

# CLEAN CODE DEVELOPER

PROFESSIONALITÄT = BEWUSSTHEIT + PRINZIPIEN

Felix Ziesel und Markus Tiede

# AGENDA

- Motivation / Idee
- Wertesystem(e)
- Prinzipien und Praktiken
- der rote Grad
- der orange Grad
- der gelbe, grüne und blaue Grad
- Fazit / Ausblick

# MOTIVATION / IDEE

Die Branche braucht einen Qualitätsmaßstab oder zumindest einen Erwartungshorizont für Professionalität.

Wir haben einen "Haufen"... junger Mitarbeiter!

# WERTESYSTEM(E)

- E - Evolvierbarkeit  
Anpassungsfähige Struktur
- K - Korrektheit
- P - Produktionseffizienz  
Angemessene Arbeitsweise
- R - Reflektion  
Kontinuierliche Verbesserung

# PRINZIPIEN UND PRAKTIKEN

grundlegenden Gesetzmäßigkeiten

*“Ob ein Prinzip eingehalten wurde, kann man dem Code immer ansehen.”*

handfeste Handlungsanweisungen

*“Tue es immer so. Jeden Tag, jederzeit.”*

# PRINZIPIELLES

- VV - **V**alue **V**ariation
- DOWN - **D**o **O**nly **W**hat's **N**eccessary
- IA - **I**solate **A**spects
- MD - **M**inimize **D**ependencies
- HP - **H**onor **P**ledges



# PRAKTISCHES

- EU - **E**mbrace **U**ncertainty
- F - **F**ocus
- VQ - **V**alue **Q**uality
- GTD - **G**et **T**hings **D**one
- SC - **S**tay **C**lean
- KM - **K**eeP **M**oving

# DIE GRADE





# DER ROTE GRAD

Don't Repeat Yourself (DRY)



Keep It Simple, Stupid (KISS)



Vorsicht vor Optimierungen



Favour Composition over Inheritance (FCol)



Pfadfinderregel



Root Cause Analysis



Versionskontrolle



Einfache Refaktorisierungen



Täglich reflektieren



# DON ´T REPEAT YOURSELF (DRY)

*“Jeder Strg+C sollte einen inneren Alarm auslösen”*

- Vermeiden von Wiederholungen
- Beseitigen von Wiederholungen
- E++, K++, P+, R+

# KEEP IT SIMPLE, STUPID (KISS)

*“Alles sollte so einfach wie möglich gemacht werden, aber nicht einfacher.”*

- Eine einfache Lösung ist stets vorzuziehen.
- Nicht Strukturen für die Zukunft planen, die eventuell nie kommt.
- E++, K++, P+, R

# VORSICHT VOR OPTIMIERUNGEN

*“ Rules of Optimization:*

*Rule 1: Don't do it.*

*Rule 2 (for experts only): Don't do it yet ”*

- Kosten/Nutzen nicht unbedingt gewährleistet.
- Code kann dadurch umständlich werden.
- E++, K, P++, R

# FAVOUR COMPOSITION OVER INHERITANCE

- "has a"-Beziehungen niemals mit Vererbung realisieren
- Fördert die Entkopplung von Klassen
- E+, K, P, R



# PFADFINDERREGEL

*“Hinterlasse einen Ort immer in einem besseren Zustand als du ihn vorgefunden hast.”*

- Große Refactorings in der Regel nicht machbar
- Vorgehen in kleinen Schritten
- Mittel gegen den "Broken-Window"-Effekt
- E++, K, P, R



# ROOT CAUSE ANALYSIS

*“Beseitige die Wurzel eines Problems nicht dessen Symptom.”*

- Dienst an der Verständlichkeit und am Aufwand
- Ansonsten: Ein Workaround für einen Workaround für einen Workaround ...
- Five Why's
- E+, K, P, R++

# VERSIONSKONTROLLE

*“Erste Voraussetzung für den Einstieg ins Clean Code Development ist der ständige Gebrauch eines Versionskontrollsystems!”*

- nimmt die Angst, etwas falsch und damit kaputt zu machen.
- E, K, P++, R+

# EINFACHE REFAKTORISIERUNGEN

- Methode/Test Case extrahieren
- Umbenennen
- E++, K, P, R+

# TÄGLICH REFLEKTIEREN

*“Keine Verbesserung, kein Fortschritt, kein Lernen ohne Reflexion.”*

- Abendliche Bewertung: Habe ich alle meine Aufgaben erledigt?  
Wie habe ich meine Aufgaben erledigt?
- Wechsel des Armbands zum anderen Arm bei Verbesserungsbedarf
- Nach 21 Tagen ohne Wechsel neuer Grad
- E, K, P, R++

# DER ORANGE GRAD

Single Level of Abstraction (SLA)



Single Responsibility Principle (SRP)



Separation of Concerns (SoC)



Sourcecode-Konventionen



Issue Tracking



Automatisierte Integrationstests



Lesen, Lesen, Lesen



Reviews





# SINGLE LEVEL OF ABSTRACTION (SLA)

*“Bitpfriemeleien nicht mit Methodenaufrufen  
mischen...”*

- Methoden sollen genau ein Abstraktionsniveau besitzen
- "Zeitungs"-analogie
  - Überschrift = Klassenname
  - Untertitel = public methods
  - Inhalt = private methods
- E++, K+, P, R



# SINGLE LEVEL OF ABSTRACTION (SLA)

```
public void performAllOperations() {  
    gatherData();  
    m_valid = !((!false && true) || !globalValid);  
    validateData();  
    m_valid = !m_valid;  
    alterData();  
    persistData();  
}
```

```
private void gatherData() {}  
private void validateData() {}  
private void alterData() {}  
private void persistData() {}
```

FALSCH

# SINGLE LEVEL OF ABSTRACTION (SLA)

```
public void performAllOperations() {  
    gatherData();  
    validateData();  
    alterData();  
    persistData();  
}  
  
private void gatherData() {}  
private void validateData() {  
    m_valid = !((!false && true) || !globalValid);  
    ...  
    m_valid = !m_valid;  
}  
private void alterData() {}  
private void persistData() {}
```

**RICHTIG**

# SINGLE RESPONSIBILITY PRINCIPLE (SRP)

*“Eine Klasse sollte nur eine Verantwortlichkeit haben.”*

- Änderungen / Erweiterungen sollen sich auf wenige Klassen beschränken lassen
- Verstoß führt zu
  - hoher Kopplung von Klassen
  - hoher Komplexität bei Änderungen
- eines der **SOLID** Prinzipien
- E++, K+, P, R

# SINGLE RESPONSIBILITY PRINCIPLE (SRP)

```
public class EierLegendeWollMilchSau {  
  
    public void legeEier() {}  
    public void habFell() {}  
    public void produziereMilch() {}  
    public void seiFett() {}  
}
```

**FALSCH**

# SINGLE RESPONSIBILITY PRINCIPLE (SRP)

```
public class Huhn { public void legeEier() {} }  
  
public class Schaf { public void habFell() {} }  
  
public class Kuh { public void produziereMilch() {} }  
  
public class Schwein { public void seiFett() {} }
```

**RICHTIG**

# SEPARATION OF CONCERNS (SOC)

*“Trennung der Belange - a.k.a. komplett verschiedener Zwecke.”*

- Belange stehen häufig orthogonal zu einander
- Beispiel für "Belange": Logging, Persistenz, Darstellung
- ähnelt Single Responsibility Principle  
Responsibility  $\subset$  Concerns
- Ergebnis
  - lose Kopplung » gute Testbarkeit + Wiederverwendbarkeit
  - enges Zusammenspiel von Attributen und Methoden einer Klasse (hohe Kohäsion) » lokale (große) Änderungen möglich
- E++, K+, P, R



# SEPARATION OF CONCERNS (SOC)

Business Domain  $\Leftrightarrow$  Persistenzinfrastruktur

Geschäftslogik  $\Leftrightarrow$  Datenbankzugriffe

*“Persistenz ist ein "Concern" der nichts mit der Business Logik zu tun hat.”*

Problem: häufig sind in einer Responsibility verschiedene Concerns vermischt.

mögliche Lösung:  
Aspektorientierten Programmierung (AOP)

# SOURCE CODE KONVENTIONEN

*“Code wird häufiger gelesen als geschrieben.”*

- BREDEX Code Conventions
- CCD
  - Namensregeln und deren konsequenter Einsatz
  - "richtige" Kommentare:

```
int laenge; // in mm
```

```
int laengeInMM;
```

# SOURCE CODE KONVENTIONEN

oder auch

```
public double Preis() {  
    // Berechnet den Bruttopreis ...  
}
```

```
public Money BruttoPreis() {...}
```

E+, K+, P(+), R

# ISSUE TRACKING

*“Nur, was man aufschreibt, vergisst man nicht und kann man effektiv delegieren und verfolgen.”*

schafft Bewusstsein und Überblick

ermöglicht Priorisierung

BREDEX: trac + bugzilla

Tätigkeit steht über Tool!

E+, K+, P++, R+

# AUTOMATISIERTE INTEGRATIONSTESTS

*“Wiederkehrende Tätigkeit nicht zu automatisieren wäre Zeitverschwendung.”*

Schafft "Sicherheitsnetz" + Regression

Anzuwenden noch vor Unit Tests

Grund: Unit Tests fordern eingehaltenes SRP!

E++, K++, P++, R

# LESEN, LESEN, LESEN

*“Lesen bildet!”*

Ständige Fortbildung um "Schritt" zu halten

Empfehlung: mind. 6 Fachbücher pro Jahr

+ regelmäßiges Lesen von Fachzeitschriften und Blogs

BREDEX: große Bibliothek + Zeitschriftenfundus

E, K, P, R+



# REVIEWS

*“Vier Augen sehen mehr als zwei.”*

Pair Programming oder Code Review

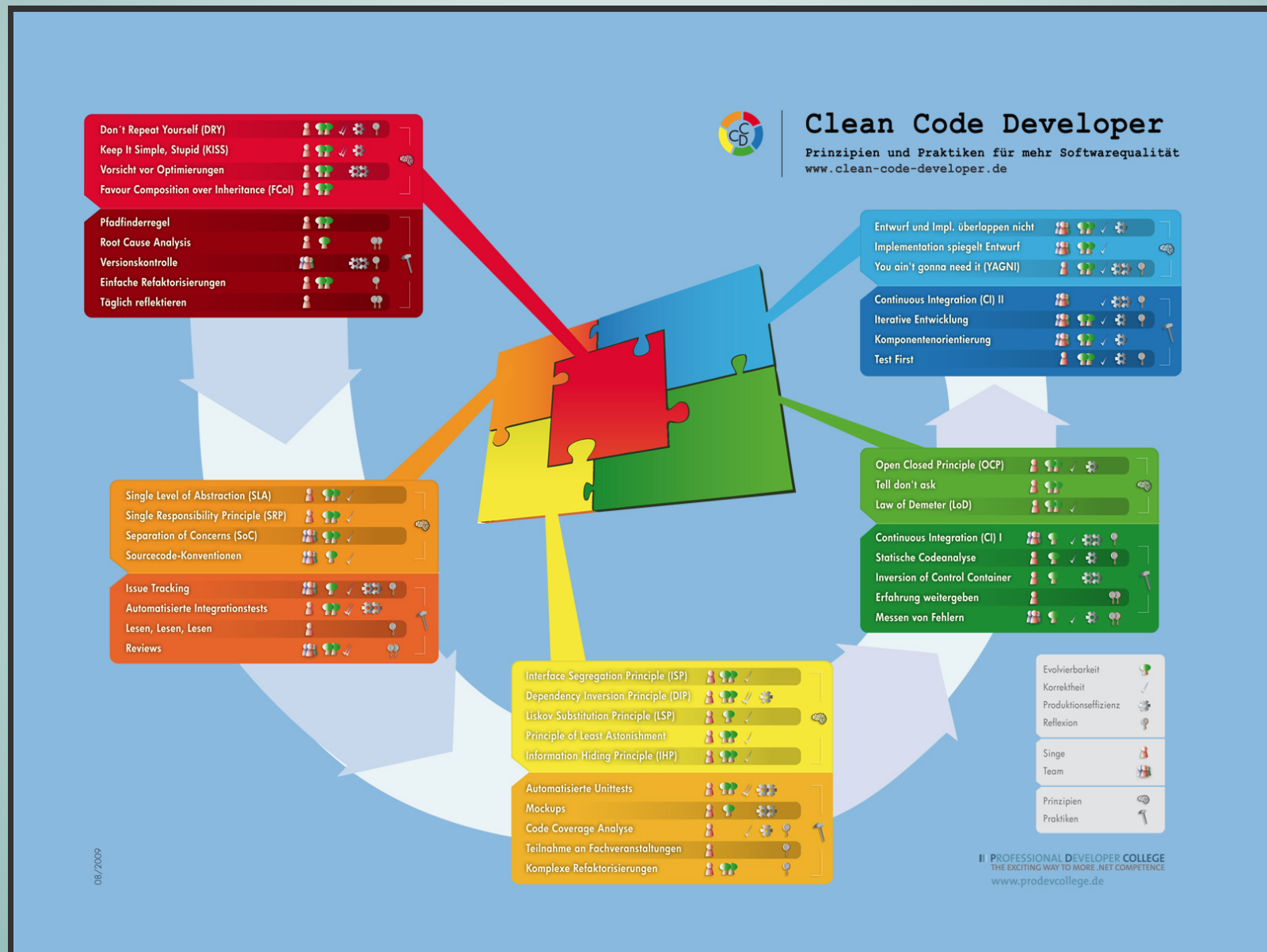
Code diskutieren und reflektieren

sehr frühes finden von Fehlern möglich

führt zu ständiger Verbesserung auf beiden Seiten

E++, K++, P, R++

# DER GELBE, GRÜNE UND BLAUE GRAD



# DER SCHWARZE UND WEISSE GRAD

## SCHWARZ

- signalisiert Interesse
- Schaffen von Voraussetzungen

## WEISS

- Ende eines Durchlaufs
- Neubeginn mit dem roten Grad

# FAZIT / ERFAHRUNGEN

Schärft die Aufmerksamkeit

Roter Grad ermöglich "sanften" Einstieg

Projekt muss es "erlauben", da höherer Aufwand (ab Orange)!

Man "fühlt" sich besser...

Für Tester nur bedingt nutzbar

# AUSBlick

Wen interessiert das Thema CCD?

Soll es einen weiteren Vortrag zu den anderen Graden geben?

Wer interessiert sich für das Wertesystem des CCD?

Welchen Grad hat euer Projekt?

# REFERENZEN

Website [clean-code-developer.de](https://clean-code-developer.de)

Buch: Clean Code: A Handbook of Agile Software Craftsmanship  
(Robert C. Martin)

**Literaturliste**