

# Vorlesung Software-Reengineering

Prof. Dr. Rainer Koschke

Arbeitsgruppe Softwaretechnik  
Fachbereich Mathematik und Informatik  
Universität Bremen

Wintersemester 2008/09

## Überblick I

① Refactoring

- ① Refactoring
  - Refactoring
  - Bad Smells
  - Refactorings von Fowler
  - Pull-Up Field
  - Extract Method
  - Bewertung Refactorings
  - Wiederholungsfragen

## Refactorings

**Refactorings** sind semantikerhaltende, restrukturierende Code-Transformationen für objekt-orientierte Programme (zur Verbesserung der Wartbarkeit)

Beschreibung nach Fowler (2000):

- Name
- Anwendbarkeit
- Motivation
- mechanische Schritte (die eigentliche Transformation), die von Hand ausgeführt werden
- Beispiel

Sehr viele dieser Refactorings sind genauso auf prozedurale Programme anwendbar.

Angestoßen von Änderungswunsch.

Prozess (inkrementell, iterativ):

- ① Identifikation eines „schlechten Geruchs“ (bad smell)
- ② Refactoring
- ③ Compile & Test
- ④ Eigentliche Änderung
- ⑤ Compile & Test

## Stink Parade of Bad Smells by Fowler (2000)

- duplizierter Code
- lange Methoden
- große Klassen
- lange Parameterlisten
- divergente Änderung
  - eine Klasse wird stets geändert in verschiedener Weise und für unterschiedliche Gründe
- Schrotflinten-Chirurgie (Shotgun Surgery)
  - kleine Änderungen überall
- Feature-Neid
  - sehr viele Attribute einer anderen Klasse werden für eine Berechnung benutzt

# Stink Parade of Bad Smells by Fowler (2000)

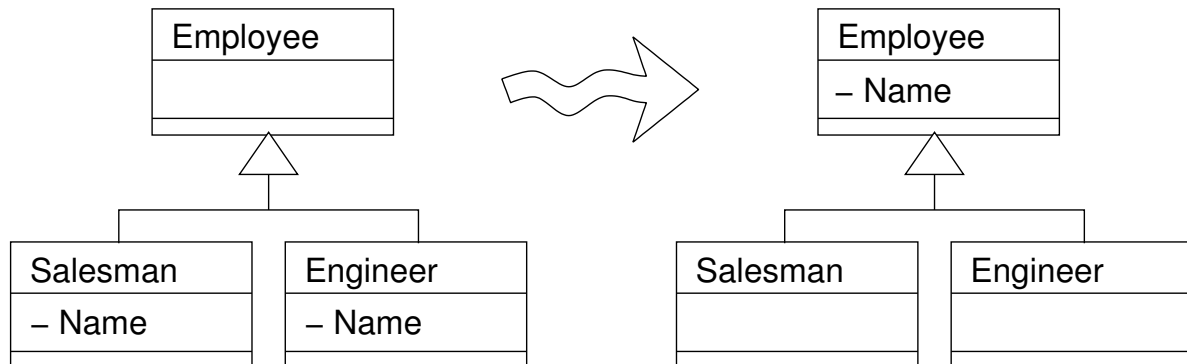
- Datenklumpen (Data Clumps)
  - eine Menge von Datenelementen, die häufig gemeinsam benutzt werden
  - z.B. Attribute einer Klasse, Parameter in Methodensignaturen
- Fixierung aufs Primitive (Primitive Obsession)
  - einfache Typen werden nicht als Klasse sondern als primitive Datentypen deklariert
- Switch-Anweisungen
  - händisches dynamisches Binden
- Parallele Vererbungshierarchien
- Faule Klassen
  - Klassen, die nichts Nützliches (mehr) tun
- ...

## 70 Refactorings von Fowler (2000)

- Methodenzusammensetzung
  - z.B. Extraktion von Methoden
- Eigenschaften zwischen Klassen bewegen
  - z.B. Verschiebung von Attributen oder Methoden
- Organisation von Daten
  - z.B. Verbergung von Attributen
- Vereinfachung bedingter Ausdrücke
  - z.B. Zerlegung komplexer Bedingungen
- Vereinfachung von Methodenaufrufen
  - z.B. Separierung von bloßem Zugriff von Manipulation
- Generalisierungen
  - z.B. Attribute oder Methoden in der Hierarchie auf- oder abwärts bewegen

## Beispiel: Pull-Up Field

Gleiches Attribut in Unterklassen wird nach Oberklasse verlegt.



## Motivation für Pull-Up Field nach Fowler (2000)

„If subclasses are developed independently, or combined through refactoring, you often find that they duplicate features. In particular, certain fields can be duplicates. Such fields sometimes have similar names but not always. The only way to determine what is going on is to look at the fields and see how they are used by other methods. If they are being used in a similar way, you can generalize them.“

„Doing this reduces duplication in two ways. It removes the duplicate data declaration and allows you to move from the subclasses to the superclass behavior that uses the field.“

- ① Inspect all uses of the candidate fields to ensure they are used in the same way.
- ② If the fields do not have the same name, rename the fields so that they have the name you want to use for the superclass field.
- ③ Compile and test.
- ④ Create a new field in the superclass.
- ⑤ If the fields are private, you will need to protect the superclass field so that the subclasses can refer to it.
- ⑥ Delete the subclass fields.
- ⑦ Compile and test.
- ⑧ Consider using *Self Encapsulate Field* on the new field.

## Extract Method

```
void printOwing() {
    Enumeration e = _order.elements();
    double outstanding = 0.0;

    // print banner
    System.out.println ("*****");
    System.out.println ("*_*_*_Customer_Owes_*");
    System.out.println ("*****");
    // calculate outstanding
    while (e.hasMoreElements()) {
        Order each = (Order) e.nextElement();
        outstanding += each.getAmount();
    }
    // print details
    System.out.println ("name_*" + _name);
    System.out.println ("amount_*" + outstanding);
}
```

## Extract Method: parameterlos

```
void printOwing() {
    Enumeration e = _order.elements();
    double outstanding = 0.0;

    printBanner();
    // calculate outstanding
    while (e.hasMoreElements()) {
        Order each = (Order) e.nextElement();
        outstanding += each.getAmount();
    }
    // print details
    System.out.println ("name_" + _name);
    System.out.println ("amount_" + outstanding);
}

void printBanner() {
    // print banner
    System.out.println ("*****");
    System.out.println ("*_Customer_Owes_*");
    System.out.println ("*****");
}
```

## Extract Method: nur Eingabeparameter

```
void printOwing() {
    Enumeration e = _order.elements();
    double outstanding = 0.0;

    printBanner();
    // calculate outstanding
    while (e.hasMoreElements()) {
        Order each = (Order) e.nextElement();
        outstanding += each.getAmount();
    }
    printDetails(outstanding);
}

void printDetails (double outstanding) {
    // print details
    System.out.println ("name_" + _name);
    System.out.println ("amount_" + outstanding);
}
```

```
void printOwing() {
    printBanner();
    double outstanding = getOutstanding();
    printDetails(outstanding);
}

double getOutstanding () {
    // calculate outstanding
    Enumeration e = _order.elements();
    double outstanding = 0.0;
    while (e.hasMoreElements()) {
        Order each = (Order) e.nextElement();
        outstanding += each.getAmount();
    }
    return outstanding;
}
```

## Bewertung Refactorings

- ① Die „Mechanics“ werden von Hand durchgeführt
  - es gibt Werkzeuge, die manche Refactorings automatisieren (Refactoring Browser für Smalltalk, Eclipse für Java).
- ② Beschreibung ist zu informell für eine automatisierte Transformation, aber zumindest eine gute Checkliste.
- ③ Die genauen Vorbedingungen sind nicht ausreichend angegeben.
- ④ Hinter „Compile & Test“ kann sich viel Arbeit verbergen.

- Welche Schritte sind beim Refactoring durchzuführen?
- Was ist ein Bad Smell? Beispiele?
- Wie lassen sich Bad Smells erkennen?

1 **Fowler 2000** FOWLER, Martin: Refactoring: Improving the Design of Existing Code. Addison-Wesley, 2000