Software-Qualität

Florian Adamsky, B. Sc.

florian.adamsky@iem.thm.de
http://florian.adamsky.it/

Softwareentwicklung im WS 2014/15

Outline

Inhaltsverzeichnis

Was ist Software-Qualität?

- DIN-ISO Norm 9126 definiert den Begriff *Software-Qualität* wie folgt:

  "Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen."

- Funktionalität: Genauigkeit, Richtigkeit, Sicherheit (security), ...
- Aber auch nicht-funktionale Aspekte wie z.B.:
  - Konformität
  - Verständlichkeit
  - Analysierbarkeit
  - Modifizierbarkeit
  - Anpassbarkeit
  - Prüfbarkeit

$\Rightarrow$ Software-Qualität verbessert die funktionalen Aspekte

Was ist Software-Qualität?

- DIN-ISO Norm 9126 definiert den Begriff *Software-Qualität* wie folgt:

  "Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen."

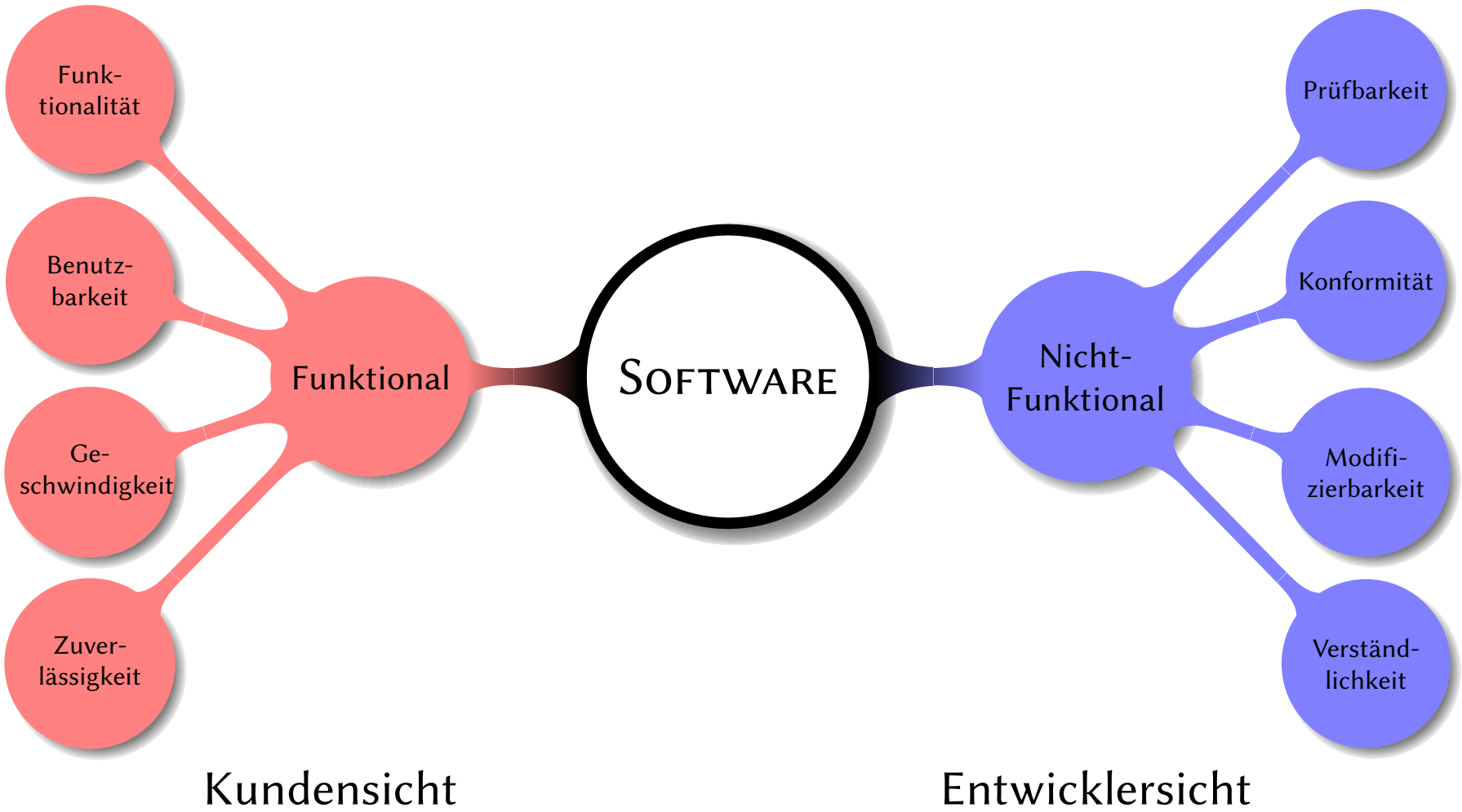
- Funktionalität: Genauigkeit, Richtigkeit, Sicherheit (security), ...
- Aber auch nicht-funktionale Aspekte wie z.B.:
  - Konformität
  - Verständlichkeit
  - Analysierbarkeit
  - Modifizierbarkeit
  - Anpassbarkeit
  - Prüfbarkeit

  } Software-Qualität

$\Rightarrow$ Software-Qualität verbessert die funktionalen Aspekte

Was ist Software-Qualität?

Qualitätsmerkmale



⇒ Software-Qualität verbessert die funktionalen Aspekte

Was ist das Problem mit der (Un)-Qualität von Software?

- Wir schreiten in das ubiquitäre Computerzeitalter
- Abhängigkeit von kritischer Software nimmt weiter zu
  - moderne Autos sind bepackt mit 30 Computern und über 100 Millionen Zeilen Code[1]
- Softwarefehler verursachen erhebliche Kosten
  - IX-Studie 2006:
  - 35.9 % des IT-Budgets geht ins Bug-Fixing
- Komplexität von Software-Systemen steigt exponentiell an

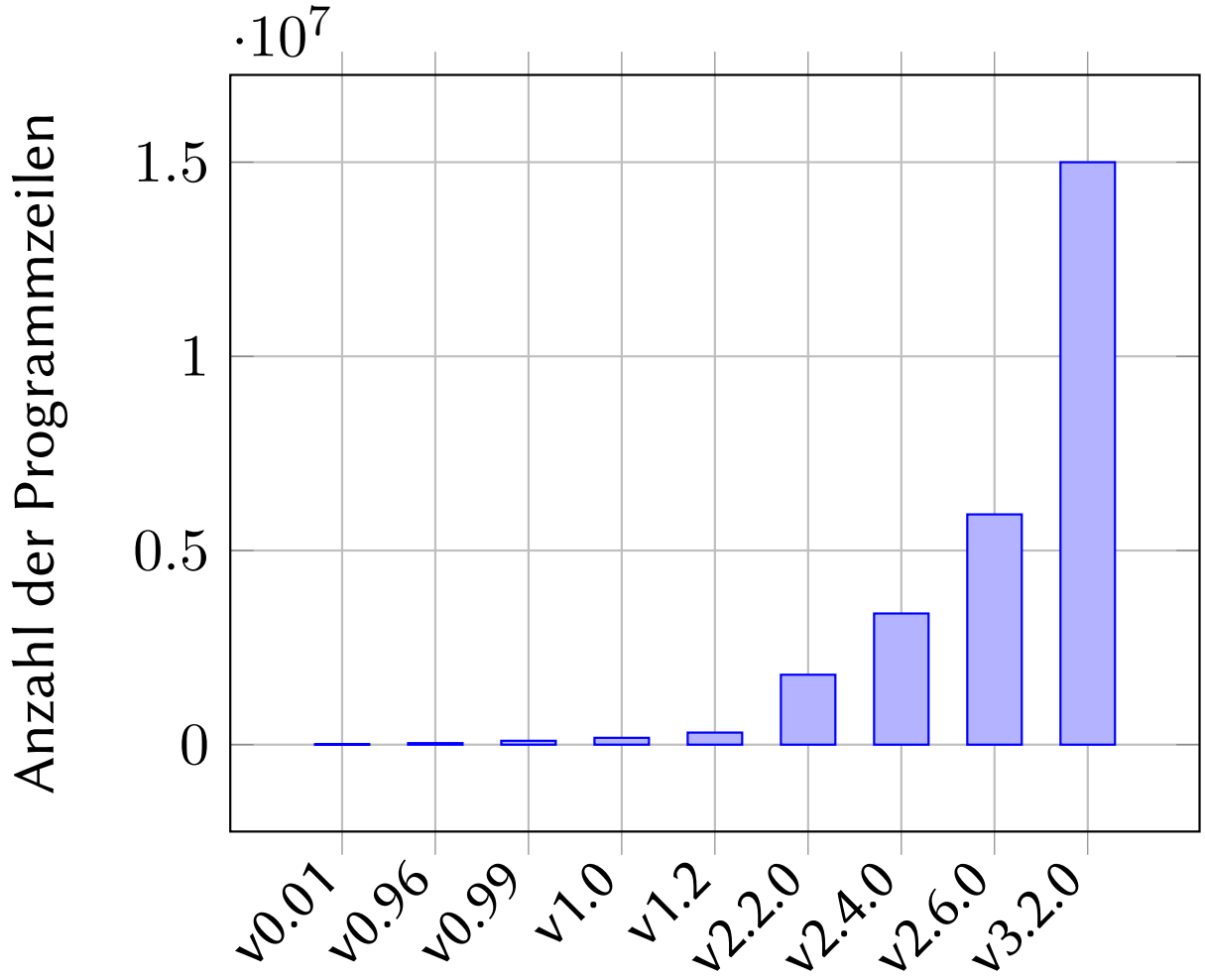
[1] https://www.nytimes.com/2010/02/05/technology/05electronics.html

Was ist das Problem mit der (Un)-Qualität von Software?

- Wi
- Ab
- So
- Ko

lionen

Entwicklung des Linux Kernel



[1] https://www.nytimes.com/2010/02/05/technology/05electronics.html

You're not so smart, then you think you are!

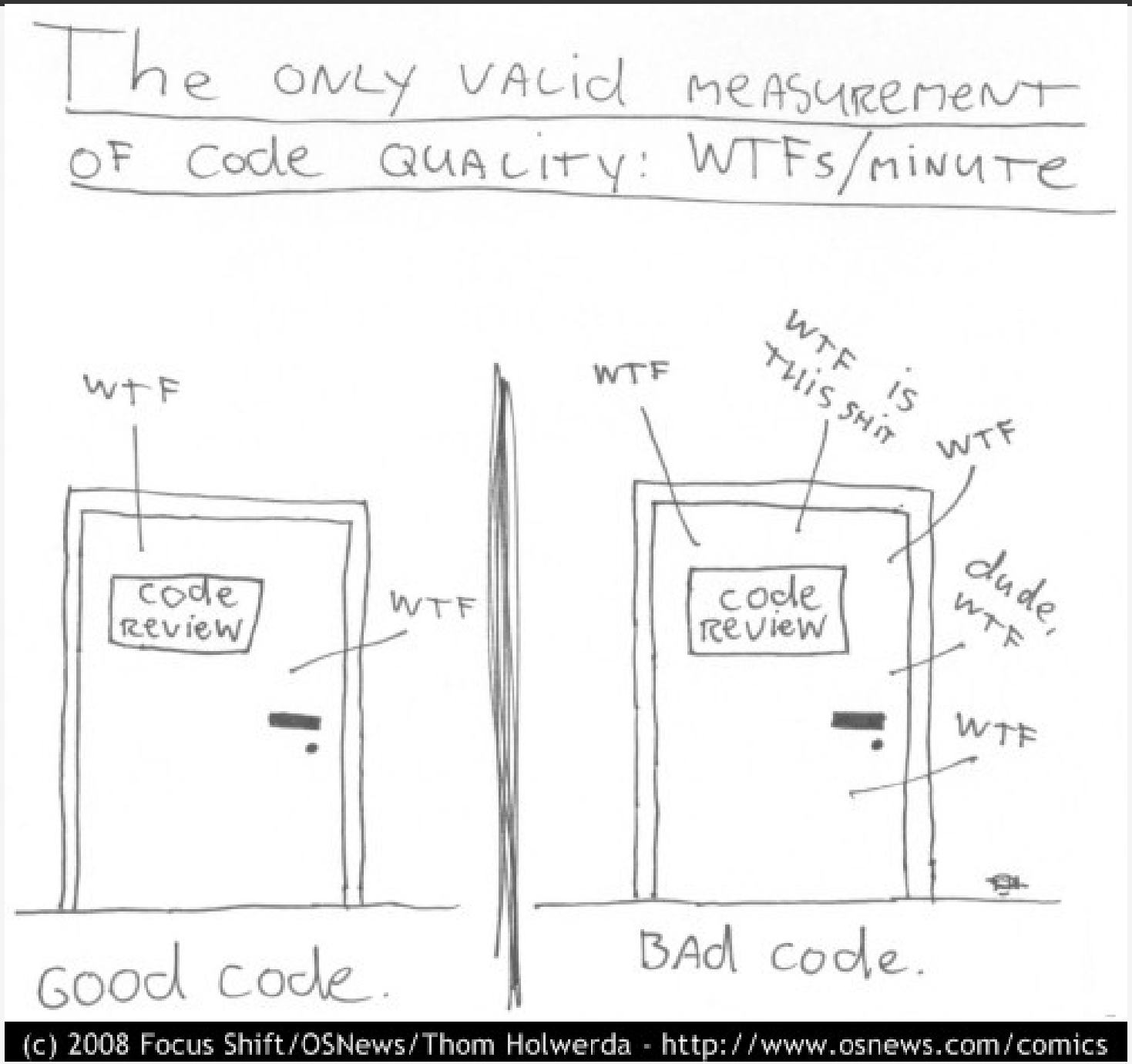
- Kurzzeitgedächtnis ist unser Arbeitsgedächtnis
  - Spanne ist begrenzt auf: 7 ± 2 Chunks
- Aufmerksamkeitsspanne ist kurz
- Unsere Pattern-Matching Fähigkeiten sind ausgeprägt

Ergo

- der Leser des Codes sollte nicht mehr als 5 Fakten auf einmal im Gedächtnis halten müssen um bestimmte Aspekte zu verstehen
- Code sollte in kleine verstehbare Pakete aufgeteilt werden
  - ähnlich werden lange Texte in Sektionen und Untersektionen aufgeteilt

Ursachen für schlechte Software-Qualität

- keine Zeit
- fehlendes Know-How
- unklare Projektzielsetzung
- falsche oder verfrühte Optimierung am Code
- Programmierer die versuchen besonders clever zu sein
- falsche Teambesetzung
- mangelhafte Projektplanung
- fehlende Qualitätssicherung


The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF IS THIS SHIT
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

Inhaltsverzeichnis

Mariner 1

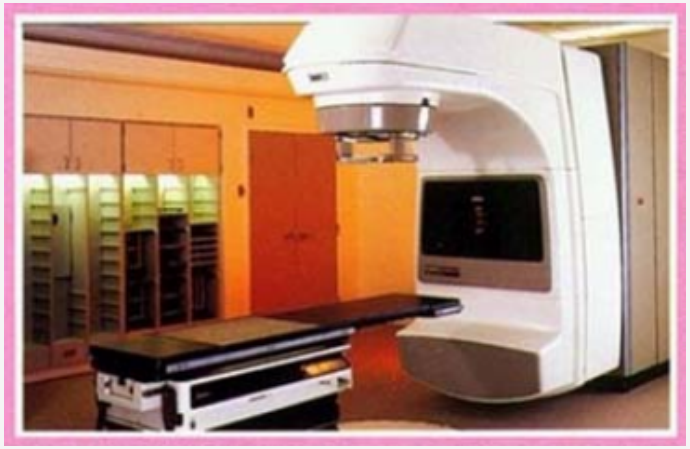
22. Juli 1962

- die NASA sprengte die Raumsonde nach dem Start, weil ein Bug dazu führte dass die Rakete vom Kurs abkam
- eine Formel auf Papier wurde falsch in Code transkripiert (Bindestrich fehlte)
- führte zu einer falschen Berechnung der Flugbahn
- Kosten: 18.2 Millionen Dollar

Therac-25

(1982–1985)

- Linearbeschleuniger zur Anwendung in der Strahlentherapie
- ein Softwarefehler tötete 3 Menschen und verletzte 3 weitere schwer
- bei dem Fehler handelte es sich um *race condition*



Quelle: `http://courses.cs.vt.edu/cs3604/lib/Therac_25/Therac_1`

Morris-Wurm

(1988)

- erster Internet Wurm
- infizierte 2000–6000 Computer in weniger als einem Tag
- keine böse Absicht, sondern Vermessung des Internets
- zum Eindringen nutze der Wurm eine Buffer-Overflow Schwachstelle im `finderd`

Beispielhafte Schwachstelle in fingerd

```
#define BUFFSIZE 8
char buff[BUFFSIZE];
gets(buff);
// ...
```

Inhaltsverzeichnis

Don’t repeat yourself (DRY)

Wiederhole dich nicht!

DRY-Prinzip

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system"

- eines der wichtigsten Fundamente in der Software-Entwicklung
- hat das Ziel Redundanz zu vermeiden
- hilft Systeme einfacher zu warten und Fehler zu finden
- Wie realisiert man das?
  - Auslagern in Funktionen, Methoden oder Klassen

```
if(sudokuBack[i][y]!=zahl) {
  if(sudokuBack[i][y+1]!=zahl) {
   if(sudokuBack[i][y+2]!=zahl) {
    if(sudokuBack[i][y+3]!=zahl) {
     if(sudokuBack[i][y+4]!=zahl) {
      if(sudokuBack[i][y+5]!=zahl) {
       if(sudokuBack[i][y+6]!=zahl) {
        if(sudokuBack[i][y+7]!=zahl) {
         if(sudokuBack[i][y+8]!=zahl) {
           return 0;
```

[1] Quelle: Sudokusolver – Medieninformatiker – 1. Semester

```
if(sudokuBack[i][y]!=zahl) {
  if(sudokuBack[i][y+1]!=zahl) {
   if(sudokuBack[i][y+2]!=zahl) {
    if(sudokuBack[i][y+3]!=zahl) {
     if(sudokuBack[i][y+4]!=zahl) {
      if(sudokuBack[i][y+5]!=zahl) {
       if(sudokuBack[i][y+6]!=zahl) {
        if(sudokuBack[i][y+7]!=zahl) {
         if(sudokuBack[i][y+8]!=zahl) {
           return 0;
```

Besser:

```
for (int y = 0; y < N; ++y) {
    if (sudokuField[i][y] == zahl)
        return true;
}
return false;
```

[1] Quelle: Sudokusolver – Medieninformatiker – 1. Semester

Keep it Simple and Stupid (KISS)

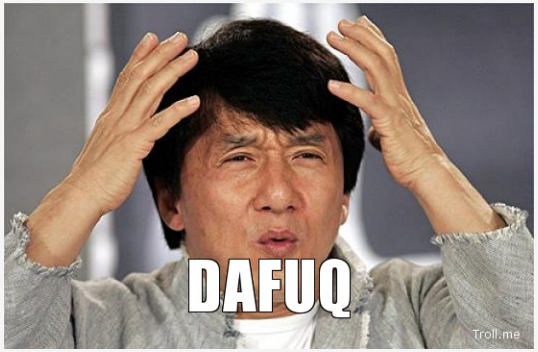
KISS-Prinzip

- Programmieren tendieren dazu Probleme zu verkomplizieren
- Einfacher Code ist
  - schneller zu verstehen und zu reparieren
  - einfacher zu warten und zu erweitern
  - wiederverwendbar und eleganter
- schreibt kurze Methoden/Funktionen
  - erhöht die wiederverwendbarkeit
  - erleichtert die testbarkeit

Negativbeispiel

Was tut das folgende Code-Fragment?

```
a = ((-foo)&b) | (~-foo)&c;
```



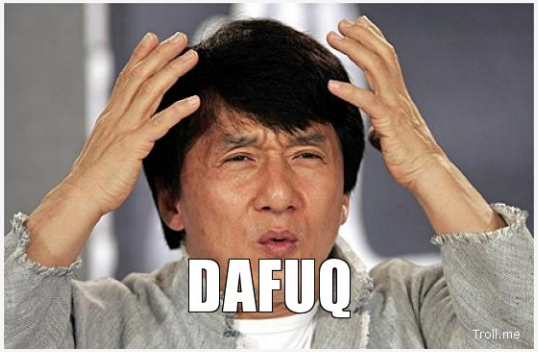
Negativbeispiel

Was tut das folgende Code-Fragment?

```
a = ((-foo)&b) | (~-foo)&c;
```

Das hier

```
if (foo)
    a = b;
else
    a = c;
```



Formatiert euren Code richtig

Einrückungen

- Tabulatoren sind böse $\rightarrow$ besser Leerzeichen
  - Tabulatoren sind auf andere Systen unterschiedlich definiert
- 4 Leerzeichen

Statements

- Jedes Statement gehört in eine extra Zeile
- Je mehr Statements in einer Zeile, desto schwerer zu lesen:

```
while (!fin.eof()) {
    string zeile; getline(fin, zeile);
    if (zeile == "test") { continue; } cout << zeile << endl;
}
```

Statements

- Jedes Statement gehört in eine extra Zeile
- Je mehr Statements in einer Zeile, desto schwerer zu lesen:

```
while (!fin.eof()) {
    string zeile; getline(fin, zeile);
    if (zeile == "test") { continue; } cout << zeile << endl;
}
```

- Besser

```
while (!fin.eof()) {
    string zeile;
    getline(fin, zeile);
    if (zeile == "test") {
        continue;
    }
    cout << zeile << endl;
}
```

Chunking

- Gruppierung von zusammengehörenden Instruktionen

```
while (!fin.eof()) {
    string zeile;
    getline(fin, zeile);
    if (zeile == "test")
        continue;
    cout << zeile << endl;
}
```

Chunking

- Gruppierung von zusammengehörenden Instruktionen

```
while (!fin.eof()) {
    string zeile;
    getline(fin, zeile);
    if (zeile == "test")
        continue;
    cout << zeile << endl;
}
```

```
while (!fin.eof()) {
    // Zeile einlesen
    string zeile;
    getline(fin, zeile);

    // Zeile überspringen bei Test
    if (zeile == "test")
        continue;

    // Zeile ausgeben
    cout << zeile << endl;
}
```

Klammern

K&R Style

```
for (int i = 0; i < N; ++i) {
    for (int y = 0; y < N; ++y) {
        cout << array[i][y];
    }
}
```

BSD Style

```
for (int i = 0; i < N; ++i)
{
    for (int y = 0; y < N; ++y)
    {
        cout << array[i][y];
    }
}
```

Namen sollten selbsterklärend sein!

Namesgebung

- Variablen-, Klassen- und Methodennamen sollten selbsterklärend sein
- aber auch schnell zu schreiben

Tipps

- denkt nach bevor Ihr einer Variable einen Namen gebt
- benutzt keine negative Logik für Namen

```
if (not debug_not_enabled) { ... }
```

besser

```
if (debug_is_enabled) { ... }
```

Namesgebung 2

- Ungarische Notation: `{Präfix}/{Datentyp} {Bezeichner}`
  - z.B.: `strName`, `pFooBar`, `kPi`
  - schwerer zu lesen `ThisIsHarderToRead`
  - Änderung des Datentypen bedeutete Variablenumbenennung
- Mit Unterstrich
  - z.B.: `max_velocity`, `err_article_soldout`, `user_id`
- Konstaten in Großbuchstaben

```
#define PI 3.1415926535
const double PI = 3.1415926535;
```

- bleibt konsistent

Negativbeispiel

```
o = 0; r = 0; p = 0;
q = 0; v = 0; u = 0;
for (r = 0; r < 3; r++) {
  v = 0;
    for (u = 0; u < 3; u++) {
      for (k3 = p + v + t;
           k3 <= p + v + t + 2; k3++) {
           o = 0;
           for (q = 0; q < 3; q++) {
             for (j3 = o + p + t;
                  j3 <= o + p + t + 2;
                  j3++) {
```

[1]Quelle: Sudokusolver – IKT – 5. Semester

Behebe das Problem, nicht die Symptome

Negativbeispiel

Example (Quelltext)

```
for (int j = 0; j <= timestamps.Length; j++) {
    try {
        if (timestamps[j] > timestamps[j + 1]) {
            correctOrder = false;
        }
    }
    catch {
    }
}
```

Negativbeispiel

Example (Quelltext)

Index Out of Range Exception

```
for (int j = 0; j <= timestamps.Length; j++) {
    try {
        if (timestamps[j] > timestamps[j + 1]) {
            correctOrder = false;
        }
    }
    catch {
    }
}
```

Kommentiere was der Code tut, nicht den Code

Kommentare

- Kommentare sind grundsätzlich sehr nützlich
- Kommentare die nicht so nützlich sind:

Kommentare

- Kommentare sind grundsätzlich sehr nützlich
- Kommentare die nicht so nützlich sind:

```
$i = 0; // set i to 0
```

Kommentare

- Kommentare sind grundsätzlich sehr nützlich
- Kommentare die nicht so nützlich sind:

```
$i = 0; // set i to 0

$i++; // use sneaky trick to add 1 to i
```

Kommentare

- Kommentare sind grundsätzlich sehr nützlich
- Kommentare die nicht so nützlich sind:

```
$i = 0; // set i to 0

$i++; // use sneaky trick to add 1 to i

if ($i == $j) { // I made sure to use == rather
                // than = here to avoid a bug
```

Inhaltsverzeichnis

Maßnahmen zur Verbesserung der Code-Qualität

- Code-Review im Team
- Programmierrichtlinien
- Unit-Tests
- Statische Code-Analyse
- Verwendung von Versionscontrolsystemen (VCS)
- Über den Tellerrand gucken
  - funktionales Paradigma (Lisp, Closure, Haskell, Erlang, F#)
- Konstantes Lernen

“Programming isn’t a good job for folks who aren’t interested in constant learning.” – Scott Hanselman

Danke

Fragen?

florian.adamsky@iem.thm.de
http://florian.adamsky.it/