they should be smaller than that. ”

- Klein! Klein! Klein!
- (Selten mehr als 20 lines, niemals mehr als 100 lines)
- Eine Aufgabe erfüllen
- Eine Abstraktionsebene erfüllen

```
public void pay() {
    for (Employee e : employees) {
        if (e.isPayday()) {
            Money pay = e.calculatePay();
            e.deliverPay(pay);
        }
    }
}
```

```
public void pay() {
    for (Employee e : employees)
        payIfNecessary(e);
}

private void payIfNecessary() {
    if (e.isPayday())
        calculateAndDeliverPay();
}

private void calculateAndDeliverPay() {
    Money pay = e.calculatePay();
    e.deliverPay(pay);
}
```

Kommentare

- Kommentare sind Lüge die darauf warten, wahr zu werden.
- Code sollte selbsterklärende sein.
- Erkläre nicht was passiert, sondern höchstens warum.
- Wannimmer du einen Kommentar schreiben willst, gehe einen Schritt zurück und überlege, wie du den Code selbsterklärender machen könntest

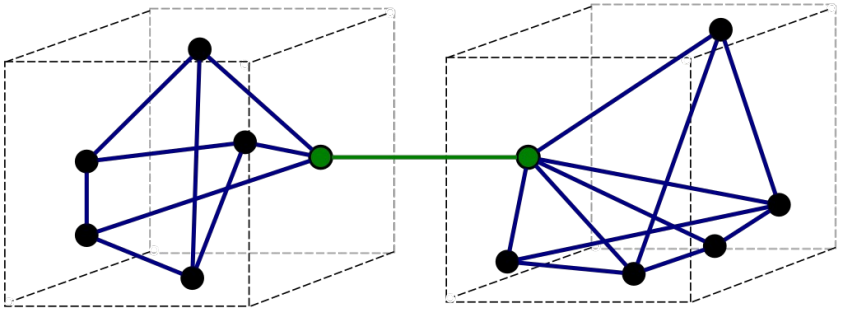
```
if (base64Code.length % 4 == 0) {
  // if it has a valid length for base64 encoded string
  console.log('lets decode the string')
}
```

```
const isLengthValid :boolean  = ba
if (isLengthValid) {
  console.log('lets decode the s
}
```

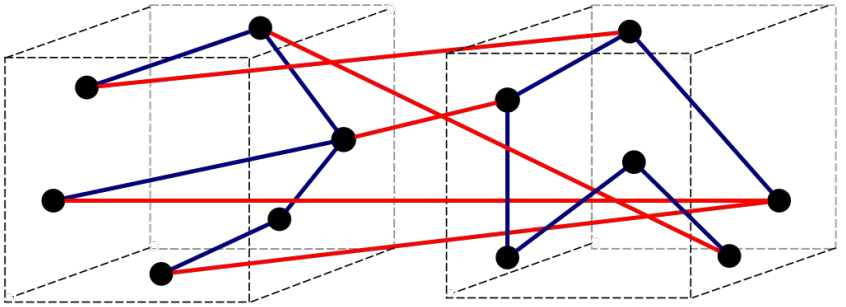
```
const hasValidLength = (base64: string) :boolean  => {
  return base64.length % 4 == 0
}

if (hasValidLength(base64Code)) {
  console.log('lets decode the string')
}
```

Klassen

- Klein, Kleiner!
- Single responsibility principle
- High cohesion
- Low coupling



a) Good (loose coupling, high cohesion)



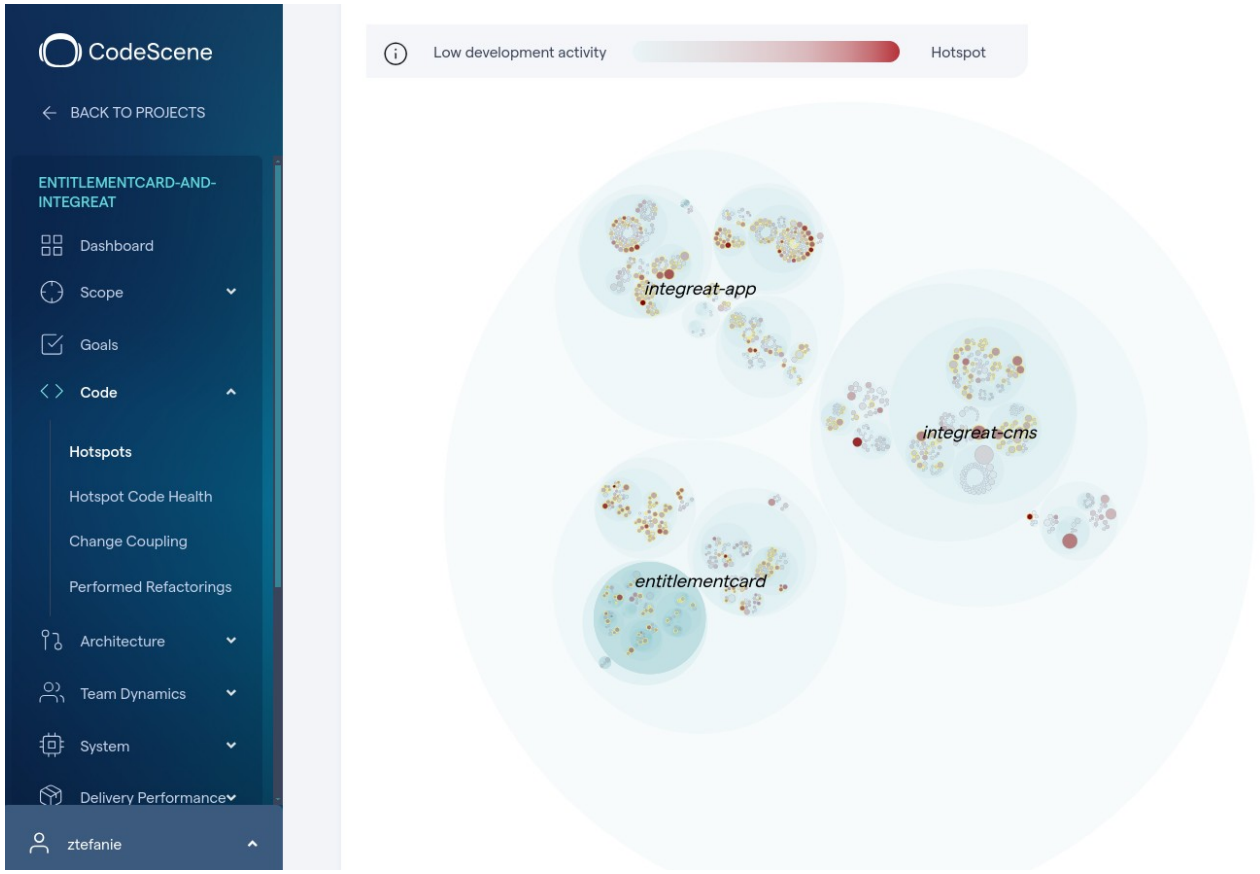
b) Bad (high coupling, low cohesion)

Smells und Heuristiken

- Redundante Kommentare
- Flag Argumente
- Nutze switch-case mit Vorsicht. Vermeide Duplizierung der selben switch-case Anweisungen, nutze Polymorphismus
- Code auf der falschen Abstraktionsebene
- Halte dich an Konventionen (Style guide)
- Ersetze magische Zahlen mit benannten Konstanten
- Kapsele komplexe Bedinungen
- Eine Funktionen sollte eine Sache machen
- Schreibe Tests
- Überprüfe die Nachbarschaft von bugs

Automatic Code Health Check

- Check out: https://codescene.com/



4 rules of simple design

- von Kent Beck
- In dieser Reihenfolge
  - Regel 1: Passes the Tests
  - Regel 2: Maximizes clarity
  - Regel 3: Minimizes duplication
  - Regel 4: Fewest Elements

Quellen & Diskussion

- Books:
  - Clean Code by Robert Martin
  - Pragmatic Programmer by Andy Hunt and David Thomas
  - Refactoring by Martin Fowler
- Cheatsheet: https://www.planetgeek.ch/wp-content/uploads/2014/11/Clean-Code-V2.4.pdf
- Articles:
  - https://exelearning.org/wiki/OttingersNaming/
  - https://martinfowler.com/bliki/BeckDesignRules.html