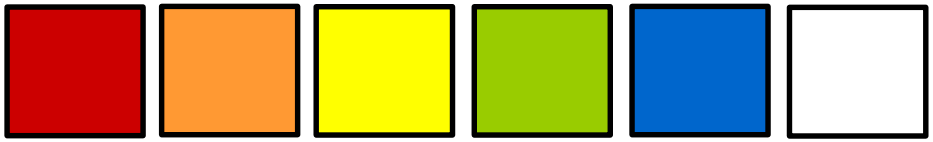


Stufe 6: Weißer Grad von CCD

Lutz Prechelt
Institut für Informatik, Freie Universität Berlin

Prinzipien und Praktiken

- Der weiße Grad umfasst die **Vereinigungsmenge** aller Prinzipien und Praktiken aus allen vorherigen Graden

- Wir schauen die nochmal schnell an, diesmal in *thematischen* Gruppen:

Freie Universität Berlin

2. Tue nur das Nötigste

- Keep it simple, stupid
- Vorsicht vor Optimierungen!
- You Ain't Gonna Need It

3. Isoliere Aspekte

- Don't Repeat Yourself
- Separation of Concerns
- Single Level of Abstraction
- Single Responsibility Prin.
- Interface Segregation Prin.
- Entwurf und Implementation überlappen nicht

4. Minimiere Abhängigkeitn

- Dependency Inversion
- Information Hiding
- Law of Demeter
- Open Closed Principle
- Tell, don't ask
- Interface Segregation Prin.

5. Halte Versprechen ein

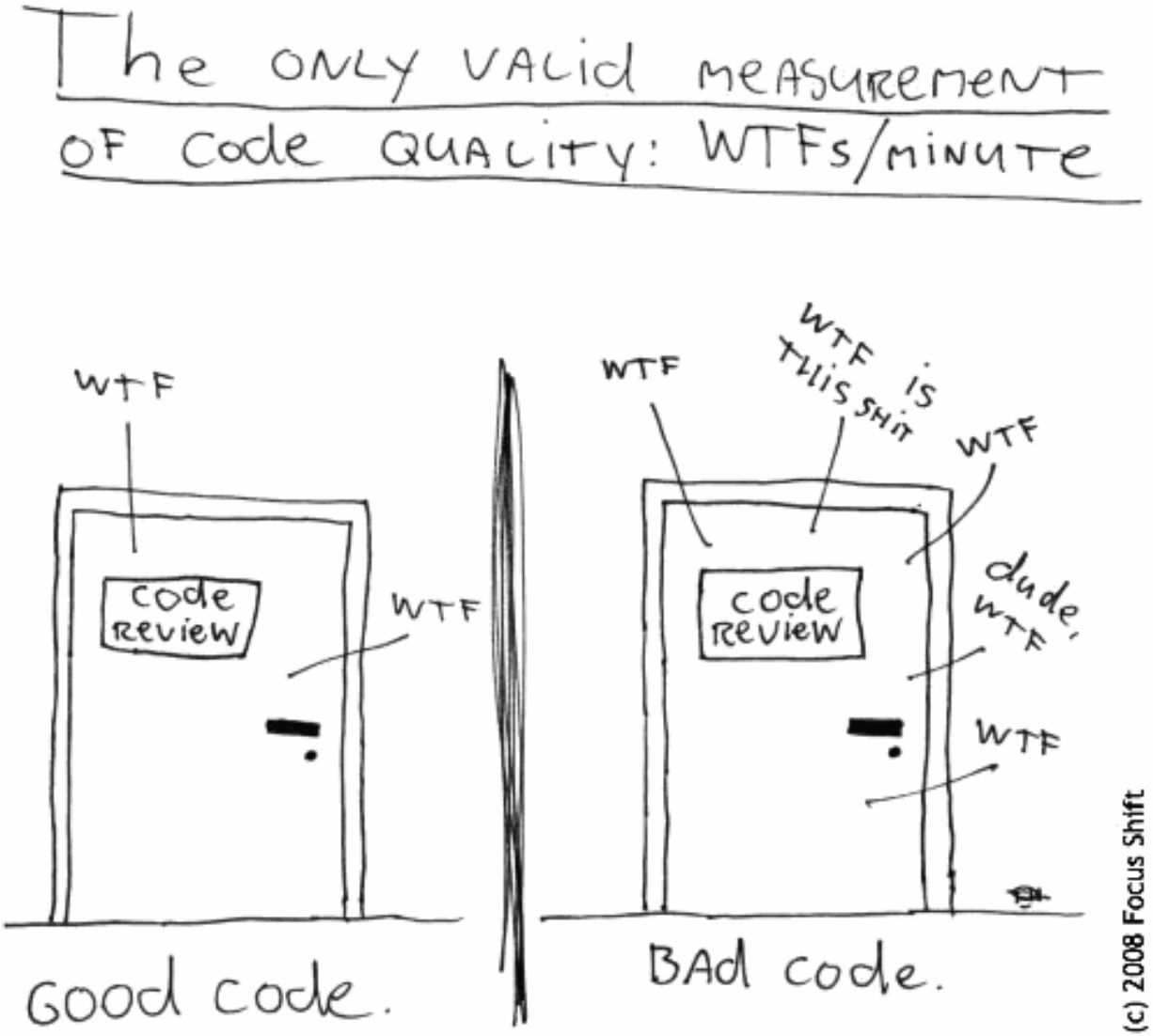
- Liskov Substitution Principle
- Prin. of Least Astonishment
- Implementation spiegelt Entwurf
- Favour Composition over Inheritance

Was ist eigentlich "Clean Code"?

- Bob Martin hat diverse hochkarätige Leute nach ihrer Definition gefragt:
  - Bjarne Stroustrup (→C++)
  - Grady Booch (→UML)
  - Dave Thomas
  - Michael Feathers
  - Ron Jeffries (→XP)
- Zusammengenommen kommen folgende Eigenschaften heraus [ClC, pp.7-13]:
  - Clean code is elegant, simple, efficient,
  - straightforward, crisp, clear, literate, readable by others,
  - unsurprising,
  - has minimal and explicit dependencies,
  - has automated tests,
  - minimizes the number of classes and methods,
  - expresses its design ideas,
  - handles errors,
  - has nothing obvious that one could do to make it better,
  - looks like the author has cared.

Was ist eigentlich Clean Code: Verständlicher Code!

- Niedrige Werte der "What the fuck??"-Metrik



clean-code-developer.de: Was haben wir besprochen? Praktiken

Freie Universität Berlin

1. Hingabe an Ungewissheit

- Autom. Integrationstests
- Autom. Unit Tests
- Continuous Integration
- Mockups (Testattrappen)
- Inversion of Ctrl. Container
- Versionskontrolle einsetzen

2. Fokussiere

- Komponentenorientierung
- Test first
- Limit Work-in-Progress

3. Wertschätze Qualität

- Reviews
- Issue Tracking

4. Mach fertig

- Iterative Entwicklung
- Continuous Delivery

5. Halte Ordnung

- Source Code Konventionen
- Statische Codeanalyse
- Code Coverage Analyse
- Pfadfinderregel beachten
- Refaktorisierungen

Freie Universität Berlin

6. Bleib am Ball

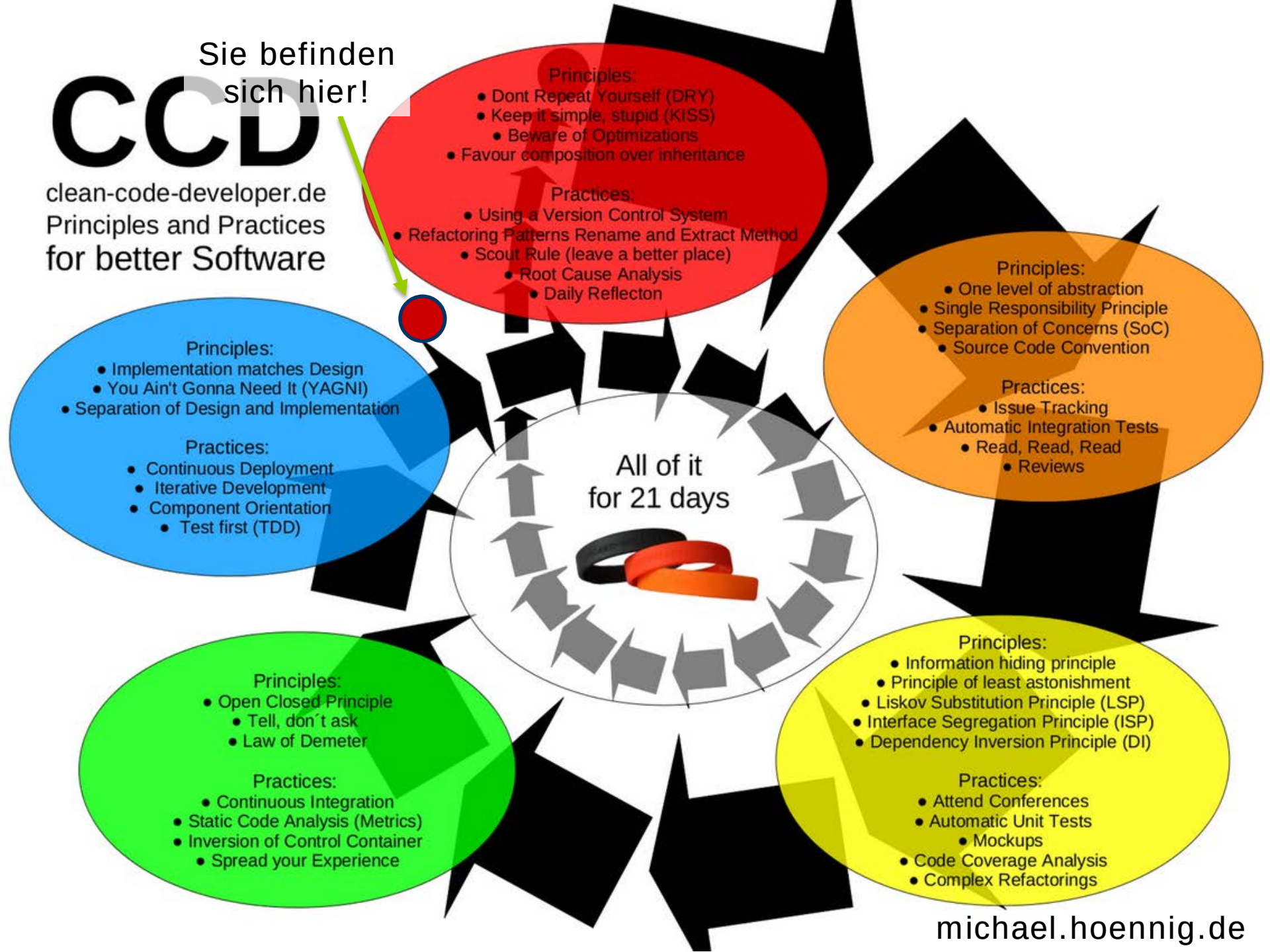
- Lesen, Lesen, Lesen
- Teilnahme an Fachveranstaltungen
- Erfahrung weitergeben
- Täglich reflektieren
- Root Cause Analysis
- Messen von Fehlern

- (Ein paar der Punkte passen in mehr als eine Kategorie)

Beurteilung

- Das ist ziemlich viel Material!
- Zumal an den meisten Stellen viele der Entwurfsprinzipien zugleich zu beachten sind
- Deshalb ist das schrittweise Lernen mit Graden sinnvoll
- Nochmal die Idee:
  - Jeden Tag die Einhaltung der Prinzipien & Praktiken des *aktuellen* Grades reflektieren
  - Nach 21 Tagen guter Einhaltung zum nächsten Grad übergehen

- Und wie gängig ist die Einhaltung bislang?
  - Einschätzungen 2014 in Ralf Westphals Forum auf XING: nur eine klare Minderheit der Entwickler tut's/kann's


Sie befinden sich hier!
CCD
clean-code-developer.de
Principles and Practices
for better Software
Principles:
• Dont Repeat Yourself (DRY)
• Keep it simple, stupid (KISS)
• Beware of Optimizations
• Favour composition over inheritance
Practices:
• Using a Version Control System
• Refactoring Patterns Rename and Extract Method
• Scout Rule (leave a better place)
• Root Cause Analysis
• Daily Reflecton
Principles:
• One level of abstraction
• Single Responsibility Principle
• Separation of Concerns (SoC)
• Source Code Convention
Practices:
• Issue Tracking
• Automatic Integration Tests
• Read, Read, Read
• Reviews
Principles:
• Implementation matches Design
• You Ain't Gonna Need It (YAGNI)
• Separation of Design and Implementation
Practices:
• Continuous Deployment
• Iterative Development
• Component Orientation
• Test first (TDD)
All of it
for 21 days
Principles:
• Open Closed Principle
• Tell, don´t ask
• Law of Demeter
Practices:
• Continuous Integration
• Static Code Analysis (Metrics)
• Inversion of Control Container
• Spread your Experience
Principles:
• Information hiding principle
• Principle of least astonishment
• Liskov Substitution Principle (LSP)
• Interface Segregation Principle (ISP)
• Dependency Inversion Principle (DI)
Practices:
• Attend Conferences
• Automatic Unit Tests
• Mockups
• Code Coverage Analysis
• Complex Refactorings
michael.hoennig.de

Zur Erinnerung: (aus Woche 1) Anforderungen an professionals

Freie Universität Berlin

aus Woche 1

- Beherrschung der Grundlagen ihres Fachs
  - Dafür ist das Studium gedacht!
- Tiefes Wissen in einem erkennbar abgegrenzten Teilbereich
  - Das erwirbt man überwiegend erst im Beruf
  - Evtl. formalisiert (z.B. Facharztausbildung)
- Kontinuierliche Fortbildung
  - Denn das Wissen aller Professionen entwickelt sich schnell fort
  - Das ist ein Anlass für die Existenz von Fachgesellschaften
    - bei uns z.B. ACM, IEEE CS, GI

für Lesetypen — für Konferenztypen

- Sorgfalt und verantwortliches Handeln
  - Denn die Klienten können sich kaum selbst schützen
  - und Software hat oft hohes Gefährdungspotential

Besonderheit der Informatik

Freie Universität Berlin

aus Woche 1

- Alle anderen Ingenieur/inn/e/n konzipieren überwiegend nur, aber setzen kaum um
  - dafür gibt es passende Handwerker/innen
- Die Informatik hat es noch wenig geschafft, Handwerksteile genügend gut abzugrenzen
  - deshalb sind wir überwiegend auch unsere eigenen Handwerker/innen
    - Ausnahmen z.B. beim Kunsthandwerk (web design etc.)
  - *"software craftsmanship"*
- Wir schließen in diesem Kurs handwerkliche Aspekte mit ein
  - Es geht also sowohl um Wissen
  - als auch um Geschick: Üben, üben, üben

Professionalität ist keine Fähigkeit, sondern eine Verhaltensweise

- Die bedrohten Anforderungen an professionelle Arbeitsergebnisse sind vielfältig

- Aber die häufigsten sind
  1. Geeignete Funktionalität
     - d.h. zweckmäßig
  2. Zuverlässigkeit der SW
  3. Sicherheit der SW
  4. Dauerhafte Fortentwickelbarkeit der SW

- Und die wichtigste Bedrohung ist meistens:
  5. Zeitmangel

aus Woche 1

- Wichtige Fähigkeiten sind deshalb:
  - Teamarbeit
    - wg. 1, 2, 3, 4, 5
  - Domäne verstehen
    - wg. 1, 2, 3
  - **Guter SW-Entwurf**
    - wg. 2, 3, 4, 5
  - **Disziplin**
    - wg. 2, 3, 4, 5
  - Verhandeln
    - wg. 5

Verhandeln

- Entwickler/innen dürfen unsinnigen Zeitvorgaben nicht nachgeben
  - Beispiel T (lokal)
    - Projektleiter Mike und Entwicklerin Paula
- sondern müssen Umfänge verhandeln
  - Beispiel L (lokal)
    - wieder Mike und Paula
- ebenso das höhere Management
  - und auch, wenn viel auf dem Spiel steht:
  - Beispiel M (lokal)
    - Entwicklungsleiter Don und Firmenchef Chuck

- Wenn eine der Stel… versagt, folgt Unbill
  - Beispiel B, Szene 1 (lokal)
    - Projektleiter Mike (oje!) und Entwicklerin Paula (OK)
      - Das ist Betteln, nicht Verhandeln
  - Beispiel B, Szene 2 (lokal)
    - Projektleiter Mike (ojeoje!) und Entwicklungsleiter Don
      - Das ist Verleugnen.
    - Wer ist hier "Teamplayer"?

aus Woche 1

(Szenen aus "Clean Coder", Kapitel 2)

Ziele dieses Kurses

- Professionalität im Ganzen reicht weit
  - z.B. zu Personalführung, Kollegialität, Umgang mit Auftraggebern, Organisationsentwicklg., etc.
- Wir beschränken uns hier weitgehend auf eine **persönliche Ebene**
- und dort vor allem auf Praktiken, die **eng am Code orientiert** sind.

- Die Praktiken bewegen sich auf zwei Ebenen:
  1. Wie sieht guter Code aus? (Softwareentwurf)
  2. Wie bekommt man ihn immer besser hin? (Methoden, Disziplin, persönliche Entwicklung)

aus Woche 1

Ende der Wiederholung aus Woche 1

Meta-Ebene: Was haben wir über die Prinzipien gelernt?

Freie Universität Berlin

1. Halte Maß!
   - Man kann alles übertreiben
   - Dann schadet es evtl. mehr als es nützt
   - Das passiert auch tatsächlich jeden Tag

2. Hinterfrage die Begriffe
   - z.B. "responsibility", "simple", "repeat", "unit test"
   - z.B. "interface", "component", wenn das keine vorgegebene technische Form hat

3. Mach Dir den Zweck klar
   - Orientierungspunkt für Entscheidungen beim Einsatz

Was haben wir nicht besprochen?

- Es gibt noch viel mehr nicht-technologische Themen, die für professionelle Softwareentwicklung regelmäßig relevant sind
- z.B. auf Entwurfsebene (CCD: "Prinzipien"):
  - Erzwingen von Verträgen
  - Ausnahmebehandlung
  - Logging
  - u.v.a.m.
- z.B. auf Verfahrensebene (CCD: "Praktiken"):
  - Aspekte von Teamarbeit
  - Aspekte von Persönlichkeitsentwicklung
- Deshalb sehr empfohlen: Die Praktiken der Kategorie "Bleib am Ball"
  - und: Ausprobieren!

Aber der Kurs ist doch noch gar nicht vorbei???

- Wie geht es denn jetzt weiter?

1. Testen
   - etwas genauer
2. Testen
   - noch etwas genauer
3. Zweiter Durchgang
   - Grade Rot bis Blau nochmal durchgehen
     - Tiefer rein in manche Aspekte/Beispiele
     - Zusammenhänge besser sehen

© miguelb, Flickr

Danke!

und jetzt: weiter mit dem Anfang der Einheit "Testen"