LU09b - Clean Code

Auf einen Blick
Code wird viel öfter gelesen als geschrieben. **Clean Code** ist Code, den andere schnell verstehen und gefahrlos ändern können. Einige wenige Prinzipien helfen dabei: sprechende Namen, kleine Funktionen mit genau einer Aufgabe (SRP), keine Wiederholungen (DRY), keine unnötige Komplexität (KISS, YAGNI). Verstösse dagegen nennt man **Code Smells**. Sie sind kein Absturz, aber ein Hinweis, dass der Code verbessert werden sollte.

Lernziele

Sie können

- die Prinzipien SRP, DRY, KISS und YAGNI erklären,
- Code Smells in einem Programm erkennen und benennen,
- an einem Beispiel zeigen, wie Code verbessert wird.

Sprechende Namen

Ein Name sagt, **was** etwas ist oder tut. Abkürzungen und Einbuchstaben-Namen zwingen die Leserin, den Code zu entschlüsseln.

```
# schlecht
def calc(l):
    s = 0
    for x in l:
        s += x
    return s / len(l)

# besser
def average(grades):
    return sum(grades) / len(grades)
```

Faustregeln: Funktionen mit Verb (`load_students`, `is_promoted`), Variablen mit Nomen (`grades`, `total`), Wahrheitswerte als Frage (`is_valid`, `has_errors`).

Single Responsibility Principle (SRP)

Eine Funktion oder Klasse hat **genau eine Aufgabe** und damit nur einen Grund, sich zu ändern.

```
# schlecht: berechnet, formatiert und gibt aus
def report(grades):
```

```
    avg = sum(grades) / len(grades)
    text = f'Schnitt: {avg:.1f}'
    print(text)

# besser: jede Funktion erledigt eine Aufgabe
def average(grades):
    return sum(grades) / len(grades)

def format_report(avg):
    return f'Schnitt: {avg:.1f}'

print(format_report(average(grades)))
```

Funktionen mit einer Aufgabe lassen sich einzeln testen. Eine Funktion, die rechnet **und** ausgibt, kann man nur über die Bildschirmausgabe prüfen.

DRY – Don't Repeat Yourself

Jedes Wissen steht nur **an einer Stelle** im Code. Kopierter Code muss bei jeder Änderung an allen Stellen angepasst werden, und irgendwann wird eine Stelle vergessen.

```
# schlecht: dreimal derselbe Ablauf
math_avg = round(sum(math) / len(math) * 2) / 2
german_avg = round(sum(german) / len(german) * 2) / 2
english_avg = round(sum(english) / len(english) * 2) / 2

# besser: eine Funktion, mehrfach verwendet
def subject_grade(grades):
    return round(sum(grades) / len(grades) * 2) / 2

math_avg = subject_grade(math)
german_avg = subject_grade(german)
english_avg = subject_grade(english)
```

Dazu gehören auch **Magic Numbers**: Steht die Zahl 4 als Genügend-Grenze an fünf Stellen im Code, gehört sie in eine Konstante `PASS_MARK = 4.0`.

KISS und YAGNI

- **KISS** – *Keep it simple, stupid*: Die einfachste Lösung, die funktioniert, ist meistens die beste. Verschachtelte Bedingungen und clevere Einzeiler sind schwer zu lesen.
- **YAGNI** – *You ain't gonna need it*: Programmieren Sie nur, was heute gebraucht wird. Vorbereitete Optionen «für später» werden oft nie gebraucht, müssen aber trotzdem getestet und gepflegt werden.

```
# KISS verletzt
def is_promoted(avg, fails):
```

```
    return True if (avg >= 4 and not fails > 1) else False

# einfach
def is_promoted(avg, fails):
    return avg >= 4 and fails <= 1
```

Typische Code Smells

Code Smell	Woran erkenne ich ihn?	Prinzip
Unklare Namen	`a`, `tmp`, `data2`, `calc`	Sprechende Namen
Lange Funktion	Funktion passt nicht auf den Bildschirm, Kommentare trennen Abschnitte	SRP
Doppelter Code	gleiche oder fast gleiche Zeilen an mehreren Stellen	DRY
Magic Number	Zahlen ohne Namen wie `4`, `0.077`, `20`	DRY
Lange Parameterliste	mehr als 3–4 Parameter	SRP
Toter Code	auskommentierter Code, unbenutzte Variablen und Imports	KISS
Erklärender Kommentar	Kommentar erklärt, **was** der Code tut, statt warum	Sprechende Namen

Refactoring

Code verbessern, ohne sein Verhalten zu ändern, heisst **Refactoring**. Damit sicher nichts kaputtgeht, braucht es Tests:

1. Tests schreiben bzw. ausführen: alle grün.
2. Einen kleinen Schritt verbessern (umbenennen, Funktion herauslösen, Konstante einführen).
3. Tests ausführen: immer noch grün? Dann weiter mit dem nächsten Schritt.

Refactoring ist auch der dritte Schritt im Zyklus von Test Driven Development (siehe LU11a).

Kontrollfragen

1. Was bedeuten SRP und DRY? Geben Sie je ein Beispiel für einen Verstoss.
2. Warum ist eine Funktion, die rechnet und gleichzeitig mit `print` ausgibt, schwer zu testen?
3. Was ist eine Magic Number, und wie beheben Sie sie?
4. Weshalb braucht man Tests, bevor man ein Refactoring durchführt?

Weiterführende Links

- Robert C. Martin: *Clean Code*, Prentice Hall 2008
- refactoring.guru: Code Smells

M450-LU09, M450-D1B

Marcel Suter

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m450/learningunits/lu09/cleancode

Last update: **2026/11/23 09:17**