Refactoring

BBZW, Modul 426

Patrick Bucher

06.06.2024

Refactoring: Ziel

- Wir können bestehenden Code **verändern**.
- Dabei richten wir **kein Chaos** an.
- Sondern **verbessern** dessen Struktur, …
- … indem wir **systematisch** vorgehen.

Refactoring: Eine agile Praktik

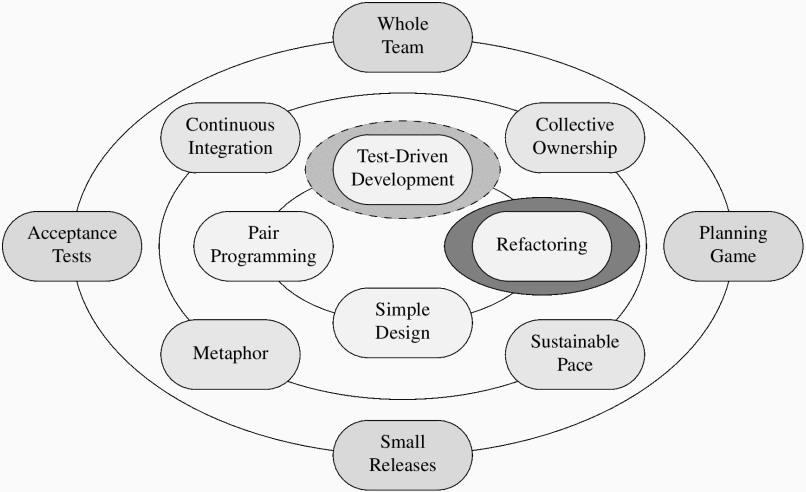


Abbildung 1: Refactoring ist eine *technische* Praktik im innersten Ring des *Circle of Life*

Refactoring: Die Definition

> ***Refactoring*** *(noun): a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.*

(Quelle: Martin Fowler & Kent Beck: *Refactoring*)

Refactoring, Bugfixing?

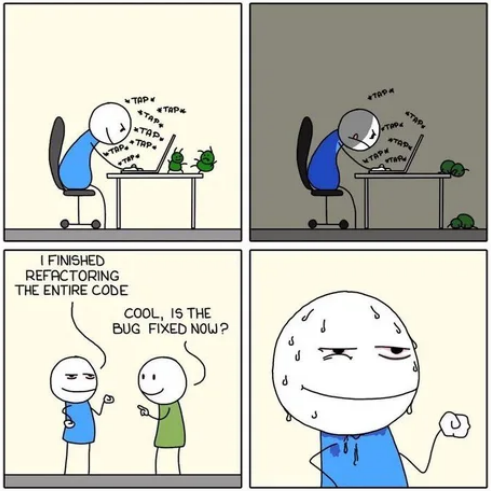


Abbildung 2: Refactoring ist *nicht* Bugfixing

Refactoring: Was bedeutet das?

- Refactoring ist die Anpassung der **inneren Struktur** der Software.
- Das von aussen wahrnehmbare **Verhalten** der Software wird *nicht* angepasst.
- Der Code wird besser **verständlich** und besser **erweiterbar**.

Frage: Wie definieren wir *"das von aussen wahrnehmbare Verhalten der Software"*?

Refactoring: Unit Tests als Voraussetzung!

Zur Erinnerung: Dank Unit Tests können wir Code **furchtlos** anpassen!

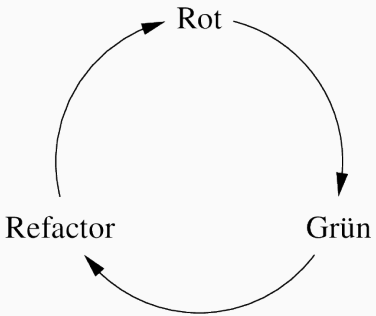


Abbildung 3: Refactoring und Test-Driven Development spielen eng zusammen

Vorgehen beim Refactoring

1. **Problem erkennen:** Was ist schlecht am Code?
   - siehe Liste von Code Smells
2. **Tests schreiben:** Wie soll sich der Code verhalten?
   - falls noch kein Test vorhanden ist
3. **Refactoring anwenden:** Das erkannte Problem lösen
   - siehe Liste von Refactorings

Problem erkennen: *Code Smells*

Ein *Code Smell* ist etwas am Code, das eigenartig aussieht und auf ein mögliches Problem hinweisen *kann* (*Heuristik*).

Wortherkunft: Wenn ein Säugling weint (Problem) und es schlecht riecht (“smell”), sollte man die Windel wechseln.

Beispiel: Berechnung von $8x^3 + 2y^4$

```
int Polynomial(int x, int y)
{
    int xToThePowerOfThree = 1;
    for (int i = 0; i < 3; i++) {
        xToThePowerOfThree *= x;
    }
    int yToThePowerOfFour = 1;
    for (int i = 0; i < 4; i++) {
        yToThePowerOfFour *= y;
    }
    return 8 * xToThePowerOfThree + 2 * yToThePowerOfFour;
}
```

Code Smell: Duplizierter Code

Es gibt zwei fast identische `for`-Schleifen zur Berechnung der Potenz:

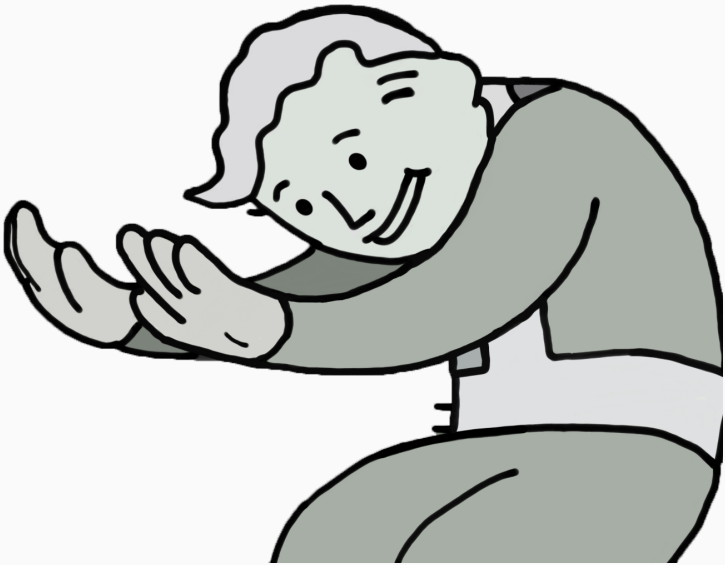
```
int xToThePowerOfThree = 1;
for (int i = 0; i < 3; i++) {
    xToThePowerOfThree *= x;
}
```

Und:

```
int yToThePowerOfFour = 1;
for (int i = 0; i < 4; i++) {
    yToThePowerOfFour *= y;
}
```

Refactoring, los!

HOLD UP!



Unit Test: Verhalten sicherstellen

```
void TestPolynomial()
{
    // Arrange
    int x = 2;
    int y = 3;
    int expected = 226; // 8x³ + 2y⁴

    // Act
    int actual = Polynomial(x, y);

    // Assert
    Assert.Equal(actual, expected);
}
```

Refactoring: Funktion extrahieren (I) – Muster erkennen

Alt (kompakt):

```
int xToThePowerOfThree = 1;
for (int i = 0; i < 3; i++) { xToThePowerOfThree *= x; }

int yToThePowerOfFour = 1;
for (int i = 0; i < 4; i++) { yToThePowerOfFour *= y; }
```

Unterschiede/Gemeinsamkeiten:

```
int [variable] = 1;
for (int i = 0; i < [n]; i++) { [variable] *= [x]; }
```

Refactoring: Funktion extrahieren (II) – Funktion schreiben

```
// calculates x^n
int Pow(int x, int n)
{
    int result = 1;
    for (int i = 0; i < n; i++) {
        result *= x;
    }
    return result;
}
```

Refactoring: Funktion extrahieren (III) – Funktion verwenden

```
int Polynomial(int x, int y)
{
    return 8 * Pow(x, 3) + 2 * Pow(y, 4);
}
```

Läuft der Test durch, sind wir fertig.

Auftrag 1

- **Arbeitsanweisungen**:
  1. Lesen Sie das README zum Thema Refactoring durch.
  2. Überfliegen Sie die verlinkten Ressourcen zu *Code Smells* und *Refactorings*.
- **Form**: Einzelarbeit
- **Ziele**:
  1. Sie können den *Nutzen* vom Refactoring nachvollziehen.
  2. Sie können das *Vorgehen* beim Refactoring nachvollziehen.
  3. Sie können häufige *Code Smells* im Programmcode wiedererkennen.
  4. Sie können gebräuchliche *Refactorings* nachvollziehen.
- **Zeit**: 10 min.

Auftrag 2

- **Arbeitsanweisungen**: Bearbeiten Sie die Aufgaben zum Thema Refactoring.
- **Form**: Einzel-, Partner- (Pair Programming) oder Gruppenarbeit (Mob Programming maximal zu dritt)
- **Ziele**:
  1. Sie können *Code Smells* identifizieren.
  2. Sie können passende *Refactorings* zu den gefundenen *Code Smells* finden und anwenden.
  3. Sie können sich dabei mithilfe von Unittests an das Vorgehen beim Refactoring halten.
- **Zeit**: 45 min. + Hausaufgabe