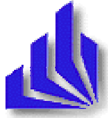
Seminar: Refactoring in eXtreme Programming
WS 2002/2003

Generic Refactoring

Björn Hagemeier <bjoernh@upb.de>

Prof. Dr. Uwe Kastens
Dipl. Inform. Jochen Kreimer



Inhalt

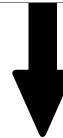
Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- Motivation
- Überblick
- Stratego
- Strafunski
- Generische Refactorings

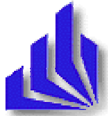
Beispielextraktion in Java

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings


```
void printOwning (double amount) {
    printBanner();
    // print details
    System.out.println("name:" + _name);
    System.out.println("amount:" + amount);
}
```



```
void printOwning(double amount) {
    printBanner();
    printDetails(amount);
}
void printDetails(double amount) {
    System.out.println("name:" + _name);
    System.out.println("amount" + amount);
}
```



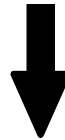
Beispielextraktion in Haskell

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

data *Prog = Prog ProgName [Dec] [Stat]*

data *Dec = Vdec Id Type | ...*

data *Stat = Assign Id Expr | If Expr Stat Stat | ...*



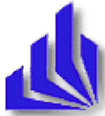
data *Prog = Prog ProgName Block*

data *Block = Block [Dec] [Stat]*

data *Dec = ...*

data *Stat = ... | BlockStat Block*

...

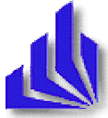


Extraktion

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- Extraktion kann auf verschiedene Paradigmen unmittelbar angewendet werden

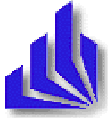
Paradigma	**Fokus**	**Abstraktion**
OOP	Statements	Methode
OOP	Eigenschaften	Klasse
Funktionale Programmierung	Ausdruck	Funktion
Funktionale Programmierung	Typausdruck	Datentyp
Funktionale Programmierung	Funktionen	Typklasse
Logische Programmierung	Literal	Prädikat
Syntax Definition	EBNF Phrase	Nichtterminal
Preprocessing	Code Fragment	Makro
Document Processing	Content Particle	Elementtyp
Cobol Programmierung	Sätze	Absatz
Cobol Programmierung	Sätze	Unterprogramm



Motivation

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- Gründe für generisches Refactoring
  - Dem Refactoring zu Grunde liegende Algorithmen sind meistens gleich
  - Es werden spezielle Implementierungen für unterschiedliche Sprachen benötigt
  - Vorhandene Algorithmen sollen wiederverwendbar sein

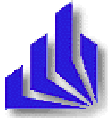


Generisches Refactoring

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- Refactorings können unabhängig von der behandelten Zielsprache beschrieben werden
- Darstellung des Programmcodes auf Modell-Ebene

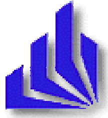
1. Ansatz
   - Anpassung des generischen Algorithmus auf den behandelten Datentyp
   - generisches Verhalten geht verloren
   - sprach-spezifische Eigenheiten können gut berücksichtigt werden
2. Ansatz
   - Darstellung als universelle Datentypen
   - sprach-spezifische Operationen müssen auf Modell-Ebene (universell) beschrieben werden



Stratego

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- Selbständige Sprache
- Gemischter Ansatz der Modellierung
  - angepasste Algorithmen für die Transformationen
  - generische Algorithmen für die Traversierung der Syntaxbäume
- Definition Strategie:
  - Eine Strategie ist ein Programm, das einen Term in einen anderen transformiert. Gelingt dies nicht, gibt es für die Transformation kein Ergebnis.



Darstellung der Programme

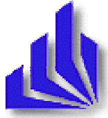
Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- Repräsentation als *Term*
  - C(t1,...,tn), wobei ti wiederum vom Typ *Term*
- Transformationsregeln führen einzelne Transformationen aus
  - *L : t1 -> t2 where s*
  - *<s> t*
- Kombinationen von Strategien
  - sequentielle Komposition *s1;s2*
  - nicht-deterministische Auswahl *s1+s2*
  - Linksauswahl *s1<+s2*
  - Negation *not(s)*
  - Rekursion *rec x(s)*

Strafunski

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

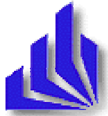
- Framework für generisches Refactoring
  - Abstraktionen, die die Konstrukte unterschiedlicher Paradigmen auf Modell-Ebene darstellen
  - Generische Algorithmen, die Analyse und Transformationen für das Refactoring bereitstellen
  - Generische Algorithmen, die Refactorings auf der Modell-Ebene beschreiben
- Implementiert in Haskell



Funktionale Strategien

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

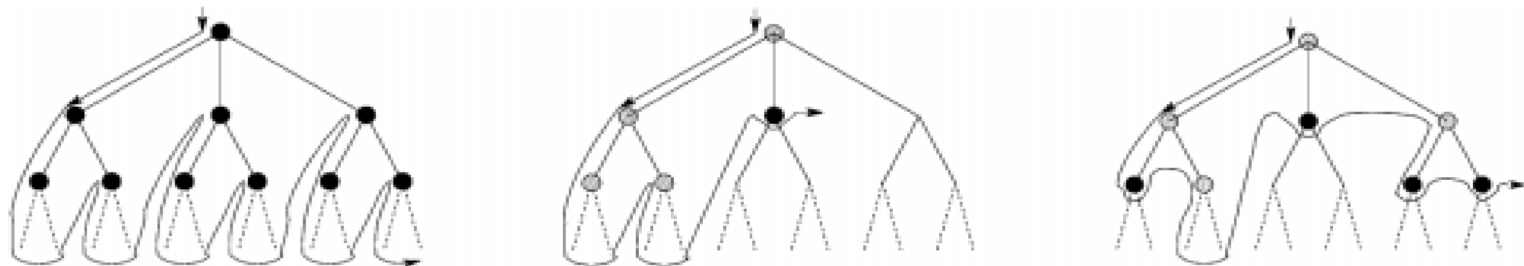
- Definition:
  - typisierte generische Funktionen, die nicht nur auf Terme jeden Typs angewendet werden können, sondern die auch generische Traversierung in Sub-Terme erlauben
- Flexibilität
- Polymorphie

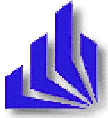


Traversierung

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- Absteigen im Syntax-Baum
- Gleichzeitiges Anwenden von Strategien auf Knoten
  - *full_td*
  - *once_td*
  - *stop_td*





Traversierung

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

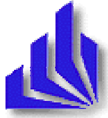
- Ein-Ebenen Traversierung:

$$all \quad :: \quad Monad\ m \Rightarrow MG\ \kappa\ m \rightarrow MG\ \kappa\ m$$

  - Alle Kindknoten abarbeiten

$$one \quad :: \quad MonadPlus\ m \Rightarrow MG\ \kappa\ m \rightarrow MG\ \kappa\ m$$

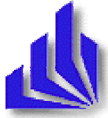
  - linkesten Knoten, der Erfolg erlaubt, abarbeiten
- Rekursive Traversierung lässt sich mit *all* und *one* definieren
  - $allrec\ op\ s \quad = \quad s`op`all(allrec\ op\ s)$
  - *onerec op s = ...*
  - *Oncetd s = ...*
  - ...



Verknüpfung von Strategien

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- binäre Kompositionen von Strategien
  - *f'seq'g* sequentiell
    - führt zunächst Strategie f, dann g auf
  - *f'pass'g* 2 Strategien; Ergebnis der 1. wird an 2. weitergegeben
  - *f'choice'g* alternative Zweige
- weitere Kombinierer:
  - Anwendung von Strategien
  - unparametrisierte Kombinierer
  - Anpassung
  - Ein-Ebenen-Traversierung



Modell

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- Klasse von Abstraktionen
  - Syntaktische Domänen
    - *Term abstr*
    - *Eq Name*
    - *Term [abstr]*
    - *Term apply*
  - Observer
    - *getAbstrName*
    - *getAbstrParas*
    - *...*
  - Instanzen implementieren sprachspezifisches Verhalten

Extraktion freier Variabler

Motivation

Überblick

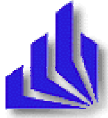
Stratego

Strafunski

Gen. Refactorings

- generischer Algorithmus zur Extraktion freier Variabler und deren Typen
- Parameter
  - initiale Typumgebung
  - Strategien *getvars* und *declvars*

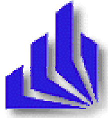
$$
\begin{aligned}
&typed_free_vars :: (MonadPlus\ m, Eq\ v) \\
&\qquad \Rightarrow [(v,t)] \rightarrow TU\ m\ [v] \rightarrow TU\ m\ [(v,t)] \rightarrow TU\ m\ [(v,t)] \\
&typed_free_vars\ env\ getvars\ declvars \\
&\ = afterTU\ (flip\ appendMap\ env)\ (tryTU\ declvars)\text{`}letTU\text{`}\lambda env' \rightarrow \\
&\quad choiceTU\ \ (afterTU\ (flip\ selectMap\ env')\ getvars) \\
&\qquad\qquad (comb\ diffMap\ \ (allTU\ (typed_free_vars\ env'\ getvars\ declvars)) \\
&\qquad\qquad\qquad (tryTU\ declvars))
\end{aligned}
$$



Generische Refactorings

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

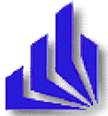
- Extraktion
  - Benutzt Introduction, um die Abstraktion in den Code einzufügen
- Introduction
  - Basis-Transformation, auf der andere Refactorings aufbauen



Generische Extraktion

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

1. Absteigen im gegebenen Programmtext
2. Freie Namen im gegebenen Codefragment bestimmen
3. Sprachspezifische Bedingungen prüfen
4. Abstraktion aus benutzerdefiniertem Namen und freien Namen als Parameter erzeugen
5. Host markieren
6. Introduction
7. Anwendung der neuen Abstraktion erzeugen
8. Fokus durch Anwendung ersetzen



Generische Extraktion

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

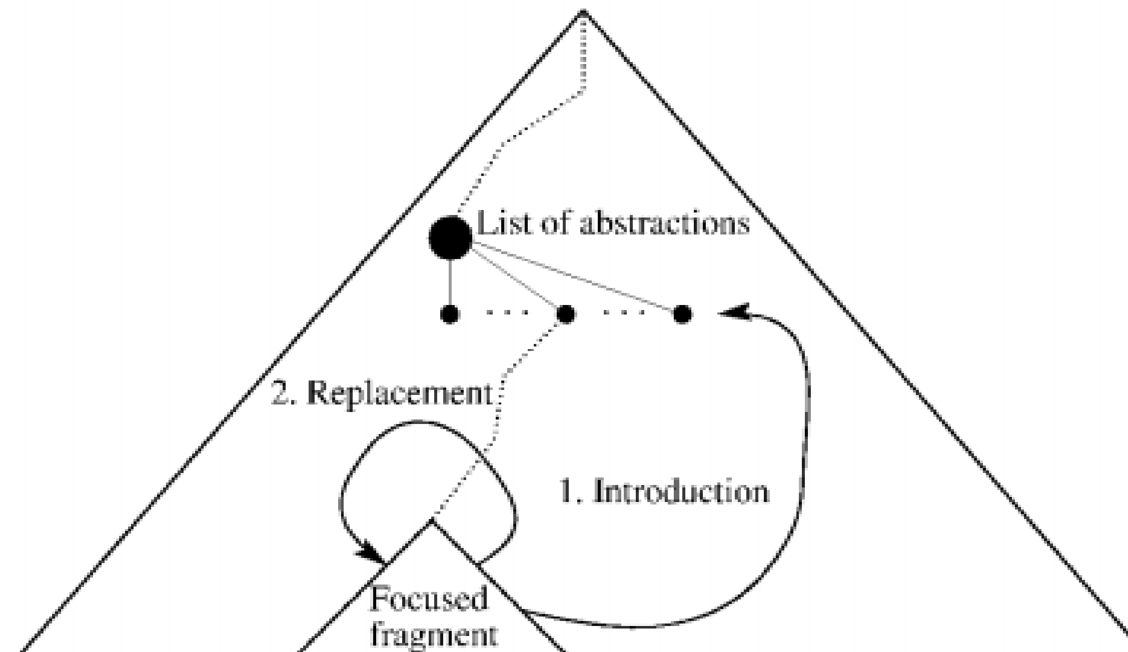


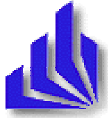
Figure 1. Illustration of abstraction extraction

Generische Extraktion (2)

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- *extract* bekommt folgende Parameter
  - *declared* Deklarationen
  - *referenced* Referenzen
  - *unwrap* Unwrap Fokus
  - *wrap* Wrap Host
  - *unwrap'* Unwrap Host
  - *check* Check Fokus
  - *name* benutzerdefinierter Name
  - *prog* gesamtes Programm

→ sprachabhängige Funktionen (declared, referenced, unwrap, wrap, unwrap', check)

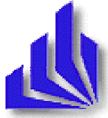


Instanziierung

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- wenig sprach-spezifischer Code
- nur Interface-Funktionen, die für das gewünschte Refactoring notwendig sind
- *extractJoos = extract*

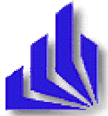
  declaredJoos
  referencedJoos
  unwrapStatement
  wrapMethods
  unwrapMethods
  check
- *check _ f = and [noReturns f, noFrees f]*



Generische Introduction

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

1. Name der gegebenen Abstraktion bestimmen
2. Freie Namen im Fokus bestimmen
3. Definierte Namen bestimmen
4. Wächter einsetzen, um sicher zu stellen, dass die neue Abstraktion nicht mit anderen interferiert
5. Erweiterte Liste von Abstraktionen zurückgeben

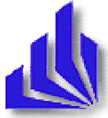


Generische Introduction (2)

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- *introduce* bekommt folgende Parameter
  - *declared* Deklarationen
  - *referenced* Referenzen
  - *unwrap* Unwrap Fokus
  - *abstr* Abstraktion

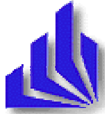
→ sprachabhängige Funktionen (*declared*, *referenced*, *unwrap*)



Zusammenfassung

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

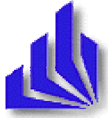
- Refactorings haben für verschiedenste Paradigmen Bedeutung
- Universelle Darstellung der Programme
- generische Funktionen
- Zentrale Elemente
  - generische Term-Traversierung
  - typspezifische Anpassungen (adhoc Kombinierer)
- Beispiele
- Zukunft: Implementierung weiterer generischer Refactorings



Literatur

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

- Ralf Lämmel: Towards Generic Refactoring. In Proc. Of Third ACM SIGPLAN Workshop on Rule-Based Programming RULE 02, Pittsburgh, USA, October 5 2002. ACM Press. 14 pages
- R. Lämmel and J. Visser: Typed Combinators for Generic Traversal. Technical Report SEN-R0124, Centrum voor Wiskunde en Informatica, August 2001. 34 pages
- R. Lämmel and J. Visser: A Strafunski Application Letter. In V. Dahl and P. Wadler, editors, Proc. of Practical Aspects of Declarative Programming 2003 (PADL 03), LNCS. Springer-verlag, January 2003. To appear; 18 pages
- Eelco Visser: Language Independent Traversals for Program Transformation,
  May 2000



Vielen Dank

Motivation
Überblick
Stratego
Strafunski
Gen. Refactorings

Zeit für Fragen