LU07b - Code Smells erkennen

Ein **Code Smell** ist kein Fehler. Das Programm läuft, die Tests sind grün - und trotzdem stimmt etwas nicht: Der Code ist schwer zu lesen, schwer zu ändern oder schwer zu testen. Der Smell ist der Hinweis, an welcher Stelle sich ein Refactoring lohnt.

Warum «Geruch»? Weil er ein Verdacht ist, kein Beweis. Eine Funktion mit 40 Zeilen kann völlig in Ordnung sein. Ein Smell sagt nur: Schauen Sie hier genauer hin.

Die zehn Smells, die Sie erkennen müssen

1. Lange Funktion

Die Funktion tut mehr als eine Sache. Erkennbar an Kommentaren, die Abschnitte markieren («# jetzt die Summe», «# jetzt die Ausgabe») - das sind die Schnittlinien.

```
def auswertung(datei):
    # einlesen
    ...
    # filtern
    ...
    # rechnen
    ...
    # formatieren und ausgeben
    ...
```

Technik: Extract Function

2. Duplizierter Code

Dieselbe Logik steht an mehreren Stellen. Beim nächsten Fix wird eine davon vergessen.

```
netto_a = round(preis_a * 1.081, 2)
netto_b = round(preis_b * 1.081, 2)
```

Technik: Extract Function, bei Varianten Parameterize Function

3. Magic Number

Eine Zahl oder ein String steht mitten im Code, und niemand weiss, was sie bedeutet.

```
if kunde["punkte"] > 500:
```

```
    rabatt = betrag * 0.15
```

Technik: Replace Magic Number with Named Constant

4. Lange Parameterliste

Fünf oder mehr Parameter, die inhaltlich zusammengehören. Beim Aufruf vertauscht man irgendwann zwei davon - und Python merkt es nicht.

```
def zeilenpreis(artikel, menge, einzelpreis, rabatt, waehrung, mwst):
    ...
```

Technik: Introduce Parameter Object (frozen Dataclass, siehe LU02e)

5. Tief verschachtelte Bedingungen

Der eigentliche Fall steht ganz innen, eingerückt hinter drei `if`. Die Sonderfälle verdecken die Regel.

```
if kunde is not None:
    if kunde["aktiv"]:
        if betrag >= 100:
            return betrag * 0.1
```

Technik: Guard Clauses (Replace Nested Conditional with Guard Clauses)

6. Kommentar als Krücke

Der Kommentar erklärt, *was* eine Zeile tut, weil die Zeile es selbst nicht sagt.

```
# prüft ob der Kunde volljährig und aktiv ist
if k["a"] >= 18 and k["s"] == 1:
    ...
```

Technik: Extract Variable oder Extract Function mit sprechendem Namen. Ein guter Kommentar erklärt das *Warum*, nicht das *Was*.

7. Globaler Zustand

Die Funktion liest oder schreibt eine Variable ausserhalb ihrer selbst. Damit hängt ihr Ergebnis von der Programmgeschichte ab - genau das, was LU02b als *unpure* beschreibt.

```
total = 0

def add_to_total(x):
    global total
```

```
    total += x
```

Technik: Zustand als Parameter hereingeben und als Rückgabewert herausgeben, alternativ ein Closure (LU05c)

8. Query und Modifier vermischt

Eine Funktion liefert einen Wert *und* verändert nebenbei etwas. Man kann sie nicht aufrufen, nur um nachzuschauen.

```
def entnehmen(bestand, artikel):
    bestand[artikel] -= 1      # verändert
    return bestand[artikel]    # und liefert
```

Technik: Separate Query from Modifier

9. Boolean-Flag-Parameter

Am Aufruf ist nicht erkennbar, was passiert: `exportieren(daten, True)` - was ist `True`?

Technik: Zwei Funktionen mit sprechenden Namen, oder Keyword-Argument erzwingen (`exportieren(daten, mit_header=True)`)

10. Schleife mit Akkumulator

Eine Schleife, die nur filtert, umformt oder aufsummiert - das ist der Smell mit dem stärksten Bezug zu diesem Modul.

```
resultat = []
for e in daten:
    if e["aktiv"]:
        resultat.append(e["name"].upper())
```

Technik: Replace Loop with Pipeline (LU07d)

Übersicht: Smell und passende Technik

Smell	Woran erkennbar	Technik
Lange Funktion	Abschnittskommentare, mehr als ein Zweck	Extract Function
Duplizierter Code	dieselbe Zeile mehrfach	Extract / Parameterize Function
Magic Number	nackte Zahl oder String im Code	Named Constant
Lange Parameterliste	5+ Parameter, die zusammengehören	Introduce Parameter Object
Verschachtelte Bedingungen	drei Einrückungsebenen und mehr	Guard Clauses

Smell	Woran erkennbar	Technik
Kommentar als Krücke	Kommentar erklärt das Was	Extract Variable / Function
Globaler Zustand	`global`, Modulvariable wird geschrieben	Parameter und Rückgabewert, Closure
Query und Modifier vermischt	Funktion liefert und verändert	Separate Query from Modifier
Boolean-Flag	`True` / `False` als Argument	zwei Funktionen oder Keyword-Argument
Schleife mit Akkumulator	`resultat = []` plus `append`	Replace Loop with Pipeline

Smell ist nicht gleich Schuld. Ein Smell rechtfertigt einen Blick, nicht automatisch eine Änderung. Wer jede Dreizeiler-Schleife in eine verschachtelte Comprehension presst, hat den Code nicht besser gemacht, sondern nur anders unleserlich. Die Frage lautet immer: Ist die Version danach für den nächsten Leser einfacher?

M323-LU07, M323-D1B