PRINZIPIEN

Maurice Müller

2017-07-22

DRY

DON’T REPEAT YOURSELF

Definition

- Code-Duplikation vermeiden
- kopierter Code sind kopierte Fehler
  - Refactoring wird erschwert
- kopierter Code deutet auf falsche oder fehlende Abstraktion hin
- gilt auch für Logik → unterschiedlicher Code löst das gleiche Problem
  - falls unterschiedliche Lösungen Sinn machen → Strategy Pattern

KISS

KEEP IT SIMPLE, STUPID!

oder: keep it short and simple

Definition

- *einfach* ist oft besser als *kompliziert*
- Code muss verständlich sein und die eigene "Genialität" ausdrücken

Ähnlich: Einfachheit vor Allgemeinverwendbarkeit

- allgemeinverwendbare Lösungen tendieren zu erhöhter Komplexität
- trotzdem sinnvolle Abstraktionen nutzen!

Beispiel: KISS

```
public class Fingers {

    public enum Finger {THUMB, INDEX, MIDDLE, RING, LITTLE}

    public Finger fromNumber(int number) {
        switch(number) {
            case 0: return Finger.THUMB;
            case 1: return Finger.INDEX;
            case 2: return Finger.MIDDLE;
            case 3: return Finger.RING;
            case 4: return Finger.LITTLE;
            default:
                throw new RuntimeException("Invalid number: " + number);
        }
    }

    private Finger[] fingers = {Finger.THUMB, Finger.INDEX, Finger.MIDDLE,

    public Finger fromNumberGenius(int number) {
        if(number < 0 || number > 4) {
            throw new RuntimeException("Invalid number: " + number);
        }
        return fingers[number - 1];
    }
}
```

YAGNI

You ain’t gonna need it!

Du wirst es nicht brauchen!

Definition

- Funktionalität erst implementieren, wenn man sie braucht
  - strikt: sogar dann, wenn es absehbar ist
- nicht implementierte Funktionalität kostet nichts → keine Tests, keine Bugs, keine Verwirrung

Konvention vor Konfiguration

auch: Convention over Configuration, Configuration By Exception

Definition

- Frameworks / Module mit sinnvollen Voreinstellungen / Konventionen ausliefern
- häufig wird nur ein Teil gebraucht → der Rest soll 'einfach funktionieren' + (-) Mehraufwand um gute Defaults zu überlegen
- Beispiel: Spring

Beispiel

Wie kann diese Methode sinnvoll vorkonfiguriert werden?

```
public List<User> fetchUsers(Filter filter) {
    Query<User> query = new Query<>();
    query.matches(filter.getLastName(), User::getLastName);
    // ...
    return query.execute();
}
```

Lösung: Methode überladen

```
public List<User> fetchUsers() {
    return fetchUsers(Filter.All);
}
```

Prinzip der minimalen Verwunderung

englisch: Principle of Least Astonishment

- das Offensichtliche tun
  - d.h., das tun, was man erwarten würde
- z.B. getDate() liefert ein Datum zurück und kein Alter

Prinzip solider Annahmen

- nicht auf Vermutungen aufbauen, sondern auf Wissen
- wenn Vermutungen dann gut dokumentieren

- ähnlich: Prinzip der stabilen Abhängigkeit
  - Abhängigkeiten möglichst von stabilen Modulen (und von instabilen / sich häufig ändernden Modulen vermeiden)