

Testbarkeit und Testdriven Development (TDD)

Carsten Gips (HSBI)

Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

Design für Testbarkeit

```
public void ausgabe(...) {  
    ... // mache komplexe Berechnungen  
    ... // berechne nette Formatierungen  
    System.out.println(...);  
}
```

Design für Testbarkeit

```
public void ausgabe(...) {  
    ... // mache komplexe Berechnungen  
    ... // berechne nette Formatierungen  
    System.out.println(...);  
}
```

```
private A berechneA(...) {  
    return ...; // mache komplexe Berechnungen  
}  
  
private String formatiereA(A a) {  
    return ...; // berechne nette Formatierungen  
}  
  
public void ausgabe(...) {  
    System.out.println(formatiereA(berechneA(...)));  
}
```

Anzeichen für schlecht testbares Design

- Keine Rückgabewerte
- Direkte Ausgaben
- Mehr als ein Zweck:
 - Berechnung plus Ausgabe
 - Mehrere unterschiedliche Berechnungen
 - ...
- Seiteneffekte, z.B. Berechnung und direkte Veränderung von Zuständen **anderer** Objekte

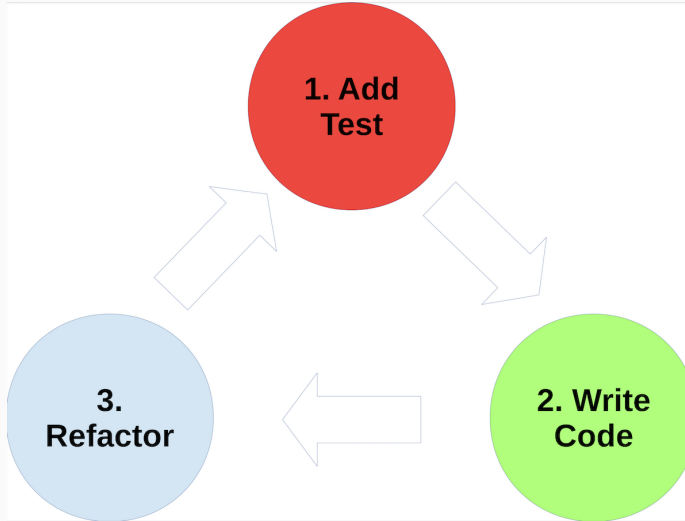
Private Methoden sind nicht (direkt) testbar

1. Gar nicht testen????
2. Sichtbarkeit auf `protected` setzen; Tests im selben Package unterbringen
3. JUnit 4/5: Annotationen => Testmethoden mit Produktionscode mischen
4. Reflection nutzen (vgl. spätere VL)

Was ist ein guter Test?

- Läuft schnell. Läuft schnell. Läuft schnell!!!
- Testet genau einen Aspekt, wenige Assertions
- Separiert Abhängigkeiten (Datenbanken, Netzwerk, ...)
- Absicht ist klar zu verstehen

TDD oder der “Red-Green-Refactor”-Zyklus



Anmerkung: Jeder Zyklus sollte sehr kurz sein!

- Testbarkeit muss mitgedacht werden: SW-Design
- Private Methoden auch testen (“Sicherheitsnetz”)
- Auch Tests brauchen SW-Design (Kapselung, Single Responsibility, ...)
- *Test first*: Test-Driven Development (*TDD*)

LICENSE



Unless otherwise noted, this work is licensed under CC BY-SA 4.0.