Code Smells

Carsten Gips (FH Bielefeld)

Code Smells: Ist das Code oder kann das weg?

```
class checker {
    static public void CheckANDDO(DATA1 inp, int c, FH.Studi
    CustD, int x, int y, int in, int out,int c1, int c2, int c3 = 4)
{
    public int i; // neues i
for(i=0;i<10;i++) // fuer alle i
{
        inp.kurs[0] = 10; inp.kurs[i] = CustD.cred[i]/c;
}
      SetDataToPlan(  CustD  );
    public double myI = in*2.5; // myI=in*2.5
    if (c1)
        out = myI; //OK
    else if(  c3 == 4  )
    {
        myI = c2 * myI;
    if (c3 != 4 || true ) { // unbedingt beachten!
        //System.out.println("x:"+(x++));
        System.out.println("x:"+(x++)); // x++
        System.out.println("out: "+out);
    } }}   }
```

Was ist guter (“sauberer”) Code (“Clean Code”)?

- Gut (“angenehm”) lesbar
- Schnell verständlich: Geeignete Abstraktionen
- Konzentriert sich auf **eine** Aufgabe
- So einfach und direkt wie möglich
- Ist gut getestet

$=>$ Jemand kümmert sich um den Code; solides Handwerk

Warum ist guter (“sauberer”) Code so wichtig?

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

Quelle: (Fowler 2011)

Warum ist guter (“sauberer”) Code so wichtig?

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

Quelle: (Fowler 2011)

Stinkender Code führt zu möglichen (späteren) Problemen.

“Broken Windows” Phänomen

Code Smells: Nichtbeachtung von Coding Conventions

- Richtlinien für einheitliches Aussehen => Andere Programmierer sollen Code schnell lesen können
  - Namen, Schreibweisen
  - Kommentare (Ort, Form, Inhalt)
  - Einrückungen und Spaces vs. Tabs
  - Zeilenlängen, Leerzeilen
  - Klammern
- Beispiele: Sun Code Conventions, Google Java Style
- *Hinweis*: Betrifft vor allem die (äußere) Form!

Hinweis: Genauere Betrachtung in "Coding Rules"

Code Smells: Schlechte Kommentare I

- Ratlose Kommentare

```
/* k.A. was das bedeutet, aber wenn man es raus nimmt, geht's nicht mehr */
/* TODO: was passiert hier, und warum? */
```

- Redundante Kommentare: Erklären Sie, was der Code **inhaltlich** tun sollte (und warum)!

```
public int i; // neues i
for(i=0;i<10;i++)
// fuer alle i
```

Code Smells: Schlechte Kommentare II

- Veraltete Kommentare
- Auskommentierter Code
- Kommentare erscheinen zwingend nötig
- Unangemessene Information, z.B. Änderungshistorien

Code Smells: Schlechte Namen und fehlende Kapselung

```java
public class Studi extends Person {
    public String n;
    public int c;

    public void prtIf() { ... }
}
```

- Programmierprinzip "**Prinzip der minimalen Verwunderung**"
- Programmierprinzip "**Kapselung/Information Hiding**"

Code Smells: Duplizierter Code

```
public class Studi {
    public String getName() { return name; }
    public String getAddress() {
        return strasse+", "+plz+" "+stadt;
    }

    public String getStudiAsString() {
        return name+" ("+strasse+", "+plz+" "+stadt+")";
    }
}
```

- Programmierprinzip "**DRY**" => "Don't repeat yourself!"

Code Smells: Langer Code

- Lange Klassen
  - Faustregel: 5 Bildschirmseiten sind viel
- Lange Methoden
  - Faustregel: 1 Bildschirmseite
  - (Martin 2009): deutlich weniger als 20 Zeilen
- Lange Parameterlisten
  - Faustregel: max. 3 . . . 5 Parameter
  - (Martin 2009): 0 Parameter ideal, ab 3 Parameter gute Begründung nötig
- Tief verschachtelte `if/then`-Bedingungen
  - Faustregel: 2 . . . 3 Einrückungsebenen sind viel

- Programmierprinzip "**Single Responsibility**"

Code Smells: Feature Neid

```
public class CreditsCalculator {
    public ECTS calculateEcts(Student s) {
        int semester = s.getSemester();
        int workload = s.getCurrentWorkload();
        int nrModuls = s.getNumberOfModuls();
        int total = Math.min(30, workload);
        int extra = Math.max(0, total - 30);
        if (semester < 5) {
             extra = extra * nrModuls;
        }
        return new ECTS(total + extra);
    }
}
```

Wrap-Up

- Code entsteht nicht zum Selbstzweck => Lesbarkeit ist wichtig
- Code Smells: Code führt zu möglichen (späteren) Problemen
  - Richtiges Kommentieren und Dokumentieren
  - Einhalten von Coding Conventions
  - Einhalten von Prinzipien des objektorientierten Programmierens

LICENSE

Exceptions

- Citation "*Any fool can write code . . .*": (Fowler 2011)