Softwaretechnik II
Vorlesung
Sommersemester 2009

Johannes Waldmann, HTWK Leipzig

15. Juni 2009

Software ist schwer zu entwickeln

;t immaterielles Produkt

nterliegt keinem Verschleiß

icht durch physikalische Gesetze begrenzt

ichter und schneller änderbar als ein technisches rodukt

at keine Ersatzteile

ltert

;t schwer zu vermessen

lzert, Band 1, S. 26 ff)

Inhalt

rogrammieren im Kleinen, Werkzeuge (Eclipse)

rogrammieren im Team, Werkzeuge (CVS, Bugzilla, 'ac)

pezifizieren, Verifizieren, Testen

ntwurfsmuster

efactoring

Organisation

orlesung:

montags, 11:15–12:45, Li 318

bungen (Z423):

dienstags, 15:30–17:00
oder donnerstags 13:45–15:15
oder donnerstags 15:30–17:00

bungsgruppen wählen: `https://autotool.imn.twk-leipzig.de/cgi-bin/Super.cgi`

Einleitung

Programme und Softwaresysteme

ıarakterisierung nach Brooks)

'rogramm: in sich vollständig, kann von seinem Autor auf em System benutzt werden, auf dem es entwickelt 'urde

'omponente eines Systems: Schnittstellen, Integration

ıarktfähiges Produkt: Generalisierung, Tests, okumentation, Wartung

Produktivität

Programmierer schafft etwa

10 (in Worten: zehn)

llständig getestete und dokumentierte) grammzeilen pro Arbeitstag.
h. ≤ 2000 Zeilen pro Jahr)

ses Maß hat sich seit 30 Jahren nicht geändert.
Produktivitätssteigerung nur durch höhere grammiersprachen möglich)

Material

alzert: Lehrbuch der Software-Technik, Band 2, pektrum Akad. Verlag, 2000

ndrew Hunt und David Thomas: The Pragmatic rogrammer, Addison-Wesley, 2000

iamma, Helm, Johnson, Vlissides: Entwurfsmuster, ddison-Wesley, 1996

lartin Fowler: Refactoring, ...

dsger W. Dijkstra: `ttp://www.cs.utexas.edu/users/EWD/`

oel Spolsky: `http://www.joelonsoftware.com/`

Leistungen:

rüfungsvoraussetzung: regelmäßiges und erfolgreiches earbeiten von Übungsaufgaben

gf. in Gruppen (wie im Softwarepraktikum)

rüfung: Klausur

The Pragmatic Programmer

drew Hunt and David Thomas)

are about your craft.

hink about your work.

erantwortung übernehmen. Keine faulen Ausreden (The at ate my source code . . .) sondern Alternativen nbieten.

eparaturen nicht aufschieben. (Software-Entropie.)

eränderungen anregen. An das Ziel denken.

ualitätsziele festlegen und prüfen.

Lernen! Lernen! Lernen!

agmatic Programmer)

eigene *Wissens-Portfolio* pflegen:

egelmäßig investieren

iversifizieren

isiken beachten

illig einkaufen, teuer verkaufen

ortfolio regelmäßig überprüfen

Regelmäßig investieren

agmatic Programmer)

des Jahr wenigstens eine neue Sprache lernen
des Quartal ein Fachbuch lesen
uch Nicht-Fachbücher lesen
/eiterbildungskurse belegen
kale Nutzergruppen besuchen (Bsp: `ttp://gaos.org/lug-l/`)
erschiedene Umgebungen und Werkzeuge usprobieren
ktuell bleiben (Zeitschriften abonnieren, Newsgruppen sen)
ommunizieren

Übung KW 11

Fred Brooks: The Mythical Man Month

uchen Sie (google) Rezensionen zu diesem Buch.

/as ist *Brooks' Gesetz*? ("Adding . . . ")

/as sagt Brooks über Prototypen? ("Plan to . . . ")

/elche anderen wichtigen Bücher zur Softwaretechnik erden empfohlen?

Edsger W. Dijkstra über Softwaretechnik

ijkstra-Archiv `ttp://www.cs.utexas.edu/users/EWD/`

hesen zur Softwaretechnik `http://www.cs.utexas.du/users/EWD/ewd13xx/EWD1305.PDF`

Was macht diese Funktion?

```
olic static int f (int x, int y, int z) {
 (x <= y) {
eturn z;
 else {
eturn
 f (f (x-1,y,z), f(y-1,z,x), f(z-1,x,y));
```

ieviele rekursive Aufrufe finden statt?

ann man das Ergebnis vorhersagen, ohne alle kursiven Aufrufe durchzuführen?

Schnittstellen

Beispiele

schreiben Sie Interfaces (Schnittstellen) im täglichen en:

atterien

D/DVD (Spieler/Brenner, Rohlinge,. . .)

uto(-Vermietung)

.

nittstellen und -Vererbung in der Mathematik:

albordnung, (totale) Ordnung

gl. Beschreibung von `Comparable<E>`

nittstellen zwischen (Teilen von) Softwareprodukten

o sind die Schnittstellen, was wird transportiert? Beispiele)

ie wird das (gewünschte) Verhalten spezifiziert, wie cher kann man sein, daß die Spezifikation erfüllt wird?

nittstellen (interfaces) in Java, Beispiel in Eclipse

clipse (Window → Preference → Compiler → ompliance 6.0) ı A.main: `B x = new B();` , Fehler → Control-1, lasse B anlegen

ı A.main: `x.p();`, Fehler → Control-1, Methode p nlegen

ı B: Refactor → extract interface.

Literatur zu Schnittstellen

ı Pugh: *Interface Oriented Design*, 2006. ISBN 766940-5-0. `http://www.pragmaticprogrammer.` `n/titles/kpiod/index.html`

hält Beispiele:

izza bestellen

nix devices, file descriptors

›xtuelle Schnittstellen

rafische Schnittstellen

Schnittstellen und Verträge

ın jemand eine Schnittstelle implementiert, dann reibt er nicht irgendwelche Methoden mit passenden nen, sondern erfüllt einen Vertrag:

nplementierung soll genau das tun, 'as beschrieben wird.

nplementierung soll nichts nderes, unsinniges, teures, gefährliches tun.

nplementierung soll bescheid geben, 'enn Auftrag nicht ausführbar ist.

p: Pizzafehlermeldung)

Design by Contract

rand Meyer: *Object-Oriented Software Construction*, ntice Hall, 1997.

`:p://archive.eiffel.com/doc/oosc/`

›ekte eines Vertrages:

orbedingungen

achbedingungen

lassen-Invarianten

Schnittstellen und Tests

n überzeuge sich von

enutzbarkeit einer Schnittstelle (unabhängig von nplementierung)

. wird das gewünschte Ergebnis durch eine Folge von lethodenaufrufen vertraglich garantiert?

orrekheit einer Implementierung

gliche Argumentation:

›rmal (Beweis)

›stend (beispielhaft)

. benutze *Proxy*, der Vor/Nachbedingungen auswertet

Stufen von Verträgen

ch K. Pugh)

ypdeklarationen

ormale Spezifikation von Vor- und Nachbedingungen

eistungsgarantien (für Echtzeitsysteme)

ienstgüte-Garantien (quality of service)

Typen als Verträge

r Typ eines Bezeichners ist seine beste Dokumentation.

nn der Compiler kann sie prüfen!)

sind Sprachen (und ihre Sprecher) arm dran, deren system ausdrucksschwach ist.

```
: a [] = { "foo", 42 }; // ??

 Mittelalter-Java:
st l = new LinkedList ();
ıdd ("foo"); l.add (42);

 korrektes Java:
st<String> l = new LinkedList<String> ();
```

Arten von Schnittstellen

s wird verwaltet?

chnittstellen für Daten

/ethoden lesen/schreiben Attribute)

chnittstellen für Dienste

/ethoden „arbeiten wirklich“)

Schnittstellen zum Datentransport

ressierung:

ahlfreier Zugriff (Festplatte)

equentieller Zugriff (Magnetband)

nsportmodus:

ull (bsp. Web-Browser)

ush (bsp. Email)

): SAX und DOM einordnen

Schnittstellen und Zustände

chnittstelle *ohne* Zustand

Vorteil: Aufrufreihenfolge beliebig
Nachteil: mehr Parameter (einer?)

chnittstelle *mit* Zustand

Nachteil: Reihenfolge wichtig
Vorteil: weniger Parameter

Mehrfache Schnittstellen

e Klasse kann mehrere Schnittstellen implementieren
ren Verträge erfüllen).
ın aber Vorsicht bei der Bezeichnung der Methoden.
und beim Verwechseln von Zuständen (Bsp. Pizza/Eis)

Beispiel: Sortieren

wesentliche Bestandteile

```
olic class Zahl
 implements Comparable<Zahl> {
    public int compareTo(Zahl that) { .. }

ıl [] a =
 { new Zahl (3), new Zahl (1), new Zahl (4)
rays.sort(a);
```

Klassen-Entwurf

ahl hat ein `private final` Attribut,

ird im Konstruktor gesetzt

ahl implementiert `String toString()`,

ann funktioniert

```
ystem.out.println(Arrays.asList(a));
```

Richtig vergleichen

; sieht clever aus, ist aber nicht allgemeingültig:

```
 public int compareTo(Zahl that) {
      return this.contents - that.contents;
  }
```

erke: *avoid clever code*)

Protokollierung mit Dekorator

gabe:

lle Aufrufe von `Zahl.compareTo` protokollieren...

hne den Quelltext dieser Klasse zu ändern!

sung: eine Klasse dazwischenschieben

```
ass Log<E> ... {
  private final contents E;
  int compareTo(Log<E> that) { .. }

g<Zahl> a [] =
  { new Log<Zahl> (new Zahl (13)), .. };
```

se Klasse heißt *Dekorator*, das ist ein Beispiel für ein

Entwurfsmuster: allgemein

Entwurfsmuster

ch Gamma, Richard Helm, Ralph Johnson, John
ssides: *Entwurfsmuster (design patterns)* — Elemente
derverwendbarer objektorientierter Software,
dison-Wesley 1996.
ert Muster (Vorlagen) für Gestaltung von Beziehungen
schen (mehreren) Klassen und Objekten, die sich in
derverwendbarer, flexibler Software bewährt haben.

minarvorträge `http://www.imn.htwk-leipzig.`
`/~waldmann/edu/ss05/case/seminar/`

Beispiel zu Entwurfsmustern

- s: Gamma, Helm, Johnson, Vlissides: *Entwurfsmuster*
- spiel: ein grafischer Editor.
- ɔekte sind unter anderem:
  - okumentstruktur
  - ormatierung
  - enutzungsschnittstelle

Beispiel: Strukturmuster: Kompositum

- stellbare Elemente sind:
  - chstabe, Leerzeichen, Bild, Zeile, Spalte, Seite
- meinsamkeiten?
- erschiede?

Beispiel: Verhaltensmuster: Strategie

- matieren (Anordnen) der darstellbaren Objekte:
- glicherweise linksbündig, rechtsbündig, zentriert

Beispiel: Strukturmuster: Dekorierer

- stellbare Objekte sollen optional eine Umrahmung oder
e Scrollbar erhalten

Beispiel: Erzeugungsmuster: (abstrakte) Fabrik

- wendung soll verschiedene GUI-Look-and-Feels
nöglichen

Beispiel: Verhaltensmuster: Befehl

- nü-Einträge, Undo-Möglichkeit
- ne auch Ereignisbehandlung in Applets

Wie Entwurfsmuster Probleme lösen

- inden passender Objekte
  - nsbesondere: nicht offensichtliche Abstraktionen
- estimmen der Objektgranularität
- pezifizieren von Objektschnittstellen und
Objektimplementierungen
  - nterscheide zwischen *Klasse* (konkreter Typ) und *Typ*
  abstrakter Typ).
  - rogrammiere auf eine Schnittstelle hin, nicht auf eine
  nplementierung!
- Viederverwendungsmechanismen anwenden
  - ehe Objektkomposition der Klassenvererbung vor

Vorlage: Muster in der Architektur

- ristopher Alexander: *The Timeless Way of Building, A
tern Language*, Oxford Univ. Press 1979.
- Menschen formen Gebäude und benutzen dazu
ster-Sprachen. Wer eine solche Sprache spricht, kann
nit unendlich viele verschiedene einzigartige Gebäude
stellen, genauso, wie man unendlich viele verschiedene
ze in der gewöhnlichen Sprache bilden kann.
- ... müssen wir tiefe Muster entdecken, die Leben
eugen können.
- ... diese Muster verbreiten und verbessern, indem wir
testen: erzeugen sie in uns das Gefühl, daß die

. . . einzelne Muster verbinden zu einer Sprache für
visse Aufgaben . . .

. . . verschiedene Sprachen zu einer größeren Struktur
binden: der gemeinsamen Sprache einer Stadt.

In Wirklichkeit hat das Zeitlose nichts mit Sprachen zu
. Die Sprache beschreibt nur die natürliche Ordnung der
ge. Sie lehrt nicht, sie erinnert uns nur an das, was wir
on wissen und immer wieder neu entdecken . . .

de wenigstens sieben (Entwurfs-)Fehler und ihre
hrscheinlichen Auswirkungen. . .

gebraische Datentypen (Komposita)

Strukturmuster: Kompositum

gabe: verschiedene (auch zusammengesetzte)
metrische Objekte

ve Lösung ohne Entwurfsmuster:

```
ass Geo {
  int type; // Kreis, Quadrat,
  Geo teil1, teil2; // falls Teilobjekte
  int ul, ur, ol, or; // unten links, ...
  void draw () {
      if (type == 0) { ... } // Kreis
      else if (type == 1) { ... } // Quadra
  }
```

Kompositum - Anwendung

st es richtig:

```
terface Geo {
  Box bounds ();
  Geo [] teile ();
  void draw ();

ass Kreis implements Geo { .. }
ass Neben implements Geo {
  Neben (Geo links, Geo Rechts) { .. }
```

Signaturen und Algebren

nehrsortige) Signatur Σ

Menge von Funktionssymbolen, für jedes: Liste von
rgumenttypen, Resultattyp

ist Σ-Algebra:

rägermenge und typkorrekte Zuordnung von
unktionssymbolen zu Funktionen

eispiel: Signatur für Vektorraum V über Körper K

Termalgebra (Bäume)

eder Signatur Σ kann man die Algebra $\mathrm{Term}(\Sigma)$
struieren:

rägermenge sind alle typkorrekten Σ-Terme

edes Symbol $f \in \Sigma$ wird durch „sich selbst“
mplementiert

lerer Name für diese Algebra: *algebraischer Datentyp*.

Algebraische Datentypen

isten:

```
ata List a = Cons a (List a) | Nil
```

äume (mit Schlüsseln in Blättern):

```
ata Tree a = Branch (Tree a) (Tree a)
           | Leaf a
```

bung: Peano-Zahlen, Wahrheitswerte

: *Kompositum* = rekursiver algebraischer Datentyp

Entwurfsfragen bei Bäumen

noten sind *innere* (Verzweigung) und *äußere* (Blatt).

ie “richtige” Realisierung ist Kompositum

```
nterface Tree<K>;
lass Branch<K> implements Tree<K>;
lass Leaf<K> implements Tree<K>;
```

löglichkeiten für Schlüssel: in allen Knoten, nur innen,
ur außen.

nn Blätter keine Schlüssel haben, geht es musterfrei?

```
ass Tree<K> // Branch, mit Leaf == null;
```

n. — betrachte Implementierung in

Rekursionsmuster (Besucher)

Verhaltensmuster: Besucher

ızip:

- ekursive Datenstruktur (algebraischer Datentyp, ompositum)
- ekursionsmuster für Algorithmen, die diese Struktur enutzen.

ılementierungen:

- ıap/fold in Haskell (funktional)
- esucher in Java (objektorientiert)
- elect/Aggregate in C# (funktional)

Wiederholung Kompositum

skell: algebraischer Datentyp

```
:a List a  = Nil | Cons a (List a)
   Nil  :: List a
   Cons :: a -> List a -> List a
```

a: Kompositum

```
:erface List<A> { }
ass Nil implements List<A> { }
ass Cons<A> implements List<A> {
  A head; List<A> tail;
```

ılementieren/testen:

```
olic class Main {
List<Integer> range(from,to) { .. }
```

```
oublic static void main(String[] args) {
  List<Integer> l = Main.range(0,10);
}
```

Wiederholung Listen

zeichnungen bisher:

```
:a List a  = Nil | Cons a (List a)
   Nil  :: List a
   Cons :: a -> List a -> List a
```

Haskell-Prelude schon vordefiniert:

```
:a     [a] = []   |      a : [a]
```

wendung:

```
oort Prelude hiding ( length )
ıgth :: [a] -> Int
ıgth l = case l of
  []     -> 0
  x : xs -> 1 + length xs
```

Operationen auf Listen

```
ıgth :: [a] -> Int
ıgth l = case l of
  x : xs -> 1 + length xs
  []     -> 0
n :: [Int] -> Int
n l = case l of
  x : xs -> x + sum xs
  [] -> 0
::              ->
l = case l of
  x : xs -> g x (f xs)
  [] -> h
```

urch *Rekursionsschema* mit Parametern `g` und `h`

Rekursionsschema foldr

```
ldr :: (a -> r -> a) -> a -> [r] -> a
ldr g h l = case l of
  x : xs -> g x (foldr g h xs)
  []     -> h
n      = foldr ( \ x y -> x + y ) 0
ıgth   =
oduct =
```

Kompositum und Visitor

inition eines Besucher-Objektes

```
:erface Visitor<A,R> {
R empty();
R nonempty(A here, R later);
```

ıandlung eines Besuchers: durch jeden Teilnehmer des npositums

```
:erface List<A> { ..
<R> R visit (Visitor<A,R> v);
```

nplementierung

Besucher (Aufgabe)

reibe Methoden für

- rodukt
- linimum, Maximum
- Vert als Binärzahl, Bsp:

```
peration.binary
  (new List<Integer>(1,1,0,1)) ==> 13
```

Desgleichen für Bäume

ebraische Datentyp:

```
:a Tree k = Leaf { key  :: k }
 | Branch { left :: Tree k
          , right :: Tree k }
```

npositum:

```
:erface Tree<K> { }
ass Leaf<K>   implements Tree<K> {
Leaf(E key) { .. }

ass Branch<K> implements Tree<K> {
Branch(Tree<K> left, Tree<K> right) { .. }
```

Bäume (Aufgabe I)

nstruktoren, toString, Testmethode

```
ass Trees {
  // vollst. bin. Baum der Höhe h
  static Tree<Integer> full (int h);

stem.out.println (Trees.full(1))
 ==> Branch{left=Leaf{key=0},right=Leaf{key
```

Besucher für Bäume (Komposita)

eses Beispiel sinngemäß aus: Naftalin, Wadler: Java
nerics and Collections, O'Reilly 2006.)

jeden Teilnehmer des Kompositums eine Methode:

```
:erface Visitor<K,R> {
  // mit Resultattyp R
  R leaf (K x);
  R branch (R left, R right);
```

Gast nimmt Besucher auf:

```
:erface Tree<K> {
<R> R visit (Visitor<K,R> v)
```

Bäume (Aufgabe II)

nutzung des Besuchers: Anzahl der Blätter:

```
ass Trees {
static <K> int leaves (Tree<K> t) {
  return t.visit(new Tree.Visitor<K,Integer
    public Integer branch
         (Integer left, Integer right) {
      return left + right;
    }
    public Integer leaf(K key) {
      return 1;
    }
  });
}
```

Funktionale Programmierung in C#

lr = Aggregate

```
port System.Linq;
port System.Collections.Generic;

st<int> l =
  new List<int>() { 3,1,4,1,5,9 };
nsole.WriteLine
  (l.Aggregate(0, (x,y) => x+y));
```

Datenströme (Iteratoren)

Verhaltensmuster: Iterator

lotivation (Streams)

efinition Iterator

yntaktische Formen (foreach, yield return)

aumdurchquerung mit Stack bzw. Queue

Unendliche Datenstrukturen

```
turals :: [ Integer ]
turals = from 0 where
  from x = x : from (x+1)
imes :: [ Integer ]
```

; ist möglich, wenn der *tail* jeder Listenzelle erst bei
darf erzeugt wird.

darfsauswertung, lazy evaluation)

/ Liste = Stream = Pipeline, vgl. InputStream (Console)

Rechnen mit Streams

x:

```
: stream.tex | tr -c -d aeuio | wc -m
```

skell:

```
n $ take 10 $ map ( \ x -> x^3 ) $ naturals
```

```
umerable.Range(0,10).Select(x=>x*x*x).Sum();
```

gische Trennung:
roduzent → Transformator(en) → Konsument

egen Speichereffizienz: verschränkte Auswertung.

ibt es bei *lazy* Datenstrukturen geschenkt, wird
nsonsten durch Iterator (Enumerator) simuliert.

Iterator (Java)

```
erface Iterator<E> {
oolean hasNext(); // liefert Status
E next(); // schaltet weiter

erface Iterable<E> {
Iterator<E> iterator();
```

sche Verwendung:

```
erator<E> it = c.iterator();
ile (it.hasNext()) {
E x = it.next (); ...
```

ürzung: `for (E x : c) { ... }`

Beispiel Iterator Java

```
atic <E extends Comparable<E>>
st<E> merge(List<E> xs, List<E> ys) {
st<E> zs = new LinkedList<E>();
erator<E> xi = xs.iterator();
erator<E> yi = ys.iterator();
      // FIXME:
ile (xi.hasNext() && yi.hasNext()) {
x = xi.next();
y = yi.next();
 (x.compareTo(y) < 0) {
.add(x);
else {
.add(y);
```

```
urn zs;
```

Enumerator (C#)

```
erface IEnumerator<E> {
E Current; // Status
ool MoveNext (); // Nebenwirkung

erface IEnumerable<E> {
IEnumerator<E> GetEnumerator();
```

sche Benutzung: ...
ürzung: `foreach (E x in c) { ... }`

Beispiel Enumerator (C#)

```
atic IList<int>
ge (IList<int> xs, IList<int> ys) {
  IList<int> zs = new List<int>();
  IEnumerator<int> xi = xs.GetEnumerator();
  IEnumerator<int> yi = ys.GetEnumerator();
  bool xgo = xi.MoveNext();
  bool ygo = yi.MoveNext();
  // FIXME
  while (xgo && ygo) {
      if (xi.Current < yi.Current) {
       zs.Add(xi.Current);
       xgo=xi.MoveNext();
      } else {
```

```
       zs.Add(yi.Current);
       ygo=yi.MoveNext();
      }
  }
  return zs;
```

ungen:

eparieren

chtigen (generischen) Typ deklarieren

eld return benutzen

Iteratoren mit yield

```
ass Range : IEnumerable<int> {
  private readonly int lo;
  private readonly int hi;
  public Range(int lo, int hi) {
      this.lo = lo; this.hi = hi;
  }
  public IEnumerator<int> GetEnumerator() {
      for (int x = lo; x < hi ; x++) {
         yield return x;
      }
      yield break;
  }
```

Streams in C#: funktional, Linq

ıktional

```
ıumerable.Range(0,10).Select(x => x^3).Sum();
```

ı von Select? Implementierung?

ą-Schreibweise:

```
:om x in new Range(0,10) select x*x*x).Sum();
```

ıchte: SQL-select „vom Kopf auf die Füße gestellt".

Befehl, Strategie, Interpreter

Befehl

spiel:

```
:erface ActionListener {
/oid actionPerformed( ActionEvent e);
```

```
ıtton b = new JButton ();
ıddActionListener (new ActionListener() {
ɔublic void actionPerformed (ActionEvent e)
;
```

ınt Befehls-Erzeugung von -Ausführung,
ıöglicht Verarbeitung von Befehlen (auswählen,
ichern, wiederholen)

Strategie

ıfter benutzter Befehl, mit Parametern

spiel:

```
:erface Comparator<T> { int compare (T x, T
st<Integer> xs = ...;
Llections.sort
  (xs, new Comparator<Integer>() { ... });
```

ıng:

ɔrtiere Strings länge-lexikografisch, . . .

'o wird Transitivität, Linearität der Relation benutzt?

Strategie (Beispiel II)

```
ɔlic class Strat extends JApplet {
ɔublic void init () {
  JPanel p = new JPanel
     (new GridLayout(8,0)); // Strategie-Obje
  for (int i=0; i<40; i++) {
    p.add (new JButton ());
  }
  this.getContentPane().add(p);
}
```

ıerkungen: Kompositum (Wdhlg), MVC (später)

Muster: Interpreter (Motivation)

dhlg. Iterator)

```
ım Colour { Red, Green, Blue }
ıss Car { int wheels; Colour colour, }
ıss Store {
Collection<Data> contents;
Iterable<Data> all ();
```

veit klar, aber wie macht man das besser:

```
ıss Store { ...
Iterable<Data> larger_than_5 ();
Iterable<Data> red ();
Iterable<Data> green_and_even ();
```

Muster: Interpreter (Realisierung)

ebraischer Datentyp (= Kompositum) für die
schreibung von Eigenschaften

```
:erface Property { }
```

tter (Konstanten)

```
ıss Has_Color { Color c }
ıss Less_Than { int x }
```

zweigungen (Kombinatoren)

.

l Programm zur Auswertung einer
sammengesetzten) Eigenschaft für gegebenes Datum.
ɔ?)

Interpreter (Material)

ıelltext aus der Vorlesung: `http://dfa.imn.htwk-leipzig.de/cgi-bin/cvsweb/st09/src/w18/Store.java?rev=1.1;cvsroot=pub`

ann insgesamt als Eclipse-Projekt importiert werden:
ile → New → Project → from CVS,

```
onnection type: pserver
ser: anonymous
ost: dfa.imn.htwk-leipzig.de
ort: default
ath: /var/lib/cvs/pub
```

ukunft *jetzt* sichern! Wahlfach WS09/10 wählen!
ttp://www.imn.htwk-leipzig.de/~waldmann/
ehre.html

Query-Sprachen

L: domainspezifische Sprache, hier: für
enbankabfragen

xterne DSL (Frage = String)

```
erson aPerson = (Person) session
.createQuery("select p from Person p left j
   where p.id = :pid")
.setParameter("pid", personId)
```

mbedded DSL (Frage = Objekt)

psichere embedded DSL

jar keine Datenbank: http://happstack.com/)

Hibernate Criteria Query API

:p://www.hibernate.org/

```
ɔort org.hibernate.criterion.Criterion; ...
st cats = sess.createCriteria(Cat.class)
  .add( Restrictions.like("name", "Fritz%") )
  .add( Restrictions.between
          ("weight", minWeight, maxWeight) )
  .list();
```

Linq in C#

```
ıumerable<Car> cars = new List<Car>()
 { new Car(){wheels = 4,
            colour = Colour.Red},
   new Car(){wheels = 3,
            colour = Colour.Blue} };
:each (var x in from c in cars
ıere c.colour == Colour.Red
elect c.wheels) {
 System.Console.WriteLine (x);
```

:p://msdn.microsoft.com/en-us/
:framework/aa904594.aspx
:enquellen: Collections, XML, DB

Zustand, DI, Beobachter, MVC

Entwurfsmuster: Zustand

stand eines Objektes = Belegung seiner Attribute
stand erschwert Programm-Benutzung und -Verifikation
ıß bei jedem Methodenaufruf berücksichtigt werden).
ıilfe: Trennung in

ustandsobjekt (nur Daten)

andlungsobjekt (nur Methoden)

e Methode bekommt Zustandsobjekt als Argument

Zustand (Beispiel)

```
ass C0 { // Zustand implizit
ɔrivate int z = 0;
ɔublic void step () { this.z++; }

ass C1 { // Zustand explizit
ɔublic int step (int z) { return z + 1; }
```

se Zustandsobjekte sind aber unsicher

Zustand in Services

nveränderliche Zustandsobjekte:

erwendung früherer Zustandsobjekte (undo, reset, test)

derverwendbare Komponenten („Software als Service“)
fen *keinen* Zustand enthalten.
read-Sicherheit, Load-Balancing usw.)
l.: Unterprogramme dürfen keine globalen Variablen
ıutzen)

ler (reinen) funktionalen Programmierung passiert das
selbst: dort *gibt es keine Zuweisungen* (nur
ıst-Deklarationen mit einmaliger Initialisierung).

Dependency Injection

rtin Fowler, http://www.martinfowler.com/
:icles/injection.html
ıängigkeiten zwischen Objekten sollen

chtbar und

onfigurierbar sein (Übersetzung, Systemstart, Laufzeit)

men:

onstructor injection (bevorzugt)

etter injection (schlecht—dadurch sieht es wie „Zustand“
us, unnötigerweise)

Verhaltensmuster: Beobachter

Programmierung von Reaktionen auf
standsänderung von Objekten

ubjekt: class Observable

anmelden: void addObserver (Observer o)
abmelden: void deleteObserver (Observer o)
Zustandsänderung: void setChanged ()
Benachrichtigung: void notifyObservers(. . .)

eobachter: interface Observer

aktualisiere: void update (. . .)

ekbeziehungen sind damit konfigurierbar.

Beobachter: Beispiel (I)

```
plic class Counter extends Observable {
  private int count = 0;
  public void step () { this.count ++;
      this.setChanged();
      this.notifyObservers();   }   }
plic class Watcher implements Observer {
  private final int threshold;
  public void update(Observable o, Object a
     if (((Counter)o).getCount() >= this.th
         System.out.println ("alarm");  }
plic static void main(String[] args) {
  Counter c = new Counter (); Watcher w = n
  c.addObserver(w); c.step(); c.step (); c.
```

Beobachter: Beispiel (II)

raktiver Sudoku-Solver:

des Feld F hat Zustand (leer oder besetzt durch Zahl)

de Teilmenge M (Zeile, Spalte, Block)
at Zustand (Menge der bereits benutzten Zahlen)

des M beobachtet alle F mit $F \in M$

edes F beobachtet alle M mit $F \in M$)

gen:

ustandsänderungen zurücknehmen

yklische Abhängigkeiten

iUI

Model/View/Controller

odell/Anzeige/Steuerung)
gl. *to control* = steuern, *nicht:* kontrollieren)

standteile (Beispiel):

lodel: Counter (getCount, step)

iew: JLabel (← getCount)

ontroller: JButton (→ step)

sammenhänge:

ontroller steuert Model

iew beobachtet Model

javax.swing und MVC

ing benutzt vereinfachtes MVC
getrennt, aber V und C gemeinsam).

ratur:

he Swing Tutorial `http://java.sun.com/docs/`
`ooks/tutorial/uiswing/`

iuido Krüger: Handbuch der Java-Programmierung,
ddison-Wesley, 2003, Kapitel 35–38

Swing: Datenmodelle

```
lider top = new JSlider(JSlider.HORIZONTAL,
lider bot = new JSlider(JSlider.HORIZONTAL,
:.setModel(top.getModel());
```

gabe: unterer Wert soll gleich 100 - oberer Wert sein.

Swing: Bäume

```
 Model:
ass Model implements TreeModel { .. }
eeModel m = new Model ( .. );

 View + Controller:
ree t = new JTree (m);

 Steuerung:
addTreeSelectionListener(new TreeSelectionLis
 public void valueChanged(TreeSelectionEvent

 Änderungen des Modells:
```

Quelltextverwaltung mit CVS

Anwendung, Ziele

ktuelle Quelltexte eines Projektes sichern

uch frühere Versionen sichern

leichzeitiges Arbeiten mehrere Entwickler

. . an unterschiedlichen Versionen

s Management bezieht sich auf *Quellen* (.c, .java, .tex,
kefile)

jeleitete Dateien (.obj, .exe, .pdf, .class) werden daraus

CVS-Überblick

ncurrent version system)

erver: Archiv (repository), Nutzer-Authentifizierung
gf. weitere Dienste (`cvsweb`)

lient (Nutzerschnittstelle): Kommandozeile
`vs checkout foobar` oder grafisch (z. B. intergriert
ı Eclipse)

Archiv (repository) besteht aus mehreren Modulen (=
zeichnissen)
lokale Kopie der (Sub-)Module beim Clienten heißt
ıdkasten (sandbox).

CVS-Tätigkeiten (I)

Projektbeginn:

erver-Admin:

Repository und Accounts anlegen (`cvs init`)

lienten:

neues Modul zu Repository hinzufügen (`cvs import`)
Modul in sandbox kopieren (`cvs checkout`)

CVS-Tätigkeiten (II)

ırend der Projektarbeit:

lienten:

vor Arbeit in sandbox: Änderungen (der anderen Programmierer) vom Server holen (`cvs update`)
nach Arbeit in sandbox: eigene Änderungen zum Server schicken (`cvs commit`)

Konflikte verhindern oder lösen

in Programmierer: editiert ein File, oder editiert es nicht.

ıehrere Programmierer:

strenger Ansatz: nur einer darf editieren
beim checkout wird Datei im Repository markiert (gelockt), bei commit wird lock entfernt
nachgiebiger Ansatz (CVS): jeder darf editieren, bei commit prüft Server auf Konflikte
und versucht, Änderungen zusammenzuführen (merge)

Welche Formate?

ɿuellen sollen Text-Dateien sein, human-readable, mit
eilenstruktur: ermöglicht Feststellen und
usammenfügen von unabhängigen Änderungen
rgibt Konflikt mit Werkzeugen (Editoren, IDEs), die
okumente nur in Binärformat abspeichern. — Das ist
ɔwieso *evil*, siehe Robert Brown: Readable and Open
ile Formats, `http://www.few.vu.nl/~feenstra/`
`ead_and_open.html`
rogramme mit grafischer Ein- und Ausgabe sollen
ıformationen *vollständig* von und nach Text konvertieren
önnen
3sp: UML-Modelle als XMI darstellen)

Logging (I)

commit soll ein Kommentar angegeben werden, damit
n später nachlesen kann, welche Änderungen aus
chem Grund ausgeführt wurden.

clipse: textarea

```
vs commit -m "neues Feature: --timeout"
```

```
macs -f server-start &
xport EDITOR=emacsclient
vs commit
```

rgibt neuen Emacs-Buffer, beenden mit `C-x #`

Logging (II)

Log-Messages für eine Datei:

```
s log foo.c
```

Log-Message soll den *Grund* der Änderung enthalten,
ın den *Inhalt* kann man im Quelltext nachlesen:

```
s diff -D "1 day ago"
```

le entsprechendes Eclipse-Kommando!

Authentifizierung

ıkal (Nutzer ist auf Server eingeloggt):

```
xport CVSROOT=/var/lib/cvs/foo
vs checkout bar
```

ɘmote, unsicher (Paßwort unverschlüsselt)

```
xport CVSROOT=:pserver:user@host:/var/lib/c
vs login
```

ɘmote, sicher

```
xport CVS_RSH=ssh2
```

Authentifizierung mit SSH/agent

chlüsselpaar erzeugen (ssh-keygen)

ffentlichen Schlüssel auf Zielrechner installieren
sh-copy-id)

rivaten Schlüssel in Agenten laden (ssh-add)

Subversion

:p://subversion.tigris.org/ — “a better CVS”

hnliche Kommandos, aber anderes Modell:

lient hat Sandbox *und* lokale Kopie des Repositories
eswegen sind weniger Server-Kontakte nötig

ommits are atomic” (CVS: commit einer einzelnen Datei
t atomic)

ersionsnummer bezieht sich auf Repository (nicht auf
inzelne Dateien)

Sandbox sind Dateien verschiedener Revisionen
estattet

Subversion (II)

erver speichert Dateien und Zusatz-Informationen in
atenbank (Berkeley DB) (CVS: im Filesystem)

nterstützt auch Umbenennen usw. mit Bewahrung der
istory.

ubversion läuft als standalone-Server oder als
pache2-Modul (benutzt WebDAV)

ommandozeilen-Client wie cvs, Grafische Clients
TortoiseSVN), Webfrontends (viewCVS/viewSVN)

itere Erläuterungen zu Subversion im Vortrag von Enrico
mer (Seminar Software-Entwicklung) `http://www.`
`.htwk-leipzig.de/~waldmann/edu/ss04/se/`

Darcs

vid Roundy, `http://darcs.net/`

icht Verwaltung von *Versionen*, sondern von *Patches*

estattet paralleles Arbeiten an verschiedenen Versionen

ein zentrales Repository

kann jedoch vereinbart werden)

Oberseminarvortrag

Übung CVS

in CVS-Archiv ansehen (cvsweb-interface)
`ttp://dfa.imn.htwk-leipzig.de/cgi-bin/`
`vsweb/havannah/different-applet/?cvsroot=`
`avannah`

in anderes Modul aus o. g. Repository anonym
uschecken (mit Eclipse):

Host: `dfa.imn.htwk-leipzig.de`, Pfad:
`var/lib/cvs/havannah`, Modul `demo`, Methode:
`server`, User: `anonymous`, kein Passwort)

rojekt als Java-Applet ausführen. . . . zeigt Verwendung
on Layout-Managern.

pplet-Fenster-Größe ändern (ziehen mit Maus).

och weiter Komponenten (Buttons) und Panels (mit
igenen Managern) hinzufügen.

in eigenes Eclipse-Projekt als Modul zu dem
ruppen-eigenen CVS-Repository hinzufügen (Team →
hare)

Daten ggf. für laufendes Semester/Server anpassen.]

ost: `cvs.imn.htwk-leipzig.de`, Pfad:
`cvsroot/case05_XX`, `XX` = Ihre Gruppennummer

CVS-Zugang benutzt Account im Linux-Pool,
ruppeneinteilung beachten)

ine Datei ändern, commit; anderer Student gleicher
ruppe: update

as passiert bei gleichzeitigen Änderungen und

CVS – Einzelheiten

Datei-Status

`s status ; cvs -n -q update`

p-to-date:
atei in Sandbox und in Repository stimmen überein
ocally modified (, added, removed):
kal geändert (aber noch nicht commited)
eeds Checkout (, Patch):
n Repository geändert (wg. unabh. commit)
eeds Merge:
okal geändert *und* in Repository geändert

CVS – Merge

:00 Heinz: checkout (Revision A)

:10 Klaus: checkout (Revision A)

:20 Heinz: editiert ($A \to H$)

:30 Klaus: editiert ($A \to K$)

:40 Heinz: commit (H)

:50 Klaus: commit

`p-to-date check failed`

:51 Klaus: update

`erging differences between` A `and` H `into` K

:52 Klaus: commit

Drei-Wege-Diff

ıutzt Kommando `diff3` K A H

ıanges von $A \to H$ berechnen

. und auf K anwenden (falls das geht)

ıflikte werden in K (d. h. beim Clienten) markiert
ł müssen vor dem nächsten commit repariert werden.

ıächlich wird `diff3` nicht als externer Prozeß
gerufen, sondern als internes Unterprogramm
unabhängig vom Prozeß-Begriff des jeweiligen OS)

ehe auch Diskussion hier:
`ttp://c2.com/cgi/wiki?DiffAlgorithm`

Unterschiede zwischen Dateien

ʼelche Zeilen wurden geändert, gelöscht, hinzugefügt?

hnliches Problem beim Vergleich von DNS-Strängen.

lgorithmus: Eugene Myers: *An $O(ND)$ Difference lgorithm and its Variations*, Algorithmica Vol. 1 No. 2, 986, pp. 251-266,
`ttp://www.xmailserver.org/diff2.pdf`

nplementierung (Richard Stallman, Paul Eggert et al.):
`ttp://cvs.sourceforge.net/viewcvs.py/`
`checkout*/cvsgui/cvsgui/cvs-1.10/diff/`
`nalyze.c`

LCS

e: die beiden Aufgaben sind äquivalent:

ürzeste Edit-Sequenz finden

ıngste gemeinsame Teilfolge (longest common ıbsequence) finden

spiel: $y = AB\boxed{C}\boxed{AB}B\boxed{A}, z = \boxed{C}B\boxed{AB}\boxed{A}C$
$x = CABA$ gilt $x \le y$ und $x \le z$,
ɔei die Relation $\le$ auf Σ^* so definiert ist:
v, falls man u aus v durch *Löschen* einiger Buchstaben ält (jedoch *ohne* die Reihenfolge der übrigen ɔhstaben zu ändern)
mit Ausgabe von `diff`

Die Einbettungs-Relation

: $u \le v$, falls u aus v durch Löschen von Buchstaben

ɪt Halbordnung (transitiv, reflexiv, antisymmetrisch),

ɪt keine totale Ordnung

ɪtfragen:

ıegeben v. Für wieviele u gilt $u \le v$?

ffizienter Algorithmus für: Eingabe u, v, Ausgabe $u \le v$
3oolean)

Die Einbettungs-Relation (II)

ȷriffe (für Halbordnungen):

ette: Menge von paarweise vergleichbaren Elementen

ntikette: Menge von paarweise unvergleichbaren lementen

ze: für $\le$ ist

ːde Kette endlich

ːde Antikette endlich

spiel: bestimme die Menge der $\le$-minimalen Elemente

…

Die Einbettungs-Relation (III)

Endlichkeit von Ketten und Antiketten bezüglich bettung gilt für

ısten

äume (Satz von Kruskal, 1960)

ıraphen (Satz von Robertson/Seymour)

3eweis über insgesamt 500 Seiten über 20 Jahre, bis a. 2000)

ȷl. Kapitel 12 in: Reinhard Diestel: Graph Theory,
`ttp://www.math.uni-hamburg.de/home/`

Aufgaben (autotool) zu LCS

CS-Beispiel (das Beispiel aus Vorlesung)

CS-Quiz (gewürfelt - Pflicht!)

CS-Long (Highscore - Kür)

LCS — naiver Algorithmus (exponentiell)

```
 | Länge einer längsten gemeinsamen Teilfol
s :: Eq a => [a] -> [a] -> Int
s [] ys = 0
s xs [] = 0
s (x : xs) (y : ys) =
naximum
  [ if x == y then 1 + lcs xs ys
                else     lcs xs ys
  , lcs      xs  (y : ys)
  , lcs (x : xs)      ys
  ]
```

-down: sehr viele rekursive Aufrufe ...

r nicht viele *verschiedene* ...

imierung durch bottom-up-Reihenfolge!

LCS — bottom-up (quadratisch) + Übung

```
ss LCS {

  // bestimmt größte Länge einer gemeinsamen Teilfol
  static int lcs (char [] xs, char [] ys) {
      int a[][] = new int [xs.length][ys.length];
      for (int i=0; i<xs.length; i++) {
          for (int j=0; j<ys.length; j++) {
              // Ziel:
              // a[i][j] enthält größte Länge einer
              // von xs[0 .. i] und ys[0 ..j]
          }
      }
      return get (a, xs.length-1, ys.length-1);
  }

 // liefert Wert aus Array oder 0, falls Indizes zu k
 static int get (int [][] a, int i, int j) {
     if ((i < 0) || (j <0)) {
         return 0;
     } else {
         return a[i][j];
     }
 }

 public static void main(String[] args) {
     String xs = "ABCABBA";
     String ys = "CBABAC";
     System.out.println (lcs (xs.toCharArray(), ys.to(
 }
```

gaben:

ervollständigen Sie die Methode `LCS.lcs`

auen Sie eine Möglichkeit ein, nicht nur die Länge einer ıngsten gemeinsamen Teilfolge zu bestimmen, sondern uch eine solche Folge selbst auszugeben.

inweis: `int [][] a` wie oben ausrechnen und *danach* om Ende zum Anfang durchlaufen (ohne groß zu uchen).

amit dann die autotool-Aufgaben lösen.

LCS – eingeschränkt linear

che nach einer LCS = Suchen eines kurzen Pfades von 0) nach `(xs.length-1, ys.length-1)`.

zelne Kanten verlaufen

ach rechts: $(i-1,j) \to (i,j)$ Buchstabe aus `xs`

ach unten: $(i,j-1) \to (i,j)$ Buchstabe aus `ys`

ach rechts unten (diagonal): $(i-1,j-1) \to (i,j)$

emeinsamer Buchstabe

imierungen:

uche nur in der Nähe der Diagonalen

eginne Suche von beiden Endpunkten

nn nur $\leq D$ Abweichungen vorkommen, dann genügt einen Bereich der Größe $D \cdot N$ zu betrachten $\Rightarrow$ *An* $ND)$ *Difference Algorithm and its Variations*.

diff und LCS

diff werden nicht einzelne *Zeichen* verglichen,
ıdern ganze *Zeilen*.
; gestattet/erfordert Optimierungen:

eilen feststellen, die nur in einer der beiden Dateien
ɔrkommen, und entfernen

`iff/analyze.c:discard_confusing_lines ()`

um Vergleich der Zeilen Hash-Codes benutzen

`iff/io.c:find_and_hash_each_line ()`

ıe Quellen `http://cvs.sourceforge.net/`
`ewcvs.py/cvsgui/cvsgui/cvs-1.10/diff/`
gabe: wo sind die Quellen für die CVS-Interaktion in

ipse?

Versionierung (I)

von Quelltexten

VS: jede Datei einzeln

VN: gesamtes Repository

arcs, git: Mengen von Patches

; ist für die *Entwickler* ganz nützlich,
ər für die *Kunden* nicht!

Versionierung (II)

pfohlenes Schema:

ersion = Liste von drei Zahlen $[x, y, z]$

rdung: lexikographisch.

derungen bedeuten:

(major): inkompatible Version

(minor): kompatible Erweiterung

(patch): nur Fehlerkorrektur

ıderformen:

gerade: stabil, y ungerade: Entwicklung

Datum

Produktqualität (analytisch)

Klassifikation der Verfahren

erifizieren (= Korrektheit beweisen)
Verifizieren
symbolisches Ausführen

esten (= Fehler erkennen)
statisch (z. B. Inspektion)
dynamisch (Programm-Ausführung)

nalysieren (= Eigenschaften vermessen/darstellen)
Quelltextzeilen (gesamt, pro Methode, pro Klasse)
Klassen (Anzahl, Kopplung)
Profiling (. . . später mehr dazu)

Fehlermeldungen

en enthalten

ystemvoraussetzungen

rbeitsschritte

eobachtetes Verhalten

rwartetes Verhalten

waltung z. B. mit Bugzilla, Trac
. Seminarvortrag D. Ehricht:
`:p://www.imn.htwk-leipzig.de/~waldmann/`
`ı/ss04/se/ehricht/bugzilla.pdf`

Testen und Schnittstellen

est für Gesamtsystem (schließlich) oder Teile (vorher)

eile definiert durch Schnittstellen

chnittstelle ⇒ Spezifikation

pezifikation ⇒ Testfälle

ten . . .

nterhalb einer Schnittstelle (unit test)

berhalb (mock objects) (vgl. dependency injection)

gl. `http://www.mockobjects.com/`

Dynamische Tests

estfall: Satz von Testdaten

esttreiber zur Ablaufsteuerung

gf. *instrumentiertes* Programm zur Protokollierung

spiele (f. Instrumetierung):

ebugger: fügt Informationen über Zeilennummern in
bjektcode ein

```
cc -g foo.c -o foo  ;  gdb foo
```

rofiler: Code-Ausführung wird regelmäßig unterbrochen

Dynamische Tests: Black/White

trukturtests (white box)

programmablauf-orientiert
datenfluß-orientiert

unktionale Tests (black box)

lischformen (unit test)

Black-Box-Tests

ıe Programmstruktur zu berücksichtigen.

ʼpische Eingaben (Normalbetrieb)
lle wesentlichen (Anwendungs-)Fälle abdecken
3sp: gerade und ungerade Länge einer Liste bei reverse)
xtreme Eingaben
ehr große, sehr kleine, fehlerhafte
ufällige Eingaben
urch geeigneten Generator erzeugt

ırend Produktentwicklung:
tmenge ständig erweitern,
ıere Tests immer wiederholen (regression testing)

Probleme mit GUI-Tests

wierig sind Tests, die sich nicht automatisieren lassen
JIs: Eingaben mit Maus, Ausgaben als Grafik)
Unterstützung sollte jede Komponente neben der
I-Schnittstelle bieten:

uch eine API-Schnittstelle (für (Test)programme)

nd ein Text-Interface (Kommando-Interpreter)

): Emacs: `M-x kill-rectangle` oder `C-x R K`, usw.

Mischformen

estfälle für jedes Teilprodukt, z. B. jede Methode

l. h. Teile der Programmstruktur werden berücksichtigt)

urchführung kann automatisiert werden (`JUnit`)

Testen mit JUnit

ıt Beck and Erich Gamma,
`:p://junit.org/index.htm`

```
ɔort static org.junit.Assert.*;
ass XTest {

 @Test
  public void testGetFoo() {
      Top f = new Top ();
      assertEquals (1, f.getFoo());
  }
```

`:p://www-128.ibm.com/developerworks/java/`
`ɔrary/j-junit4.html`

ıit ist in Eclipse-IDE integriert (New → JUnit Test Case
4.0)

JUnit und Extreme Programming

ıt Beck empfiehlt *test driven apporach*:

rst alle Test-Methoden schreiben,

ann eigentliche Methoden implementieren

. bis sie die Tests bestehen (und nicht weiter!)

rodukt-Eigenschaften, die sich nicht testen lassen, *sind icht vorhanden*.

u jedem neuen Bugreport einen neuen Testfall anlegen

tfall schreiben ist *Spezifizieren*, das geht *immer* dem ɔlementieren voraus. — *Testen* der Implementierung ist die zweitbeste Lösung (besser ist *Verifizieren*).

Delta Debugging

dreas Zeller: *From automated Testing to Automated bugging*, automatische Konstruktion von

ıinimalen Bugreports

ehlerursachen (bei großen Patches)

dell:

`est : Set<Patch> -> { OK, FAIL, UNKNOWN }`

`d(low,high,n)=(x,y)`

Vorbedingung `low` ⊆ `high`,
`test(low)=OK, test(high)=FAIL`
Nachbedingung `x` ⊆ `y`,

Delta Debugging (II)

```
low, high, n) =
 let diff = size(high) - size(low)
     c_1, .. c_n = Partition von (high - low)
 if exists i : test (low + c_i) == FAIL
     then dd(                    )
 else if exists i : test (high - c_i) == OK
     then dd(                    )
 else if exists i : test (low + c_i) == OK
     then dd(                            )
 else if exists i : test (high - c_i) == FAIL
     then dd(                            )
 else if n < diff
     then dd(          ) else (low, high)
```

:p://www.infosun.fim.uni-passau.de/st/
ɔers/computer2000/

Programmablauf-Tests

:ieht sich auf Programm-Ablauf-Graphen (Knoten:
veisungen, Kanten: mögliche Übergänge)

nweisungs-Überdeckung: jede Anweisung mindestens
inmal ausgeführt
weigüberdeckung: jede Kante mindestens einmal
urchlaufen — Beachte: `if (X) then { A }`
fadüberdeckung: jeder Weg (Kantenfolge) mindestens
inmal durchlaufen — Beachte: Schleifen (haben viele
urchlaufwege)
ariante: jede Schleife (interior) höchstens einmal
edingungs-Überdeckung: jede atomare Bedingung
inmal true, einmal false.

Prüfen von Testabdeckungen

Werkzeugunterstützung, Bsp.: *Profiler*:
ɜt bei Ausführung Anzahl der Ausführungen …

. jeder Anweisung (Zeile!)

. jeder Verzweigung (then oder else)

nügt für welche Abdeckungen?)

ıfiling durch Instrumentieren (Anreichern)

es Quelltextes

der der virtuellen Maschine

Übung Profiling (C++)

spiel-Programm(e):
:p://www.imn.htwk-leipzig.de/~waldmann/
ı/ss04/case/programme/analyze/cee/
gaben:

ompilieren und ausführen für Profiling:

```
++ -pg -fprofile-arcs heap.cc -o heap
/heap > /dev/null
 welche Dateien wurden erzeugt? (ls -lrt)
prof heap # Analyse
```

ompilieren und ausführen für Überdeckungsmessung:

```
++ -ftest-coverage -fprofile-arcs heap.cc -
```

```
/heap > /dev/null
 welche Dateien wurden erzeugt? (ls -lrt)
cov heap.cc
 welche Dateien wurden erzeugt? (ls -lrt)
```

ɪptionen für `gcov` ausprobieren! (`-b`)

`eap` reparieren: `check` an geeigneten Stellen aufrufen,
m Fehler einzugrenzen

`edian3` analysieren: Testfälle schreiben (hinzufügen)
ır: Anweisungsüberdeckung, Bedingungsüberdeckung,
fadüberdeckung

berdeckungseigenschaften mit `gcov` prüfen

`edian5` reparieren

Profiling (Java)

ommandozeile: `java -Xprof ...`

ı Eclipse: benutzt TPTP http://www.eclipse.org/
rticles/Article-TPTP-Profiling-Tool/
ptpProfilingArticle.html http://www.
clipse.org/tptp/home/documents/tutorials/
rofilingtool/profilingexample_32.html

ıstallation: Eclipse → Help → Update …

n Pool vorbereitet, benötigt aber genau diese
clipse-Installation und java-1.5

```
xport PATH=/home/waldmann/built/bin:$PATH
nset LD_LIBRARY_PATH
home/waldmann/built/eclipse-3.2.2/eclipse &
```

ür JDK-1.6: TPTP-4.4 in Eclipse-3.3 (Europa))

Code-Optimierungen

ıy Hoare first said,
ɪ Donald Knuth famously repeated,
ımature optimization is the root of all evil.

rste Regel für Code-Optimierung: *don't do it …*

weite Regel: *… yet!*

t korrekten Code schreiben, *dann*
ɜsourcenverbrauch messen (profiling),
ın eventuell kritische Stellen verbessern.

ɜser ist natürlich: kritische Stellen vermeiden.
liotheksfunktionen benutzen!

Kosten von Algorithmen schätzen

-Oh-Notation zum Vergleich des Wachstums von ıktionen kennen und anwenden

infache Schleife

eschachtelte Schleifen

inäres Teilen

ɔinäres) Teilen und Zusammenfügen

ombinatorische Explosion

ɜse Liste aus Pragmatic Programmer, p. 180)

asymptotischen Laufzeiten lassen sich durch lokale timierungen *nicht* ändern, also: vorher nachdenken ıt sich

Code-Transformationen zur Optimierung

n Bentley: Programming Pearls, ACM Press, 1985, 1999)

eit sparen auf Kosten des Platzes:

Datenstrukturen anreichern (Komponenten hinzufügen)
Zwischenergebnisse speichern
Cache für häufig benutzte Objekte

latz sparen auf Kosten der Zeit:

Daten packen
Sprache/Interpreter (Bsp: Vektorgrafik statt Pixel)

chleifen-Akrobatik, Unterprogramme auflösen usw. berlassen wir mal lieber dem Compiler/der (virtuellen) laschine

Gefährliche „Optimierungen“

fahr besteht immer, wenn die Programm-Struktur anders die Denk-Struktur ist.

nwendungsspezifische Datentypen vermieden bzw. usgepackt → primitive obsession

ndikator: String und int)

xistierende Frameworks ignoriert

ndikatoren: kein import java.util.*; sort selbst eschrieben, XML-Dokument als String)

nterprogramm vermieden bzw. aufgelöst → zu lange lethode (bei 5 Zeilen ist Schluß)

äter ausführlicher bei *code smells* → Refactoring)

Code-Metriken

lche Code-Eigenschaften kann man messen? Was ıen sie aus?

nzahl der Methoden pro Klasse

nzahl der . . . pro . . .

ɛxtuelle Komplexität: Halstaed

trukturelle Komplexität: McCabe

ɔO-Klassenbeziehungen

Code-Metriken: Halstaed

ert nach Balzert, Softwaretechnik II)

Anzahl aller Operatoren/Operationen (Aktionen)

Anzahl unterschiedlicher Operatoren/Operationen

Anzahl aller Operanden/Argumente (Daten)

Anzahl unterschiedlicher Operanden/Argumente

$+ a$ Größe des Vokabulars, $O + A$ Größe der nplementierung)

grammkomplexität: $\frac{o \cdot A}{2 \cdot a}$

Code-Metriken: McCabe

ert nach Balzert, Softwaretechnik II)

lomatische Zahl (des Ablaufgraphen $G = (V, E)$)
$- |V| + 2c$ wobei $c =$ Anzahl der
ɜammenhangskomponenten
ispiele)
e: durch Hinzufügen einer Schleife, Verzweigung usw. gt dieser Wert um eins.

OO-Metriken

ttribute bzw. Methoden pro Klasse

iefe und Breite der Vererbungshierarchie

opplung (zwischen Klassen) wieviele andere Klassen nd in einer Klasse bekannt? (je weniger, desto besser)

ohäsion (innerhalb einer Klasse): hängen die Methoden ng zusammen? (je enger, desto besser)

Kohäsion: Chidamber und Kemerer

zahl der Paare von Methoden, die kein gemeinsames ibut benutzen) $-$ (Anzahl der Paare von Methoden, die gemeinsames Attribut benutzen)
ɛichnet fehlende Kohäsion, d. h. kleinere Werte sind ɪser.

Kohäsion: Henderson-Sellers

M Menge der Methoden

. Menge der Attribute

ır $a \in A$: $Z(a)$ = Menge der Methoden, die a benutzen

Mittelwert von $|Z(a)|$ über $a \in A$

lende Kohäsion: $\frac{|M|-z}{|M|-1}$
ınere Werte sind besser)

Code-Metriken (Eclipse)

ipse Code Metrics Plugin installieren und für eigenes
jekt anwenden.

`ttp://eclipse-metrics.sourceforge.net/`

ıstallieren in Eclipse: Help → Software Update → Find
→ Search for New → New (Remote/Local) site

rojekt → Properties → Metrics → Enable, dann Projekt
→ Build, dann anschauen

Refactoring

Herkunft

ıt Beck: *Extreme Programming*, Addison-Wesley 2000:

aar-Programmierung (zwei Leute, ein Rechner)

est driven: erst Test schreiben, dann Programm
nplementieren

esign nicht fixiert, sondern flexibel

Refaktoring: Definition

rtin Fowler: *Refactoring: Improving the Design of
sting Code*, A.-W. 1999,
`:p://www.refactoring.com/`
: Software so ändern, daß sich

xternes Verhalten nicht ändert,

ıterne Struktur verbessert.

ıe auch William C. Wake: *Refactoring Workbook*, A.-W.
)4 `http://www.xp123.com/rwb/`
l Stefan Buchholz: Refactoring (Seminarvortrag)
`:p://www.imn.htwk-leipzig.de/~waldmann/`
`ı/current/se/talk/sbuchhol/`

Refactoring anwenden

ıancher Code „riecht"(schlecht)

Liste von *smells*)

r (oder anderer) muß geändert werden

Liste von *refactorings*, Werkzeugunterstützung)

nderungen (vorher!) durch Tests absichern

JUnit)

Refaktorisierungen

ntwurfsänderungen …

erwende Entwurfsmuster!

kleine" Änderungen

Abstraktionen ausdrücken:
neue Schnittstelle, Klasse, Methode, (temp.) Variable
Attribut bewegen, Methode bewegen (in andere Klasse)

Code Smell # 1: Duplicated Code

e Idee sollte an *genau einer* Stelle im Code formuliert
rden:
de wird dadurch

ichter verständlich

ichter änderbar

doppelter Quelltext (copy–paste) führt immer zu
rtungsproblemen.

Duplicated Code → Schablonen

olizierter Code wird verhindert/entfernt durch

chablonen (beschreiben das Gemeinsame)

ıit *Parametern* (beschreiben die Unterschiede).

spiel dafür:

nterprogramm (Parameter: Daten, Resultat: Programm)

olymorphe Klasse (Parameter: Typen, Resultat: Typ)

nterprogramm höherer Ordnung (Parameter:
rogramm, Resultat: Programm)

ın Programme als Parameter nicht erlaubt sind (Java),
ın werden sie als Methoden von Objekten versteckt (vgl.

Size does matter

tere Code smells:

ınge Methode

roße Klasse

ınge Parameterliste

verursacht durch anderen Smell: Primitive Obession

Primitive Daten (*primitive obsession*)

nptome: Benutzung von `int, float, String...`
ıachen:

ıhlende Klasse:

. B. String → FilePath, Email, ...

chlecht implementiertes Fliegengewicht

. B. `int i` bedeutet `x[i]`

mulierter Attributname:

. B. `Map<String,String> m; m.get("foo");`

ıebung: Klassen benutzen, Array durch Objekt ersetzen
B. `class M { String foo; ...}`)

Typsichere Aufzählungen

inition (einfach)

```
olic enum Figur { Bauer, Turm, König }
```

inition mit Attribut (aus JLS)

```
olic enum Coin {
  PENNY(1), NICKEL(5), DIME(10), QUARTER(25);
  Coin(int value) { this.value = value; }
  private final int value;
  public int value() { return value; }
```

inition mit Methode:

```
olic enum Figur {
Bauer { int wert () { return    1; } },
Turm  { int wert () { return    5; } },
```

```
König { int wert () { return 1000; } };
abstract int wert ();
```

ıutzung:

```
gur f = Figur.Bauer;
gur g = Figur.valueOf("Turm");
 (Figur h : Figur.values()) {
 System.out.println (h + ":" + h.wert());
```

Verwendung von Daten: Datenklumpen

ıler: Klumpen von Daten wird immer gemeinsam benutzt

```
ring infile_base; String infile_ext;
ring outfile_base; String outfile_ext;

atic boolean is_writable (String base, String
```

ikator: ähnliche, schematische Attributnamen
ung: Klasse definieren

```
ass File { String base; String extension; }

atic boolean is_writable (File f);
```

Datenklumpen—Beispiel

spiel für Datenklumpen und -Vermeidung:

```
va.awt

ctangle(int x, int y, int width, int height
ctangle(Point p, Dimension d)
```

gleichen Sie die Lesbarkeit/Sicherheit von:

```
v Rectangle (20, 40, 50, 10);
v Rectangle ( new Point (20, 40)
            , new Dimension (50, 10) );
```

gleichen Sie:

```
va.awt.Graphics: drawRectangle(int,int,int,
va.awt.Graphics2D: draw (Shape);
  class Rectangle implements Shape;
```

Verwendung von Daten: Data Class

ıler:
sse mit Attributen, aber ohne Methoden.

```
ass File { String base; String ext; }
```

ung:
le typische Verwendung der Attribute in Client-Klassen,
p: `f.base + "/" + f.ext`)
reibe entsprechende Methode, verstecke Attribute (und
en Setter/Getter)

```
ass File {  ...
  String toString () { ... }
```

Aufgabe Refactoring

rfelspiel-Simulation:
ıummelmex: zwei (mehrere) Spieler, ein Würfelbecher
elzug ist: aufdecken oder (verdeckt würfeln, ansehen,
agen, weitergeben) bei Aufdecken wird vorige Ansage
 vorigem Wurf verglichen, das ergibt Verlustpunkt für
ı Aufdecker oder den Aufgedeckten

or Refactoring:
`ttp://www.imn.htwk-leipzig.de/~waldmann/`
`du/ss07/case/programs/refactor/stage0/`
/elche Code-Smells?

ach erstem Refactoring:

du/ss07/case/programs/refactor/stage1/
/as wurde verbessert? Welche Smells verbleiben?

ach zweitem Refactoring:
ttp://www.imn.htwk-leipzig.de/~waldmann/
du/ss07/case/programs/refactor/stage2/
/as wurde verbessert? Welche Smells verbleiben?

Temporäre Attribute

nptom: viele `if (null == foo)`
ache: Attribut hat nur während bestimmter
grammteile einen sinnvollen Wert
ilfe: das ist kein Attribut, sondern eine temporäre
iable.

Nichtssagende Namen

ame drückt Absicht nicht aus)
nptome:

esteht aus nur einem oder zwei Zeichen

nthält keine Vokale

umerierte Namen (`panel1, panel2, \dots`)

nübliche Abkürzungen

reführende Namen

ebung: umbenennen, so daß Absicht deutlicher wird.
zu muß diese dem Programmierer selbst klar sein!)
rkzeugunterstützung!

Name enthält Typ

nptome:

lethodenname bezeichnet Typ des Arguments oder
esultats

```
lass Library { addBook( Book b ); }
```

ttribut- oder Variablenname bezeichnet Typ (sog.
ngarische Notation) z. B. `char ** ppcFoo`

ehe
ttp://ootips.org/hungarian-notation.html

rundsätzlich) Name bezeichnet Implementierung statt
edeutung

ebung: umbenennen (wie vorige Folie)

Programmtext

ommentare

→ *don't comment bad code, rewrite it*

omplizierte Boolesche Ausdrücke

→ umschreiben mit Verzweigungen, sinnvoll
ezeichneten Hilfsvariablen

onstanten (*magic numbers*)

→ Namen für Konstanten, Zeichenketten externalisieren
18N)

Größe und Komplexität

lethode enthält zuviele Anweisungen (Zeilen)

lasse enthält zuviele Attribute

lasse enthält zuviele Methoden

gabe: welche Refaktorisierungen?

Mehrfachverzweigungen

nptom: `switch` wird verwendet

```
ass C {
  int tag; int FOO = 0;
  void foo () {
      switch (this.tag) {
          case FOO: { .. }
          case 3:   { .. }
  }   }
```

ache: Objekte der Klasse sind nicht ähnlich genug
ilfe: Kompositum-Muster

```
:erface C { void foo (); }
ass Foo implements C { void foo () { .. } }
```

null-Objekte

nptom: `null` (in Java) bzw. 0 (in C++) bezeichnet ein
sonderes Objekt einer Klasse, z. B. den leeren Baum
er die leere Zeichenkette

ache: man wollte Platz sparen oder „Kompositum“
meiden.

chteil: `null` bzw. `*0` haben keine Methoden.

hilfe: ein extra Null-Objekt deklarieren, das wirklich zu
Klasse gehört.

Richtig refaktorisieren

nmer erst die Tests schreiben

ode kritisch lesen (eigenen, fremden), eine Nase für
nrüchigkeiten entwickeln (und für perfekten Code).

de Faktorisierung hat ein Inverses.

neue Methode deklarieren ↔ Methode inline
xpandieren)

ntscheiden, welche Richtung stimmt!

Verkzeug-Unterstützung erlernen

Aufgaben zu Refaktoring (I)

ode Smell Cheat Sheet (Joshua Keriewsky):
`ttp://industriallogic.com/papers/`
`mellstorefactorings.pdf`

mell-Beispiele `http://www.imn.htwk-leipzig.`
`e/~waldmann/edu/ss05/case/rwb/` (aus
efactoring Workbook von William C. Wake
`ttp://www.xp123.com/rwb/`)

h6-properties, ch6-template, ch14-ttt

Aufgaben zu Refaktoring (II)

actoring-Unterstützung in Eclipse:

```
ckage simple;

olic class Cube {
  static void main (String [] argv) {
      System.out.println (3.0 + " " + 6 * 3
      System.out.println (5.5 + " " + 6 * 5
  }
```

ract local variable, extract method, add parameter, . . .

Aufgaben zu Refaktoring (II)

clipse → Refactor → Extract Interface

Create Factory”

inde Beispiel für “Use Supertype”

Class Design

Klassen-Entwurf

enutze Klassen! (sonst: primitive obsession)

rdne Attribute und Methoden richtig zu
Refactoring: move method, usw.)

okumentiere Invarianten für Objekte, Kontrakte für
Methoden

telle Beziehungen zwischen Klassen durch Interfaces
ar

. . Entwurfsmuster)

Immutability

shua Bloch: Effective Java, Addison Wesley, 2001)

nutable = unveränderlich

spiele: String, Integer, BigInteger

eine Set-Methoden

eine überschreibbaren Methoden

lle Attribute privat

lle Attribute final

hter zu entwerfen, zu implementieren, zu benutzen.

Immutability

nmutable Objekte können mehrfach benutzt werden
sharing).

statt Konstruktor: statische Fabrikmethode. Suche
eispiele in Java-Bibliothek)

uch die Attribute der immutable Objekte können
achgenutzt werden (keine Kopie nötig)

Beispiel: negate für BigInteger)

nmutable Objekte sind sehr gute Attribute anderer
Objekte:

eil sie sich nicht ändern, kann man die Invariante des

Vererbung bricht Kapselung

plementierungs-Vererbung: bad,
ınittstellen-Vererbung: good.)

ıblem: `class B extends A` $\Rightarrow$
ängt ab von Implementations-Details von A.

Venn man nur A ändert, kann B kaputtgehen.

ispiel)

Vererbung bricht Kapselung

hua Bloch (Effective Java):

esign and document for in heritance

. or else prohibit it

l-Beschreibung muß Teile der Implementierung
umentieren (welche Methoden rufen sich gegenseitig
), damit man diese sicher überschreiben kann.
ıs ist ganz furchtbar.)

tt Vererbung: benutze Komposition (Wrapper) und dann
egation.