STEADFORCE

Wie schafft man es, in einem historisch gewachsenen Projektumfeld das Bewusstsein für Clean Code zu erhöhen?

David Cole

Thomas Werner

Wir...

David Cole

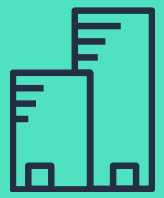
SCRUM Master

Thomas Werner

Senior Entwickler

... sind von Steadforce

Wir sind ein Full-Service Partner in der Entwicklung von digitalen Produkten und bei der Umsetzung von digitalen Geschäftsprozessen.



100+ Mitarbeiter in München, Timişoara (Rumänien)

Seit über 30 Jahren inhabergeführt

Clean Code

- Warum?
- Wie?

Steadforce Stand 2016

- Langjährige, zuverlässige Mitarbeiter
- Viel Fachwissen
- Erfolgreiche Projekte
- Zufriedene Kunden
- Firma ist profitabel

Die Welt ist in Ordnung!

Oder gibt es etwas zu verbessern?

Wie sah es aus?

Clean Code ist eine gute Idee, aber dafür haben wir keine Zeit.

Bei uns haben wir langjährige Mitarbeiter – es reicht, wenn sie den Code verstehen.

Wie sah es aus?

Unsere Kunden sind nicht bereit, mehr Geld für Code Reviews oder Clean Code zu bezahlen.

Wir sind ein Dienstleistungsunternehmen – wir entwickeln, was und wie es der Kunde haben möchte.

Wie sah es aus?

Unser Controlling überwacht, dass unsere Projekte in Time & Budget fertig werden.

Unser Code ist historisch gewachsen – ein Refactoring für Clean Code käme fast einer Neuentwicklung gleich.

Wie sah es aus?

Ich bin der einzige Entwickler im Projekt – es gibt niemanden, der Code Reviews machen kann.

Wir sind in der Wartungsphase: Alle Anforderungen müssen immer vorgestern fertig gewesen sein – ich bin froh, wenn die Änderungen rechtzeitig funktionieren.

Wie sah es aus?

Bei unserem Projekt machen wir Code Reviews und überprüfen Code Qualität, Test Coverage und Lesbarkeit.

Bei uns bedeutet fertig wirklich fertig – dafür wird die „Definition of Done“ gelebt.

Steadforce – Stand 2016 - Zusammenfassung

Viele Projekte sind erfolgreich.

Die Anzahl von Bugs in den Produkten ist gering.

Einige Installationen liefen jedoch nicht ganz reibungslos.

Clean Code wird in den Projekten nicht überall gelebt.

Es gibt zu wenig Austausch zwischen den Projekten.

2016: Entstehung des CAMPUS bei Steadforce

Aufgaben:

- Qualität verbessern
- Best Working Practices implementieren
- Mehr Austausch zwischen den Projekten

- Technische Strategie der Firma definieren
- Trends in der IT verfolgen

Maßnahmen um Software Entwicklung zu verbessern ..

- Agil
- SCRUM
- Code Reviews
- Clean Code

*** Best Working Practices ***

Aufgaben

Management überzeugen

Kollegen überzeugen

Prozesse ändern

Management überzeugen

Qualität und Kundenzufriedenheit sind Unternehmensziele

Fragen an das Management:

- Was ist Qualität?
- Was passiert wenn wir nicht auf Qualität achten?

Management überzeugen

Welche Projekte legen am meisten Wert auf Clean Code/ Code Reviews ?

Welche Projekte verdienen auf lange Sicht Geld ?

- Qualität ist Voraussetzung für nachhaltige Rendite
- Qualität ist Voraussetzung für Kundenzufriedenheit

Mitarbeiter überzeugen

Vorträge

Coachings

Zeit investieren

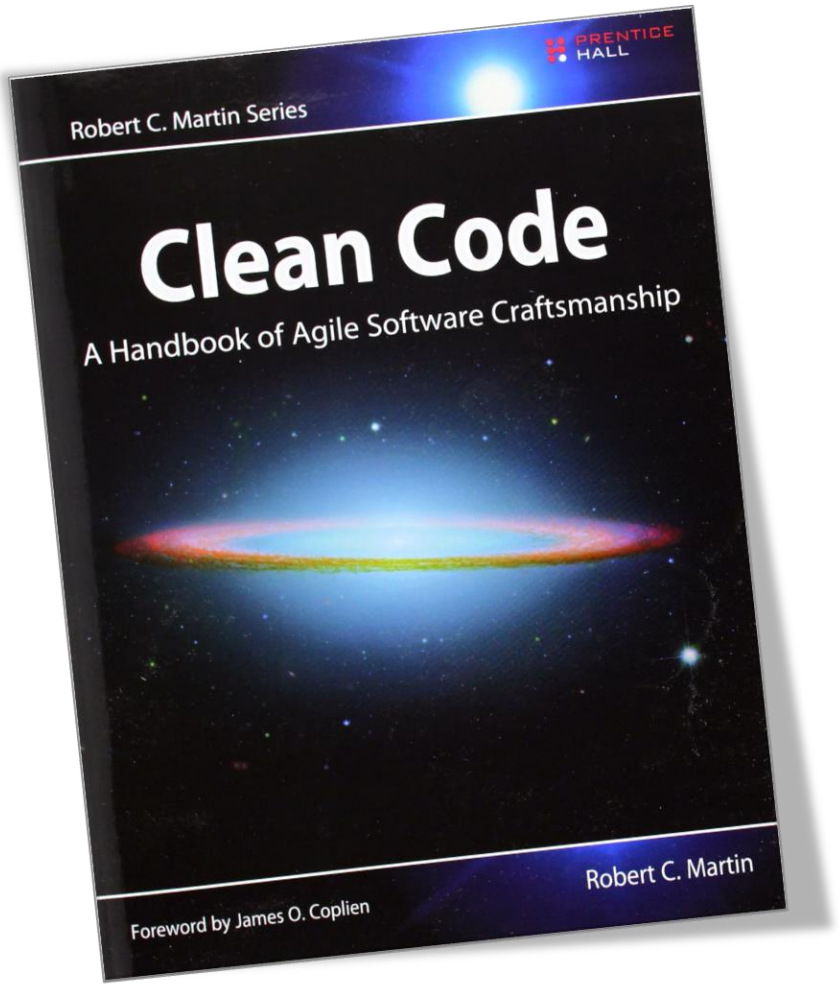
Clean Code Vortrag

- Einmalig für alle Mitarbeiter
- Als Teil des Onboarding für neue Mitarbeiter
- Ziele:
  - Gemeinsames Verständnis darüber, wie Clean Code aussehen soll
  - Awareness für die Vorteile schaffen
  - Machbarkeit zeigen

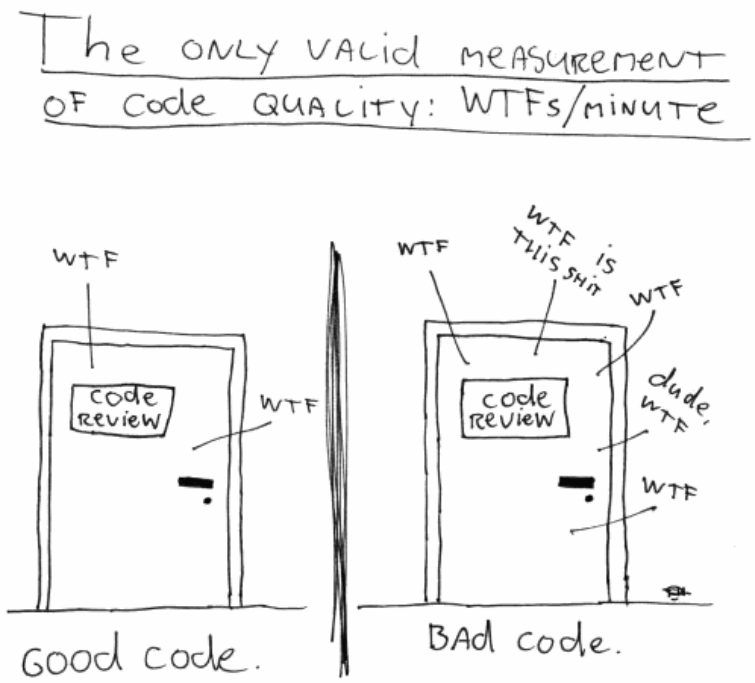
Clean Code Vortrag

Teil 1 – Theorie

Teil 2 - Praxis


Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

Part 1: How to write good code



Meaningful Names

Class Names

```
Customer, WikiPage, Account, AddressParser
// should be nouns
```

Method Names

```
postPayment, deletePage, save
// should have verb or verb phrase names

Complex fulcrumPoint =
Complex.fromRealNumber(23.0);
// is better than
Complex fulcrumPoint = new Complex(23.0);
```

Meaningful Names

Pick one word per concept

```
fetch, retrieve, get
// equivalent methods

controller, manager, driver
// confusing
```

Don't use type information in names

```
Integer phoneString;
// name not changed when type changed!
```

Meaningful Names

Variable, function or class names should answer:

–Why it exists

–What it does

–How it is used

Intention-Revealing Names

Searchable Names

Good names may be long

It may take some time to choose a good name

Functions

should be

- small
- do one thing

Code should be readable from top to bottom (stepdown rule)

Should have descriptive names

Functions

Should not have side effects

Don’t repeat yourself

Structured Programming (one entry, one exit)

Functions

One level of abstraction per function

```
// high level of abstraction
getHtml();

// intermediate level of abstraction
String pagePathName =
PathParser.render(pagePath);

// remarkably low level
.append("\n")
```

Functions

Switch statements

```
public int calculatePay(Employee e) throws
InvalidEmployeeType {
   switch (e.getType()) {
      case SALARIED:
         return calculateSalariedPay(e);
      case HOURLY:
         return calculateHourlyPay(e);
      default:
         throw new
InvalidEmployeeType(e.getType());
   }
}
```

Functions

Switch statements

```
public abstract class Employee {
   public abstract int calculatePay();
}

public class EmployeeFactroyImpl implements
EmployeeFactory {
   public Employee
makeEmployeeType(EmployeeRecord r)
         throws InvalidEmployeeType {
      switch (r.getType) {
         case SALARIED:
            return new SalariedEmployee(r);
         case HOURLY:
            return new HourlyEmployee(r);
         default:
```

Function Arguments

Functions should have as few arguments as possible

Reasons for Arguments

```
// asking a question about an argument
boulean fileExists(“MyFile”);

// transform / operate on the argument
InputStream fileOpen(“MyFile”);

// alter the state of the system
void passwordAttemptFailedNtimes(int attempts);
```

Function Arguments

Try to avoid flag arguments

```
render(true);

//better:
renderForSuite();
renderForSingleTest();
```

Comments

–Don‘t use a comment when you can use a function or a variable

–Comments do not make up for bad code

–Don’t comment bad code - rewrite it.

Comments

Good Comment

- Legal Comments (e.g. copyright)
- Clarification(e.g. for a RegEx)
- Warning of consequences (e.g. this test case may take very long)
- TODO comment
- JavaDocs in public APIs

Comments

Bad Comments

–Redundant comments (e.g. for getter / setter)

–Noise comments (e.g. // Default constructor)

–Commented-Out Code

–HTML Comments

–Javadocs in Nonpublic Code

Formatting

- Blank lines separates concepts
- Vertical distance
  - Variables should be declared close to their usage
  - Instance variables should be declared at the top of the class
  - Functions that call each other should be vertical close (caller above called)

Objects and Data Structures

Data / Object Anti-Symmetry

–Objects hide their data behind abstractions and expose functions that operate on that data

–Data structures expose their data and have no meaningful functions

Objects and Data Structures

Law of Demeter

–Method f of Class C should only call methods of

– C

– An objects created by C

– An object passed as an argument to f

– An object held in an instance variable of C

Error Handling

- Prefer exceptions to returning error codes
- Don't return/ pass null

Unit Tests

Test-Driven-Development

–Write the unit test before you write production code

–Don’t write more than one unit test that is sufficient to fail

–Don’t write more production code than is sufficient to pass the failing test

One single concept per test

Classes

Class organization

–Public static constants

–Private static variables

–Private instance variables

–Public functions

–Private utility functions called by public functions

Classes should be small

Single Responsibility Principle

Summary

Boy Scout Rule

–Leave the code better than you found it

The newspaper Metaphor

–Headline

–Synopsis

–details

Clean Code Vortrag

Teil 1 – Theorie

Teil 2 – Praxis

Wie mache ich aus schlechtem Code guten Code?

Mitarbeiter überzeugen

Vorträge

Coachings

Zeit investieren

Prozesse ändern

Neuer Softwareentwicklungsprozess definiert

- (Code) Review als Phase mit folgenden Zielen:
  - Wissenstransfer
  - Verbesserung der Code Qualität
  - Bessere Wartbarkeit
  - Sicherstellung der Funktionalität

Wie sieht es zwei Jahre später aus…

… für die Entwickler

Clean Code ist kein Ersatz für sorgfältiges Arbeiten

Clean Code alleine führt nicht dazu, dass jeder Junior Entwickler sofort perfekt arbeitet

… für die Entwickler

Keine Diskussion mit dem Management über Aufwände für Code Reviews und Refactoring

Die Entwickler glauben an Clean Code

- Es wird über Clean Code geredet!
- Code Reviews werden als normal angesehen
- Feedback wird positiv und nicht als Kritik empfunden

Unsere Qualität ist besser geworden

- Junior Entwickler kommen schneller an Bord
- Senior Entwickler, die immer gute Qualität geliefert haben, tun es weiterhin ;-)

… für das Management

- Weiterhin Projekte, die nicht in Time & Budget fertig werden
- Weiterhin Herausforderungen mit altem Code

Steadforce – Stand 2019

- Steadforce wird als Premium Supplier gesehen
- Unsere Prozesse sind ein Pluspunkt beim Recruiting
- Steadforce ist profitabler geworden

Gesamtfazit

- Clean Code und Code Reviews können in jeder Firma eingeführt werden
- Zuerst muss das Management überzeugt werden
- Die Änderung muss von den Entwicklern für die Entwickler gemacht werden
- Dazu ist ein zentrales Forum wie unser Campus Team erforderlich

STEADFORCE

FRAGEN?

Don‘t hesitate to get in touch with us.

Steadforce GmbH
Germany

Westendstraße 193
80686 München

+49 89 51727 0
munich@steadforce.com
www.steadforce.com

SC Steadforce RO SRL
Romania

Str. Piața Consiliul Europei Nr. 2A
Timișoara 300627

+40 356 716 098
contact.tsr@steadforce.com
www.steadforce.com