

Vorlesung Software-Reengineering

Dipl. Inf. Gunther Vogel¹

¹ISTE

Fakultät für Informatik, Elektrotechnik und Informationstechnik
Universität Stuttgart
(Prof. Dr. Rainer Koschke, Universität Bremen)

Wintersemester 2006/2007

Überblick I

- 1 Einführung
- 2 Statische Programmanalyse
- 3 Program Slicing
- 4 Refactoring
- 5 Klonerkennung
- 6 Mustersuche
- 7 Codetransformation

Überblick II

- 8 Metriken
- 9 Strukturelle Architektursichten
- 10 Begriffsanalyse
- 11 Merkmalsuche
- 12 Analyse und Restrukturierung von Vererbungshierarchien
- 13 Software Visualisierung
- 14 Durchführung von Reengineering-Projekten

- 1 Einführung
 - Administrativa
 - Lernziele
 - Motivation
 - Wichtige Begriffe
 - Wartung
 - Reverse Engineering
 - Restrukturierung
 - Reengineering
 - Wrapping
 - Business Process Reengineering
 - Ziele und Aufgaben
 - Unterschiede zur Vorwärtsentwicklung

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)

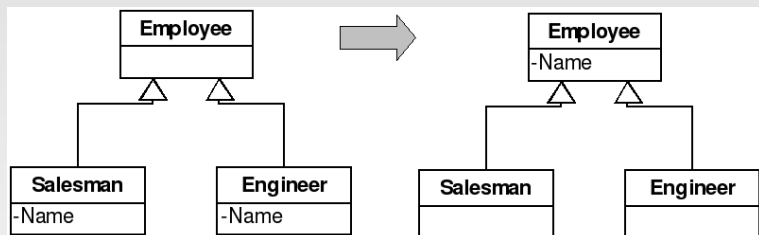
- Datenklumpen (Data Clumps)
 - eine Menge von Datenelementen, die häufig gemeinsam benutzt werden
 - z.B. Attribute einer Klasse, Parameter in Methodensignaturen
- Fixierung aufs Primitive (Primitive Obsession)
 - einfache Typen werden nicht als Klasse sondern als primitive Datentypen deklariert
- Switch-Anweisungen
 - 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.”

Mechanics für Pull-Up Fields

- ① Inspect all uses of the candidate fields to ensure they are used in the same way.

Mechanics für Pull-Up Fields

- 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.

Mechanics für Pull-Up Fields

- 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.

Mechanics für Pull-Up Fields

- 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.

Mechanics für Pull-Up Fields

- 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.

Mechanics für Pull-Up Fields

- 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.

Mechanics für Pull-Up Fields

- 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.

Mechanics für Pull-Up Fields

- 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 printOwning() {
    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 printOwning() {
    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 printOwning() {
    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);
}
```

Extract Method: mit Ausgabeparameter

```
void printOwning() {
    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;
}
```

- 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.

Wiederholungs- und Vertiefungsfragen I

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

- 1 Baumöl u. a. 1996** BAUMÖL, Ulrike ; BORCHERS, Jens ; EICKER, Stefan ; HILDEBRAND, Knut ; JUNG, Reinhard ; LEHNER, Franz: Einordnung und Terminologie des Software Reengineering. In: Informatik Spektrum 19 (1996), S. 191–195
- 2 Bellon 2002** BELLON, Stefan: Vergleich von Klonerkennungstechniken. Fakultät Informatik, Universität Stuttgart, Deutschland, Diplomarbeit, 2002
- 3 Boehm 1981** BOEHM, Barry: Software Engineering Economics. Englewood Cliffs, NJ : Prentice Hall, 1981
- 4 Bruntink und van Deursen 2004** BRUNTINK, M. ; DEURSEN, A. van: Predicting Class Testability Using Object-Oriented Metrics. In: Proceedings of the Fourth IEEE International Workshop on Source Code Analysis and Manipulation, IEEE Computer Society Press, September 2004

- 5 Chen und Rajlich 2000** CHEN, Kunrong ; RAJLICH, Václav: Case Study of Feature Location Using Dependence Graph. In: Proceedings of the 8th International Workshop on Program Comprehension. Limerick, Irland : IEEE Computer Society Press, Juni 2000, S. 241–249
- 6 Chikofsky und Cross II. 1990** CHIKOFSKY, Elliot J. ; CROSS II., James H.: Reverse Engineering and Design Recovery: A Taxonomy. In: IEEE Software 7 (1990), Januar, Nr. 1, S. 13–17
- 7 Cytron u. a. 1991** CYTRON, Ron ; FERRANTE, Jeanne ; ROSEN, Barry K. ; WEGMAN, Mark N. ; ZADECK, F. K.: Efficiently Computing Static Single Assignment Form and the Control Dependence Graph. In: ACM Transactions on Programming Languages and Systems 13 (1991), Oktober, Nr. 4, S. 451–490
- 8 Dagpinar und Jahnke 2003** DAGPINAR, M. ; JAHNKE, J.: Predicting Maintainability with Object-Oriented Metrics – An Empirical Comparison. In: Working Conference on Reverse Engineering, IEEE Computer Society Press, 2003

- 9 van Deursen u. a. 2004** DEURSEN, Arie van ; HOFMEISTER, Christine ; KOSCHKE, Rainer ; MOONEN, Leon ; RIVA, Claudio: Symphony: View-Driven Software Architecture Reconstruction, IEEE Computer Society Press, Juni 2004, S. 122–132
- 0 Eisenbarth u. a. 2003** EISENBARTH, Thomas ; KOSCHKE, Rainer ; SIMON, Daniel: Locating Features in Source Code. In: IEEE Computer Society Transactions on Software Engineering 29 (2003), März, Nr. 3, S. 210–224
- 1 Fenton und Pfleeger 1996** FENTON, N. ; PFLEEGER, S.: Software Metrics: A Rigorous and Practical Approach. 2nd. London : International Thomson Computer Press, 1996
- 2 Fjedstad und Hamlen 1979** FJEDSTAD, R.K. ; HAMLLEN, W.T.: Application Program Maintenance Study: Report to our Respondents. In: Proceedings of the GUIDE 48. Philadelphia, PA, 1979
- 3 Fowler 2000** FOWLER, Martin: Refactoring: Improving the Design of Existing Code. Addison-Wesley, 2000

- 4 IEEE Std. 1471-2000 2000** IEEE P1471: IEEE Recommended Practice for Architectural Description of Software-intensive Systems—Std. 1471-2000. 2000
- 5 Koschke 2000** KOSCHKE, Rainer: Atomic Architectural Component Recovery for Program Understanding and Universität Stuttgart, Deutschland, Dissertation, 2000
- 6 Koschke u. a. 1998** KOSCHKE, Rainer ; GIRARD, Jean-François ; WÜRTHNER, Martin: An Intermediate Representation for Reverse Engineering Analyses. In: Proceedings of the 5th Working Conference on Reverse Engineering. Honolulu, HI, USA : IEEE Computer Society Press, Oktober 1998, S. 241–250
- 7 Koschke und Simon 2003** KOSCHKE, Rainer ; SIMON, Daniel: Hierarchical Reflexion Models. In: Working Conference on Reverse Engineering, IEEE Computer Society Press, November 2003, S. 36–45
- 8 Lientz und Swanson 1980** LIENTZ, B.P. ; SWANSON, E.B.: Software Maintenance Management. Reading, MA : Addison-Wesley, 1980

- 9 Morgan 1998** MORGAN, Robert: Building an Optimizing Compiler. Digital Press, 1998
- 0 Murphy u. a. 1995** MURPHY, Gail C. ; NOTKIN, David ; SULLIVAN, Kevin: Software Reflexion Models: Bridging the Gap Between Source and High-Level Models. In: Proc. of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM Press, 1995, S. 18–28
- 1 Murphy u. a. 2001** MURPHY, Gail C. ; NOTKIN, David ; SULLIVAN, Kevin J.: Software Reflexion Models: Bridging the Gap between Design and Implementation. In: IEEE Computer Society Transactions on Software Engineering 27 (2001), April, Nr. 4, S. 364–380
- 2 Muthanna u. a. 2000** MUTHANNA, S. ; KONTOGIANNIS, K. ; PONNAMBALAM, K. ; STACEY, B.: A Maintainability Model for Industrial Software Systems Using Design Level Metrics. In: Working Conference on Reverse Engineering, IEEE Computer Society Press, 2000

- 3 Sneed 1995** SNEED, Harry: Planning the Reengineering of Legacy Systems. In: IEEE Software (1995), January. – beschreibt die Planung von Reengineering-Projekten
- 4 Sneed 1999** SNEED, Harry: Risks involved in Reengineering Projects. In: Working Conference on Reverse Engineering, IEEE Computer Society Press, Oktober 1999. – beschreibt typische Risiken bei Reengineering-Projekten
- 5 Snelting und Tip 2000** SNELTING, Gregor ; TIP, Frank: Understanding Class Hierarchies Using Concept Analysis. In: ACM Transactions on Programming Languages and Systems 22 (2000), Mai, Nr. 3, S. 540–582
- 6 Tilley u. a. 1996** TILLEY, S. ; PAUL, S. ; SMITH, D. B.: Towards a Framework for Program Understanding. In: Proceedings of the International Workshop on Program Comprehension, IEEE Computer Society Press, 1996, S. 19–28. – URL <http://www.computer.org/proceedings/wpc/7283/72830019abs.htm>

- 7 Wiggert 1998** WIGGERT, T. A.: Using Clustering Algorithms in Legacy Systems Remodularization. In: Working Conference on Reverse Engineering, IEEE Computer Society Press, 1998, S. 33–43. – Nice introduction to software clustering. Does not introduce a new clustering technique, but gives an overview on the clustering techniques compiled from literature on general clustering (not software related).
- 8 Wilde u. a. 1992** WILDE, Norman ; GOMEZ, Juan A. ; GUST, Thomas ; STRASBURG, Douglas: Locating User Functionality in Old Code. In: Proceedings of the International Conference on Software Maintenance. Orlando, FL, USA : IEEE Computer Society Press, November 1992, S. 200–205