Einleitung

Programme und Softwaresys

(Charakterisierung nach Brooks)

- *Programm:* in sich vollständig, kann von se dem System benutzt werden, auf dem es e wurde
- *Komponente eines Systems:* Schnittstellen
- *marktfähiges Produkt:* Generalisierung, Tes Dokumentation, Wartung

Produktivität

Ein Programmierer schafft etwa

10 (in Worten: zehn)

(vollständig getestete und dokumentierte) Programmzeilen pro Arbeitstag.
(d. h. ≤ 2000 Zeilen pro Jahr)

Dieses Maß hat sich seit 30 Jahren nicht geä
($\Rightarrow$ Produktivitätssteigerung nur durch höhere Programmiersprachen möglich)

Material

- Balzert: Lehrbuch der Software-Technik, Ba Spektrum Akad. Verlag, 2000
- Andrew Hunt und David Thomas: The Prag Programmer, Addison-Wesley, 2000
- Gamma, Helm, Johnson, Vlissides: Entwur Addison-Wesley, 1996
- Martin Fowler: Refactoring, ...

Linux-Pool des Fachschaftsrates
`http://fachschaft.imn.htwk-le`

Softwaretechnik II
Vorlesung
Sommersemester 200

Johannes Waldmann, HTWK Leip

18. Juni 2008

Software ist schwer zu entwi

- ist immaterielles Produkt
- unterliegt keinem Verschleiß
- nicht durch physikalische Gesetze begrenz
- leichter und schneller änderbar als ein tech Produkt
- hat keine Ersatzteile
- altert
- ist schwer zu vermessen

(Balzert, Band 1, S. 26 ff)

Inhalt

- Programmieren im Kleinen, Werkzeuge (Ec
- Programmieren im Team, Werkzeuge (CVS Trac)
- Spezifizieren, Verifizieren, Testen
- Entwurfsmuster
- Refactoring

Organisation

- Vorlesung:
  - dienstags (u), 15:30–17:00, Li110
  - *und* dienstags (g), 11:15–12:45, Li41
- Übungen (Z424):
  - Mi (u + g) 7:30–9:00
  - *oder* Mi (u + g) 9:30–11:00
  - *oder* Do (u) 19:00–20:30 und Fr (g) 7
- Übungsgruppen wählen: `https://au` `htwk-leipzig.de/cgi-bin/Super`
- (für CVS/Bugzilla) jeder benötigt einen

The Pragmatic Programm

(Andrew Hunt and David Thomas)

1. Care about your craft.
2. Think about your work.
3. Verantwortung übernehmen. Keine faulen A cat ate my source code . . .) sondern Altern anbieten.
4. Reparaturen nicht aufschieben. (Software-E
5. Veränderungen anregen. An das Ziel denke
6. Qualitätsziele festlegen und prüfen.

Regelmäßig investieren

(Pragmatic Programmer)

- jedes Jahr wenigstens eine neue Sprache l
- jedes Quartal ein Fachbuch lesen
- auch Nicht-Fachbücher lesen
- Weiterbildungskurse belegen
- lokale Nutzergruppen besuchen (Bsp: `http://gaos.org/lug-l/`)
- verschiedene Umgebungen und Werkzeug ausprobieren
- aktuell bleiben (Zeitschriften abonnieren, N lesen)
- kommunizieren

Edsger W. Dijkstra über Softwar

- Dijkstra-Archiv `http://www.cs.utexas.edu/users/E`
- Thesen zur Softwaretechnik `http://www. edu/users/EWD/ewd13xx/EWD1305.PD`

Schnittstellen

Beispiele

Beschreiben Sie Interfaces (Schnittstelle Leben:

- Batterien
- CD/DVD (Spieler/Brenner, Rohlinge,. . .
- Auto(-Vermietung)
- . . .

Schnittstellen und -Vererbung in der Math

- Halbgruppe, Monoid, Gruppe, Ring, Kö

Leistungen:

- Prüfungsvoraussetzung: regelmäßiges und Bearbeiten von Übungsaufgaben

  ggf. in Gruppen (wie im Softwarepraktikum)
- Prüfung: Klausur

Lernen! Lernen! Lernen

(Pragmatic Programmer)

Das eigene *Wissens-Portfolio* pflegen:

- regelmäßig investieren
- diversifizieren
- Risiken beachten
- billig einkaufen, teuer verkaufen
- Portfolio regelmäßig überprüfen

Übung KW 11

Fred Brooks: The Mythical Man

- Suchen Sie (google) Rezensionen zu diese
- Was ist *Brooks' Gesetz*? (“Adding . . . ”)
- Was sagt Brooks über Prototypen? (“Plan t
- Welche anderen wichtigen Bücher zur Soft werden empfohlen?

Was macht diese Funk

```
public static int f (int x, int
 if (x <= y) {
  return z;
 } else {
  return
   f (f (x-1,y,z), f(y-1,z,x),
 }
}
```

- wieviele rekursive Aufrufe finden statt?
- kann man das Ergebnis vorhersagen, o rekursiven Aufrufe durchzuführen?

- in A.main: `B x = new B();` , Fehler → C
  Klasse B anlegen
- in A.main: `x.p();`, Fehler → Control-1, Me
  anlegen
- in B: Refactor → extract interface.

Schnittstellen und Verträg

wenn jemand eine Schnittstelle implementier
schreibt er nicht irgendwelche Methoden mit
Namen, sondern erfüllt einen Vertrag:

- Implementierung soll genau das tun,
  was beschrieben wird.
- Implementierung soll nichts
  anderes, unsinniges, teures, gefährliches tu
- Implementierung soll bescheid geben,
  wenn Auftrag nicht ausführbar ist.

(Bsp: Pizzafehlermeldung)

Schnittstellen und Tests

man überzeuge sich von

- Benutzbarkeit einer Schnittstelle (unabhäng
  Implementierung)

  . . . wird das gewünschte Ergebnis durch eir
  Methodenaufrufen vertraglich garantiert?
- Korrekheit einer Implementierung

mögliche Argumentation:

- formal (Beweis)
- testend (beispielhaft)

  . . . benutze *Proxy*, der Vor/Nachbedingunge

Typen als Verträge

Der Typ eines Bezeichners ist seine best
(denn der Compiler kann sie prüfen!)
Es sind Sprachen (und ihre Sprecher) arr
Typsystem ausdrucksschwach ist.

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

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

// korrektes Java:
List<String> l = new LinkedLis
```


- Halbordnung, (totale) Ordnung

  vgl. Beschreibung von `Comparable<E>`

Schnittstellen zwischen (Teilen von) Software

- wo sind die Schnittstellen, was wird transpo
  (Beispiele)
- wie wird das (gewünschte) Verhalten spezif
  sicher kann man sein, daß die Spezifikatior

Schnittstellen (interfaces) in Java, Beispiel in

- Eclipse (Window → Preference → Compile
  Compliance 6.0)
- Klasse A mit Methode main

Literatur zu Schnittstelle

Ken Pugh: *Interface Oriented Design*, 2006. I
0-0766940-5-0. `http://www.pragmaticp`
`com/titles/kpiod/index.html`
enthält Beispiele:

- Pizza bestellen
- Unix devices, file descriptors
- textuelle Schnittstellen
- grafische Schnittstellen

Design by Contract

Betrand Meyer: *Object-Oriented Software Co*
Prentice Hall, 1997.
`http://archive.eiffel.com/doc/oos`
Aspekte eines Vertrages:

- Vorbedingungen
- Nachbedingungen
- Klassen-Invarianten

Stufen von Verträge

(nach K. Pugh)

- Typdeklarationen
- Formale Spezifikation von Vor- und Nac
- Leistungsgarantien (für Echtzeitsystem
- Dienstgüte-Garantien (quality of service

Schnittstellen zum Datentran

Adressierung:

- wahlfreier Zugriff (Festplatte)
- sequentieller Zugriff (Magnetband)

Transportmodus:

- Pull (bsp. Web-Browser)
- Push (bsp. Email)

Bsp: SAX und DOM einordnen

Mehrfache Schnittsteller

Eine Klasse kann mehrere Schnittstellen imp
(deren Verträge erfüllen).
Dann aber Vorsicht bei der Bezeichnung der
. . . und beim Verwechseln von Zuständen (Bs

Klassen-Entwurf

- `Zahl` hat ein `private final` Attribut,
  wird im Konstruktor gesetzt
- `Zahl` implementiert `String toString()`
  dann funktioniert

```
System.out.println(Arrays.asList
```


Protokollierung mit Dek

Aufgabe:

- alle Aufrufe von `Zahl.compareTo` pro
- *ohne* den Quelltext dieser Klasse zu än

Lösung: eine Klasse dazwischenschieber

```
class Log<E> ... {
    private final contents E;
    int compareTo(Log<E> that)
}
Log<Zahl> a [] =
    { new Log<Zahl> (new Zahl
```

Diese Klasse heißt *Dekorator*, das ist ein
Entwurfsmuster.

Arten von Schnittstellen

Was wird verwaltet?

- Schnittstellen für Daten
  (Methoden lesen/schreiben Attribute)
- Schnittstellen für Dienste
  (Methoden „arbeiten wirklich“)

Schnittstellen und Zustän

- Schnittstelle *ohne* Zustand
  - Vorteil: Aufrufreihenfolge beliebig
  - Nachteil: mehr Parameter (einer?)
- Schnittstelle *mit* Zustand
  - Nachteil: Reihenfolge wichtig
  - Vorteil: weniger Parameter

Beispiel: Sortieren

wesentliche Bestandteile

```
public class Zahl
   implements Comparable<Zahl> {
      public int compareTo(Zahl t
}

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


Richtig vergleiche

das sieht clever aus, ist aber nicht allgem

```
   public int compareTo(Zahl t
        return this.contents -
    }
```

(merke: *avoid clever code*)

Beispiel zu Entwurfsmuste

aus: Gamma, Helm, Johnson, Vlissides: *Entw*
Beispiel: ein grafischer Editor.
Aspekte sind unter anderem:

- Dokumentstruktur
- Formatierung
- Benutzungsschnittstelle

Beispiel: Verhaltensmuster: St

Formatieren (Anordnen) der darstellbaren Ob
möglicherweise linksbündig, rechtsbündig, ze

Beispiel: Erzeugungsmuster: (al Fabrik

Anwendung soll verschiedene GUI-Look-and
ermöglichen

OO-Entwurfsmuste

Jedes Muster beschreibt eine in unserer
wiederkehrende Aufgabe und erläutert de
Lösung. Wir können die Lösung beliebig
aber niemals identisch wiederholen.
Ausgangspunkt/Beispiel: Model/View/Co
Benutzerschnittstellen in Smalltalk-80)

- Aktualisierung von entkoppelten Objekt
- hierarchische Zusammensetzung von V *Komposition*
- Beziehung View–Controller: *Strategie*

Entwurfsmuster (Überbl

Entwurfsmuster

Erich Gamma, Richard Helm, Ralph Johnson
Vlissides: *Entwurfsmuster (design patterns)* -
wiederverwendbarer objektorientierter Softwa
Addison-Wesley 1996.
liefert Muster (Vorlagen) für Gestaltung von E
zwischen (mehreren) Klassen und Objekten,
wiederverwendbarer, flexibler Software bewä

Seminarvorträge (voriges Jahr)
`http://www.imn.htwk-leipzig.de/~w`
`edu/ss05/case/seminar/`

Beispiel: Strukturmuster: Komp

darstellbare Elemente sind:
Buchstabe, Leerzeichen, Bild, Zeile, Spalte, S
Gemeinsamkeiten?
Unterschiede?

Beispiel: Strukturmuster: Dek

darstellbare Objekte sollen optional eine Umr
eine Scrollbar erhalten

Beispiel: Verhaltensmuste

Menü-Einträge, Undo-Möglichkeit

siehe auch Ereignisbehandlung in Applet

Wie Entwurfsmuster Probleme

- Finden passender Objekte
  insbesondere: nicht offensichtliche Abstrak
- Bestimmen der Objektgranularität
- Spezifizieren von Objektschnittstellen und Objektimplementierungen
  unterscheide zwischen *Klasse* (konkreter T (abstrakter Typ).
  programmiere auf eine Schnittstelle hin, nic Implementierung!
- Wiederverwendungsmechanismen anwend ziehe Objektkomposition der Klassenvererb
- Unterscheide zw. Übersetzungs- und Laufz

gewisse Aufgaben . . .

17. . . . verschiedene Sprachen zu einer größ verbinden: der gemeinsamen Sprache einer

27. In Wirklichkeit hat das Zeitlose nichts mit tun. Die Sprache beschreibt nur die natürlich Dinge. Sie lehrt nicht, sie erinnert uns nur an schon wissen und immer wieder neu entdeck

Strategie

Beispiel:

```
interface Comparator<T> { int com
class C implements Comparator<Int
List<Integer> l = ...;
Collections.sort (l, new C ());
```

Alternative: anonyme innerer Klasse

PS: daran sieht man: Entwurfsmuster sind de objektorientierte Ersatz fÃ¼r funktionale Prog (Unterprogramme als Daten, d. h. als Argum Resultate von Unterprogrammen)

Erzeugungsmuste

Bsp: fexible, konfigurierbare GUIs

- (Konstruktor)
- Fabrikmethode
- abstrakte Fabrik
- Singleton (Einermenge)
- Prototyp

Musterkatalog

- Erzeugungsmuster:
  - klassenbasiert: Fabrikmethode
  - objektbasiert: abstrakte Fabrik, Erbauer, F Singleton
- Strukturmuster:
  - klassenbasiert: Adapter
  - objektbasiert: Adapter, Brücke, Dekoriere Fliegengewicht, Kompositum, Proxy
- Verhaltensmuster:
  - klassenbasiert: Interpreter, Schablonenm
  - objektbasiert: Befehl, Beobachter, Besuch Memento, Strategie, Vermittler, Zustand, Zuständigkeitskette

Vorlage: Muster in der Archit

Christopher Alexander: *The Timeless Way of Pattern Language*, Oxford Univ. Press 1979.

10. Menschen formen Gebäude und benutze Muster-Sprachen. Wer eine solche Sprache damit unendlich viele verschiedene einzigarti herstellen, genauso, wie man unendlich viele Sätze in der gewöhnlichen Sprache bilden ka

14. . . . müssen wir tiefe Muster entdecken, di erzeugen können.

15. . . . diese Muster verbreiten und verbesse sie testen: erzeugen sie in uns das Gefühl, d Umgebung lebt?

16. . . . einzelne Muster verbinden zu einer Sp

Entwurfsmuster (Beispi

Dekorierer

Beispiel:

```
interface I { void m (); }
class C implements I { void m ()
class D implements I {
  C contents; void m () { .. }
}
```

Ein Dekorierer D fÃ¼r eine Klasse C erfÃ¼llt d Schnittstelle I, die Implementierung fÃ¼gt neu Eigenschaften hinzu.

Vorteil: flexibler als statische Implementierun

Befehl

Beispiel:

```
interface ActionListener {
  void actionPerformed( Action
}

JButton b = new JButton ();
b.addActionListener (new Actio
  public void actionPerformed
} );
```

trennt Befehls-Erzeugung von -Ausführur ermöglicht Verarbeitung von Befehlen (au speichern, wiederholen)

Erzeugungsmuster Prototy

- Prototyp hat eine Methode `clone`, die eine
- in Java kann jedes Objekt als Prototyp dien `clone` gehört zu Klasse `Object`)
- Default-Implementierung ist *flache Kopie*, d Attributen, die Verweise (auf Objekte) sind, Verweis kopiert, nicht sein Ziel.
- In Sprachen ohne Zuweisungen braucht m (alle Datenstrukturen sind dann persistent).

Kompositum - Aufgabe

verschiedene (auch zusammengesetzte) geo Objekte

Kompositum - Anwendun

```
interface Geo {
    Box bounds ();
    Geo [] teile ();
    void draw ();
}
class Kreis implements Geo { .. }
class Neben implements Geo {
    Neben (Geo links, Geo Rechts)
}
```


Fliegengewicht (Beisp

Beispiel: Schriftzeichen

- Höhe, Breite, Bitmap, Position (Zeile, Sp
- intrinsischer (innerer) und extrinsischer Zustand
- intrinsischer im Objekt, extrinsischer in

Konstruktor/Fabrik

- Konstruktor
- Fabrikmethode (statisch)

  Vorteil: Objekterzeugung wird versteckt, err sharing von Objekten
- abstrakte Fabrik

  Vorteil: erzeugte Klasse wird versteckt, meh

Erzeugungsmuster Singlet

- Eine Klasse ist eine *Singleton-Klasse*, wen nur ein Objekt geben soll.
- Möglichkeit in Java: Konstruktor privat, Fact

ohne Entwurfsmuster:

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

Finde wenigstens sieben (Entwurfs-)Fehler u wahrscheinlichen Auswirkungen. . .
(später: *code smells* → *refactoring*)

Fliegengewicht

- ein Teil des Zustandes wird *externalisie*

  d. h. weiter außen verwaltet (in der Fab
- dann gibt es nur noch wenige verschied Zustände

  d. h. man braucht nur wenige verschied

Beobachter: Beispiel

```
public class Counter extends Obse
    private int count = 0;
    public void step () { this.co
        this.setChanged();
        this.notifyObservers();
public class Watcher implements O
    private final int threshold;
    public void update(Observable
       if (((Counter)o).getCount(
           System.out.println ("a
public static void main(String[]
    Counter c = new Counter (); W
    c.addObserver(w); c.step(); c
```


MVC-Beispiel

```
public class View implements Obse
    private Label l = new Label("
    Component getComponent() { re
    public void update(Observable
        this.l.setText(Integer.to
}   }
public class Simple_MVC_Applet ex
    Button b = new Button ("step"
    Counter c = new Counter (); V
    public void init () {
        this.add (b); this.add(v.
        c.addObserver(v);
        b.addActionListener(new A
```


javax.swing und MVC

Swing benutzt vereinfachtes MVC (M getrenn
C gemeinsam).
Literatur:

- The Swing Tutorial `http://java.sun.co`
  `books/tutorial/uiswing/`
- Guido Krüger: Handbuch der Java-Program
  Addison-Wesley, 2003, Kapitel 35–38

Swing: Bäume

```
// Model:
class Model implements TreeMod
TreeModel m = new Model ( .. )

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

// Steuerung:
t.addTreeSelectionListener(new
   public void valueChanged(Tr

// Änderungen des Modells:
m.addTreeModelListener(..)
```


Verhaltensmuster: Beobacl

- Subjekt: class Observable
  - anmelden: void addObserver (Observer c
  - abmelden: void deleteObserver (Observe
  - Zustandsänderung: void setChanged ()
  - Benachrichtigung: void notifyObservers(.
- Beobachter: interface Observer
  - aktualisiere: void update (. . .)

Model/View/Controller

(Modell/Anzeige/Steuerung)
(engl. *to control* = steuern, *nicht:* kontrolliere
Beispiel:

- Daten-Modell: Zähler(stand)
- Anzeige: Label mit Zahl oder . . .
- Steuerung: Button


```
public void actionPer
  c.step();
} }); } }
```


Swing: Datenmodel

```
JSlider top = new JSlider(JSli
JSlider bot = new JSlider(JSli
bot.setModel(top.getModel());
```

Aufgabe: unterer Wert soll gleich 100 - ob

Muster: Iterator

```
Collection<Integer> c =
    Arrays.asList(new Integer[] {
Iterator<Integer> it = c.iterator
while (it.hasNext()) {
  Integer x = it.next ();
  System.out.println (x);
}
interface Iterator<E> {
    boolean hasNext (); // bleibt
    E next (); // schaltet weiter
}
for (Integer x : c) { ... } // be
```



```
  // TODO: wird delegiert
  public String toString() {  }

  // TODO, soll Folge [ lo, lo +
  public static Zahlenfolge range
}
```

testen:

```
public class Main {
  public static void main(String[
    Zahlenfolge f = Zahlenfolge.r
    System.out.println (f);
  }
}
```



```
    return accu;
  }
}
```

Beispiel:

```
public class Operation {
  static int summe (Zahlenfolge f
    return f.visit(new Zahlenfolg
      public int empty() {
        return 0;
      }
      public int nonempty(int pre
        return previous_result +
      }
    });
```


Besucher (Aufgabe

schreibe Methoden für

- Produkt
- Minimum, Maximum
- Wert als Binärzahl, Bsp:

```
Operation.binary (new Zahlenf
```


Beispiel MVC: Matrix-GU

http://dfa.imn.htwk-leipzig.de/matchbo

Aufgaben:

- deaktiviere Gleiter in Ausgabe-Matrix
- fixiere Eingaben oben links und unten recht
- teste, ob Ausgabe oben rechts positiv
- Auswahlmöglichkeit für Alphabet und Produ

Muster: Besucher

Klasse für Zahlenfolgen:

```
public class Zahlenfolge {

  private final List<Integer> con

  // Konstruktor mit variabler Ar
  public Zahlenfolge(Integer ...
         this.contents = new Lin
         this.contents.addAll(Ar
  }

  // TODO: wird delegiert
  public void add (int x) {  }
```


Definition eines Besucher-Objektes (inneres

```
class Zahlenfolge { ...
  public interface Visitor {
    int empty();
    int nonempty(int previous_res
  }
}
```

Behandlung eines Besuchers

```
class Zahlenfolge { ...
  public int visit(Visitor v) {
    int accu = v.empty();
    for (int x : this.contents) {
      accu = v.nonempty(accu, x);
    }
```



```
  }
}
```

```
  public <R> R visit(Visitor<R> v
    R accu = v.empty();
    for (int x : this.contents) {
      accu = v.nonempty(accu, x);
    }
    return accu;
  }
}
```

Beispiel:

```
public class Operation {
  static boolean contains (Zahlen
    return f.visit(new Zahlenfolg
      public Boolean empty() { re
      ...
```


Kompositum (Wdhlg.)

in Wirklichkeit handelt es sich um *algebraisch*

```
data Tree k = Leaf { key  :: k }
    | Branch { left :: Tree k, rig
```

Modellierung in Java (als *Kompositum*)

```
interface Tree<K> { }
class Leaf<K>   implements Tree<K
  Leaf(E key) { .. }
}
class Branch<K> implements Tree<K
  Branch(Tree<K> left, Tree<K> ri
}
```


Besucher für Bäume (Kompo

(dieses Beispiel sinngemäß aus: Naftalin, Wa
Generics and Collections, O'Reilly 2006.)

für jeden Teilnehmer des Kompositums eine

```
interface Visitor<K,R> { // mit R
    R leaf (K x);
    R branch (R left, R right);
}
```

der Gast nimmt Besucher auf:

```
interface Tree<K> {
  <R> R visit (Visitor<K,R> v)
}
```


Generischer Besucher

Eigentlich soll ja auch das gehen,

```
Operation.contains (new Zahlenfol
```

geht aber nicht, weil der Rückgabewert bishe
ist.

Lösung der Rückgabetyp wird ein Parameter

```
class Zahlenfolge { ...
  public interface Visitor<R> {
    R empty();
    R nonempty(R previous_result,
  }
}
```

Behandlung eines Besuchers

```
class Zahlenfolge { ...
```



```
  }
}
```


Bäume (Aufgabe I)

Konstruktoren, toString, Testmethode

```
class Trees {
    // vollst. bin. Baum der Höhe
    static Tree<Integer> full (in
}
System.out.println (Trees.full(1)
   ==> Branch{left=Leaf{key=0},ri
```


Bäume (Aufgabe II

Benutzung des Besuchers Anzahl der Blä

```
class Trees {
  static <K> int leaves (Tree<
    return t.visit(new Tree.Vi
      public Integer branch(In
        return left + right;
      }
      public Integer leaf(K ke
        return 1;
      }
    });
  }
}
```

Vergleich Iterator/Besuch

- Iterator: benutzt als Verweis in Datenstruktu Benutzer des Iterators steuert den Program
- Besucher: enthält Befehl, der für jedes Eler Struktur auszuführen ist, Struktur steuert A

Beispiel: Summation

Muster: Interpreter (Motivat

(Wdhlg. Iterator)

```
enum Color { Red, Green, Blue }
class Data { int x; color c, }
class Store {
  Collection<Data> contents;
  Iterable<Data> all ();
}

interface Iterable<K> { Iterator<
interface Iterator<K> { .. }
```


Muster: Interpreter (Realisier

algebraischer Datentyp (= Kompositum) für c Beschreibung von Eigenschaften

```
interface Property { }
// Blätter (Konstanten)
class Has_Color { Color c }
class Less_Than { int x }
// Verzweigungen (Kombinatoren)
...
```

und Programm zur Auswertung einer (zusammengesetzten) Eigenschaft für gegeb

Iteratoren und Bedarfsaus

(vgl. Beispiel Iterator/Query)

das Konstruieren von Iteratoren ist eine C man kÃ¶nnte auch jeweils gesamte Colle hat jedoch Angst vor dem Rechenaufwan viel) und Speicherverbrauch (zu viel).

LÃ¶sung:

- Bedarfsauswertung
- Garbage Collection

Iterator für Bäume

Ansatz:

```
public class TreeIterator<K> impl
  // TODO:
  public boolean hasNext() { .. }
  public K next() { .. }
  // ANSATZ: aktuelle Position (P
  private final Stack<Tree<K>> pa
  public TreeIterator(Tree<K> bas
    this.path.push(base);
} }
```


Entwurfsfragen bei Bäum

- Knoten sind *innere* (Verzweigung) und *äuß*
- Die “richtige” Realisierung ist Kompositum

```
interface Tree<K>;
class Branch<K> implements Tree<
class Leaf<K> implements Tree<K>
```

- Möglichkeiten für Schlüssel: in allen Knoter nur außen.

Wenn Blätter keine Schlüssel haben, geht es

```
class Tree<K> // für Verzweigunge
```

Jein. — betrachte Implementierung in java.ut

Muster: Interpreter (Motivat

aber wie macht man das besser:

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


Übung Woche 17

- Hausaufgabe auswerten: Besucher für
- Iterator für Binärbaum
- Interpreter für Property (vervollständige
- Interpreter für Property verbessert impl (Collection nicht kopieren)

Zustand (II)

- unveränderliche Zustandsobjekte: als Argu Resultat von Methoden
- Verwendung früherer Zustandsobjekte (für test)

Dependeny Injection

- Service Locator oder DI
- Constructor oder Setter
- Code oder Config-File

in jedem Falle: trenne Konfiguration von Benu

- unflexibel: algorithmische Abhängigkeiten
- unflexibel: enge Kopplung
- unflexibel: Implementierungs-Vererbung
- Unmöglichkeit, Klassen direkt zu ändern

CVS-Überblick

(concurrent version system)

- Server: Archiv (repository), Nutzer-Auth ggf. weitere Dienste (`cvsweb`)
- Client (Nutzerschnittstelle): Kommando `cvs checkout foobar` oder grafisch in Eclipse)

Ein Archiv (repository) besteht aus mehre Verzeichnissen)
Die lokale Kopie der (Sub-)Module beim C Sandkasten (sandbox).

Entwurfsmuster: Zustan

Zustand eines Objektes = Belegung seiner A
Zustand erschwert Programmanalyse und -V
(muß bei jedem Methodenaufruf berücksichti
Abhilfe: Trennung in

- Zustandsobjekt (nur Daten)
- Handlungsobjekt (nur Methoden)

jede Methode bekommt Zustandsobjekt als A

Dependency Injection

Martin Fowler, `http://www.martinfowle articles/injection.html`
Abhängigkeiten zwischen Objekten sollen

- sichtbar und
- konfigurierbar sein.

Formen:

- Constructor injection (bevorzugt)
- Setter injection

Veränderungen in Entwürfen vor

- unflexibel: Erzeugen eines Elements durch seiner Klasse

  flexibel: abstrakte Fabrik, Fabrikmethode, F
- unflexibel: Abhängigkeit von speziellen Ope

  flexibel: Zuständigkeitskette, Befehl
- unflexibel: Abhängigkeit von Hard- und Soft

  flexibel: abstrakte Fabrik, Brücke
- unflexibel: Abhängigkeit von Objektrepräse -implementierung

Quelltextverwaltung n

Anwendung, Ziele

- aktuelle Quelltexte eines Projektes sich
- auch frühere Versionen sichern
- gleichzeitiges Arbeiten mehrere Entwick
- . . . an unterschiedlichen Versionen

Das Management bezieht sich auf *Quelle* Makefile)
abgeleitete Dateien (.obj, .exe, .pdf, .class erzeugt, stehen aber *nicht* im Archiv

CVS-Tätigkeiten (II)

während der Projektarbeit:

- Clienten:
  - vor Arbeit in sandbox: Änderungen (der a Programmierer) vom Server holen (`cvs u`
  - nach Arbeit in sandbox: eigene Änderung Server schicken (`cvs commit`)

Welche Formate?

- Quellen sollen Text-Dateien sein, human-re Zeilenstruktur: ermöglicht Feststellen und Zusammenfügen von unabhängigen Änder
- ergibt Konflikt mit Werkzeugen (Editoren, ID Dokumente nur in Binärformat abspeichern sowieso *evil*, siehe Robert Brown: Readabl File Formats, `http://www.few.vu.nl/~ read_and_open.html`
- Programme mit grafischer Ein- und Ausgab Informationen *vollständig* von und nach Te können
  (Bsp: UML-Modelle als XMI darstellen)

Logging (II)

alle Log-Messages für eine Datei:

```
cvs log foo.c
```

Die Log-Message soll den *Grund* der Änderu denn den *Inhalt* kann man im Quelltext nachl

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

finde entsprechendes Eclipse-Kommando!

Authentifizierung mit SSI

- Schlüsselpaar erzeugen (ssh-keygen)
- öffentlichen Schlüssel auf Zielrechner in (ssh-copy-id)
- privaten Schlüssel in Agenten laden (ss

CVS-Tätigkeiten (I)

Bei Projektbeginn:

- Server-Admin:
  - Repository und Accounts anlegen (`cvs i`
- Clienten:
  - neues Modul zu Repository hinzufügen (`c`
  - Modul in sandbox kopieren (`cvs checko`

Konflikte verhindern oder lö

- ein Programmierer: editiert ein File, oder ec
- mehrere Programmierer:
  - strenger Ansatz: nur einer darf editieren beim checkout wird Datei im Repository m (gelockt), bei commit wird lock entfernt
  - nachgiebiger Ansatz (CVS): jeder darf ed commit prüft Server auf Konflikte und versucht, Änderungen zusammenzuf

Logging (I)

bei commit soll ein Kommentar angegeben w man später nachlesen kann, welche Änderur welchem Grund ausgeführt wurden.

- Eclipse: textarea
- `cvs commit -m "neues Feature: --`
- ```
  emacs -f server-start &
  export EDITOR=emacsclient
  cvs commit
  ```

  ergibt neuen Emacs-Buffer, beenden mit `C-`

Authentifizierung

- lokal (Nutzer ist auf Server eingeloggt):

  ```
  export CVSROOT=/var/lib/cvs/f
  cvs checkout bar
  ```

- remote, unsicher (Paßwort unverschlüs

  ```
  export CVSROOT=:pserver:user@
  cvs login
  ```

- remote, sicher

  ```
  export CVS_RSH=ssh2
  export CVSROOT=:ext:user@host
  ```
  ```

Subversion (II)

- Server speichert Dateien und Zusatz-Inforn Datenbank (Berkeley DB) (CVS: im Filesys

  unterstützt auch Umbenennen usw. mit Bev History.

- Subversion läuft als standalone-Server ode Apache2-Modul (benutzt WebDAV)

- Kommandozeilen-Client wie cvs, Grafische (TortoiseSVN), Webfrontends (viewCVS/vie

Weitere Erläuterungen zu Subversion im Vort Reimer (Seminar Software-Entwicklung) `htt imn.htwk-leipzig.de/~waldmann/edu`

Übung CVS

- ein CVS-Archiv ansehen (cvsweb-interface `http://dfa.imn.htwk-leipzig.de/c cvsweb/havannah/different-applet havannah`

- ein anderes Modul aus o. g. Repository anc auschecken (mit Eclipse):

  (Host: `dfa.imn.htwk-leipzig.de`, Pfad `/var/lib/cvs/havannah`, Modul `demo`, `pserver`, User: `anonymous`, kein Passwo

  Projekt als Java-Applet ausführen. . . . zeigt von Layout-Managern.

- ein eigenes Eclipse-Projekt als Modul zu de gruppen-eigenen CVS-Repository hinzufüg Share)

  [Daten ggf. für laufendes Semester/Server

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

  (CVS-Zugang benutzt Account im Linux-Po Gruppeneinteilung beachten)

- eine Datei ändern, commit; anderer Studer Gruppe: update

  was passiert bei gleichzeitigen Änderunger unabhängigen commits?

CVS – Merge

- 9:00 Heinz: checkout (Revision A)
- 9:10 Klaus: checkout (Revision A)
- 9:20 Heinz: editiert ($A \to H$)
- 9:30 Klaus: editiert ($A \to K$)
- 9:40 Heinz: commit (H)
- 9:50 Klaus: commit

  `up-to-date check failed`
- 9:51 Klaus: update

  `merging differences between` A
- 9:52 Klaus: commit

Subversion

`http://subversion.tigris.org/` — "a

- ähnliche Kommandos, aber anderes Model
- Client hat Sandbox *und* lokale Kopie des R

  deswegen sind weniger Server-Kontakte nö
- "commits are atomic" (CVS: commit einer e ist atomic)
- Versionsnummer bezieht sich auf Reposito einzelne Dateien)

  in Sandbox sind Dateien verschiedener Re gestattet

Darcs

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

- nicht Verwaltung von *Versionen*, sondern v

  gestattet paralleles Arbeiten an verschiede

- kein zentrales Repository

  (kann jedoch vereinbart werden)

vgl. Oberseminarvortrag

Applet-Fenster-Größe ändern (ziehen mit M

Noch weiter Komponenten (Buttons) und P eigenen Managern) hinzufügen.

CVS – Einzelheit

Datei-Status

`cvs status ; cvs -n -q update`

- Up-to-date:
  Datei in Sandbox und in Repository stir
- Locally modified (, added, removed):
  lokal geändert (aber noch nicht commit
- Needs Checkout (, Patch):
  im Repository geändert (wg. unabh. co
- Needs Merge:
  Lokal geändert *und* in Repository geän

Unterschiede zwischen Dat

- welche Zeilen wurden geändert, gelöscht, h
- ähnliches Problem beim Vergleich von DNS
- Algorithmus: Eugene Myers: *An $O(ND)$ Di Algorithm and its Variations*, Algorithmica V 1986, pp. 251-266, `http://www.xmailserver.org/diff2`
- Implementierung (Richard Stallman, Paul E `http://cvs.sourceforge.net/viewc *checkout*/cvsgui/cvsgui/cvs-1.1 analyze.c`

LCS

Idee: die beiden Aufgaben sind äquivalent:

- kürzeste Edit-Sequenz finden
- längste gemeinsame Teilfolge (longest com subsequence) finden

Beispiel: $y = AB\boxed{C}\boxed{AB}B\boxed{A}, z = \boxed{C}B\boxed{A}$
für $x = CABA$ gilt $x \leq y$ und $x \leq z$,
wobei die Relation $\leq$ auf Σ^* so definiert ist:
$u \leq v$, falls man u aus v durch *Löschen* einig erhält (jedoch *ohne* die Reihenfolge der übrig Buchstaben zu ändern)
vgl. mit Ausgabe von `diff`

Die Einbettungs-Relation

Begriffe (für Halbordnungen):

- Kette: Menge von paarweise vergleichbare
- Antikette: Menge von paarweise unvergleic Elementen

Sätze: für $\leq$ ist

- jede Kette endlich
- jede Antikette endlich

Beispiel: bestimme die Menge der $\leq$-minimal für ...

Aufgaben (autotool) zu

- LCS-Beispiel (das Beispiel aus Vorlesu
- LCS-Quiz (gewürfelt - Pflicht!)
- LCS-Long (Highscore - Kür)

Drei-Wege-Diff

benutzt Kommando `diff3` K A H

- changes von $A \rightarrow H$ berechnen
- ... und auf K anwenden (falls das geht)

Konflikte werden in K (d. h. beim Clienten) m und müssen vor dem nächsten commit repar

tatsächlich wird `diff3` nicht als externer Pro aufgerufen, sondern als internes Unterprogra (→ unabhängig vom Prozeß-Begriff des jewe

- siehe auch Diskussion hier: `http://c2.com/cgi/wiki?DiffAlgor`

Die Einbettungs-Relation

Def: $u \leq v$, falls u aus v durch Löschen von E

- ist Halbordnung (transitiv, reflexiv, antisymm
- ist keine totale Ordnung

Testfragen:

- Gegeben v. Für wieviele u gilt $u \leq v$?
- Effizienter Algorithmus für: Eingabe u, v, Au (Boolean)

Die Einbettungs-Relatio

Die Endlichkeit von Ketten und Antiketten Einbettung gilt für

- Listen
- Bäume (Satz von Kruskal, 1960)
- Graphen (Satz von Robertson/Seymou

  (Beweis über insgesamt 500 Seiten übe ca. 2000)

  vgl. Kapitel 12 in: Reinhard Diestel: Gra `http://www.math.uni-hamburg.d diestel/books/graph.theory/`

Optimierung durch bottom-up-Reihenfolge!


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

    public static void main(String[] arg
        String xs = "ABCABBA";
        String ys = "CBABAC";
        System.out.println (lcs (xs.toCh
    }
}
```

Aufgaben:

LCS – eingeschränkt linea

Suche nach einer LCS = Suchen eines kurze
`(0,0)` nach `(xs.length-1, ys.length-`
einzelne Kanten verlaufen

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

  gemeinsamer Buchstabe

diff und LCS

Bei `diff` werden nicht einzelne *Zeichen*
sondern ganze *Zeilen*.
das gestattet/erfordert Optimierungen:

- Zeilen feststellen, die nur in einer der b
  vorkommen, und entfernen

  `diff/analyze.c:discard_confus`
- Zum Vergleich der Zeilen Hash-Codes

  `diff/io.c:find_and_hash_each_`

siehe Quellen `http://cvs.sourcefor`
`viewcvs.py/cvsgui/cvsgui/cvs-1`
Aufgabe: wo sind die Quellen für die CVS

LCS — naiver Algorithmus (expo

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

top-down: sehr viele rekursive Aufrufe ...
aber nicht viele *verschiedene* ...

LCS — bottom-up (quadratisch)

```
class LCS {

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


- vervollständigen Sie die Methode `LCS.lcs`
- bauen Sie eine Möglichkeit ein, nicht nur di
  längsten gemeinsamen Teilfolge zu bestimm
  auch eine solche Folge selbst auszugeben.

  Hinweis: `int [][] a` wie oben ausrechne
  vom Ende zum Anfang durchlaufen (ohne g
  suchen).

  damit dann die autotool-Aufgaben lösen.

Optimierungen:

- Suche nur in der Nähe der Diagonalen
- Beginne Suche von beiden Endpunkten

Wenn nur $\leq D$ Abweichungen vorkomme
es, einen Bereich der Größe $D \cdot N$ zu be
$O(ND)$ *Difference Algorithm and its Varia*

Mehr zu CVS

Keyword Expansion

in den gemanagten Dateien werden Schlüsse
`commit` durch aktuelle Daten ersetzt.
Zu Beginn: `$Key$`, danach `$Key: Value $`

```
$Id: keyword.tex,v 1.1 2005-04-18 16:59:
$Author: waldmann $
$Date: 2005-04-18 16:59:07 $
$Header: /var/lib/cvs/edu/edu/ss08/st2/f
$Name:  $
$RCSfile: keyword.tex,v $
$Revision: 1.1 $
$Source: /var/lib/cvs/edu/edu/ss08/st2/f
$State: Exp $
```


Log-Messages lenken vom eigentlichen Quel
soll ja *ohne* Kenntnis der Geschichte verstän

Symbolische Revisionen (Ta

jedes Dokument hat seine eigene Versionsnu
(revision), z. B. (dieses Dokument):
`$Revision: 1.1 $`
Es gibt also *keine* Version eines gesamten M
symbolische Revisionen (tags).

```
cvs tag -r release-1_0
```

Vorsicht: im Namen sind keine Punkte erlaub
die Revisionsnamen können bei
`diff, update, checkout` benutzt werde

Branches (II)

Aufgaben:

- Betrachten Sie Tags/Branches im CVS-
  `http://ccvs.cvshome.org/sourc`
  `ccvs/diff/` z. B. Datei `diff3.c`
- Lesen Sie Erläuterungen zu Branches i
  von Thomas Preuß (Seminar Software-
  `http://www.imn.htwk-leipzig.d`
  `edu/ss04/se/`

Eclipse?

Das Keyword `$Log$`

. . . wird durch die Liste *aller* Log-Messages e
Damit das als Kommentar in Quelltexten steh
erhält jede Zeile den gleichen Präfix:

```
// $Log: log.tex,v $
// Revision 1.1  2005-04-18 16:59:07  wa
// files
//
// Revision 1.2  2004/05/10 08:34:42  wa
// besseres LaTeX-display
//
// Revision 1.1  2004/05/10 08:26:25  wa
// Vorlesung 10. 5.
//
```

Die Nützlichkeit dieses Features ist umstritte

Text- und Binär-Dateien

per Default werden gemanagte Dateien als T
behandelt:

- Keyword Expansion findet statt
- Zeilenenden werden systemspezifisch übe
  (DOS: CR LF, Unix: LF)

das ist für Binärdateien (Bilder, Echsen) tödli
diese gehören normalerweise auch nicht ins
Falls es doch nötig ist, kann man Dateien als
markieren, dann finden keine Ersetzungen st

Verzweigungen (branc

Dei Geschichte eines Dokumentes ist pe
kann jedoch bei Bedarf zu einem Baum v
übliches Vorgehen bei größeren Projekte

- ein *main branch*
- evtl. experimentelle branches
- akzeptierte Features werden in main-br
  aufgenommen
- bei jedem Release wird ein release-bra
- wichtige Bugfixes aus main-branch wer
  release-branches angewendet

CVS-Benachrichtigungen

Optional: In Datei
`/var/lib/cvs/case_XX/CVSROOT/user`
Address-Umsetzung:
`heinz:heinz@woanders.com`
Diese Datei ist nicht von CVS gemanagt, mul
erzeugt werden.

Fehlermeldungen

sollen enthalten

- Systemvoraussetzungen
- Arbeitsschritte
- beobachtetes Verhalten
- erwartetes Verhalten

Verwaltung z. B. mit Bugzilla, Trac
Vgl. Seminarvortrag D. Ehricht:
`http://www.imn.htwk-leipzig.de/~w`
`edu/ss04/se/ehricht/bugzilla.pdf`

Dynamische Tests

- Testfall: Satz von Testdaten
- Testtreiber zur Ablaufsteuerung
- ggf. *instrumentiertes* Programm zur Protok

Beispiele (f. Instrumetierung):

- Debugger: fügt Informationen über Zeilennu Objektcode ein

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

- Profiler: Code-Ausführung wird regelmäßig und „aktuelle Zeile“ notiert, anschließend S

Black-Box-Tests

ohne Programmstruktur zu berücksichtige

- typische Eingaben (Normalbetrieb)
  alle wesentlichen (Anwendungs-)Fälle a
  (Bsp: gerade und ungerade Länge eine
- extreme Eingaben
  sehr große, sehr kleine, fehlerhafte
- zufällige Eingaben
  durch geeigneten Generator erzeugt

während Produktentwicklung:
Testmenge ständig erweitern,
frühere Tests immer wiederholen (regress

CVS-Benachrichtigunge

Client-Befehl: `cvs watch add` beginnt *Beo*
eines (Teil-)Moduls: bei jeder Aktionen (comn
Repository wird Email an *watchers* versandt.
In Datei `/var/lib/cvs/case_XX/CVSROO`
steht der Mailer-Aufruf (per Default auskomm
`ALL mail %s -s "CVS notification"`
Beachte: das Verzeichnis CVSROOT verhält
ein CVS-Modul, d. h.

```
cvs checkout CVSROOT
cd CVSROOT
emacs notify
cvs commit -m mail
```


Produktqualität (analyti

Klassifikation der Verfahr

- Verifizieren (= Korrektheit beweisen)
  – Verifizieren
  – symbolisches Ausführen
- Testen (= Fehler erkennen)
  – statisch (z. B. Inspektion)
  – dynamisch (Programm-Ausführung)
- Analysieren (= Eigenschaften vermessen/c
  – Quelltextzeilen (gesamt, pro Methode, pr
  – Klassen (Anzahl, Kopplung)
  – Profiling (. . . später mehr dazu)

Testen und Schnittstelle

- Test für Gesamtsystem (schließlich) oder T
- Teile definiert durch Schnittstellen
- Schnittstelle ⇒ Spezifikation
- Spezifikation ⇒ Testfälle

Testen . . .

- unterhalb einer Schnittstelle (unit test)
- oberhalb (mock objects) (vgl. dependency i

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

Dynamische Tests: Black

- Strukturtests (white box)
  – programmablauf-orientiert
  – datenfluß-orientiert
- Funktionale Tests (black box)
- Mischformen (unit test)

Mischformen

- Testfälle für jedes Teilprodukt, z. B. jede Me
  (d. h. Teile der Programmstruktur werden b
- Durchführung kann automatisiert werden (

JUnit ist in Eclipse-IDE integriert (New → JU
→ 4.0)

Delta Debugging

Andreas Zeller: *From automated Testing to A*
Debugging, automatische Konstruktion von

- minimalen Bugreports
- Fehlerursachen (bei großen Patches)

Modell:

- `test : Set<Patch> -> { OK, FAIL,`
- `dd(low,high,n)=(x,y)`
  - Vorbedingung `low` $\subseteq$ `high`,
    `test(low)=OK, test(high)=FAIL`
  - Nachbedingung `x` $\subseteq$ `y`,
    `size(y) - size(x)` „möglichst klein“

Übung zum Tester

Die folgende Methode soll binäre Suche i

- wenn (Vorbedingung) $\forall k : x[k] \leq x[k+$
- dann (Nachbedingung) gilt für den Rück
  `binsearch(x,i)`:

  falls i in $x[..]$ vorkommt,
  dann $x[p] = i$, sonst $p = -1$.

```
public static int binsearch (i
    int n = x.length;
    int low = 0;
    int high = n;
    while (low < high) {
```


Probleme mit GUI-Tests

schwierig sind Tests, die sich nicht automatis
(GUIs: Eingaben mit Maus, Ausgaben als Gra
zur Unterstützung sollte jede Komponente ne
GUI-Schnittstelle bieten:

- auch eine API-Schnittstelle (für (Test)progra
- und ein Text-Interface (Kommando-Interpre

Bsp: Emacs: `M-x kill-rectangle` oder `C`

Testen mit JUnit

Kent Beck and Erich Gamma,
`http://junit.org/index.htm`

```
import static org.junit.Assert.*;
class XTest {

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

`http://www-128.ibm.com/developerw`
`library/j-junit4.html`

JUnit und Extreme Programi

Kent Beck empfiehlt *test driven apporach*:

- *erst* alle Test-Methoden schreiben,
- *dann* eigentliche Methoden implementieren
- . . . bis sie die Tests bestehen (und nicht we
- Produkt-Eigenschaften, die sich nicht teste
  nicht vorhanden.
- zu jedem neuen Bugreport einen neuen Te

Testfall schreiben ist *Spezifizieren*, das geht
Implementieren voraus. — *Testen* der Implem
nur die zweitbeste Lösung (besser ist *Verifizi*

Delta Debugging (I

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

`http://www.infosun.fim.uni-pas`
`papers/computer2000/`

- Legen Sie einen JUnit-Testcase an (Eclipse → JUnit Test Case) mit etwa diesem Code:

```
int [] x = { 3, 4, 6, 8, 9 };
int p = binsearch (x, 4);
assertTrue (p == ???);
```

- Finden Sie Argumente, für die sich die Met fehlerhaft verhält.
- Reparieren Sie die Methode.
- Zusatz: Vergleichen Sie mit der entsprech aus der Java-Standard-Bibliothek (welche k welcher Name?) Sie benötigen dazu ein ko (mit Quelltexten).

Prüfen von Testabdeckung

mit Werkzeugunterstützung, Bsp.: *Profiler*: mißt bei Ausführung Anzahl der Ausführunge

- . . . jeder Anweisung (Zeile!)
- . . . jeder Verzweigung (then oder else)

(genügt für welche Abdeckungen?)

Profiling durch Instrumentieren (Anreichern)

- des Quelltextes
- oder der virtuellen Maschine


```
./heap > /dev/null
# welche Dateien wurden erzeugt?
gcov heap.cc
# welche Dateien wurden erzeugt?
```

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

- `heap` reparieren: `check` an geeigneten Ste um Fehler einzugrenzen
- `median3` analysieren: Testfälle schreiben (für: Anweisungsüberdeckung, Bedingungsü Pfadüberdeckung

  Überdeckungseigenschaften mit `gcov` prüf
- `median5` reparieren


```
unset LD_LIBRARY_PATH
/home/waldmann/built/eclipse-
```

(für JDK-1.6: TPTP-4.4 in Eclipse-3.3 (


```
        int mid = (low + high) /
        if (i < x[mid]) {
            high = mid;
        } else if (i > x[mid]) {
            low = mid;
        } else {
            return mid;
        }
    }
    return -1;
}
```

Aufgaben:

- Legen Sie eine Klasse an, die `binsearch`

Programmablauf-Tests

bezieht sich auf Programm-Ablauf-Graphen (Anweisungen, Kanten: mögliche Übergänge)

- Anweisungs-Überdeckung: jede Anweisung einmal ausgeführt
- Zweigüberdeckung: jede Kante mindestens durchlaufen — Beachte: `if (X) then {`
- Pfadüberdeckung: jeder Weg (Kantenfolge) einmal durchlaufen — Beachte: Schleifen (Durchlaufwege)
  Variante: jede Schleife (interior) höchstens
- Bedingungs-Überdeckung: jede atomare B einmal true, einmal false.

Übung Profiling (C++)

Beispiel-Programm(e):
`http://www.imn.htwk-leipzig.de/~w edu/ss04/case/programme/analyze/c`
Aufgaben:

- Kompilieren und ausführen für Profiling:

```
g++ -pg -fprofile-arcs heap.cc -
./heap > /dev/null
# welche Dateien wurden erzeugt?
gprof heap # Analyse
```

- Kompilieren und ausführen für Überdeckun

```
g++ -ftest-coverage -fprofile-ar
```


Profiling (Java)

- Kommandozeile: `java -Xprof ...`
- in Eclipse: benutzt TPTP `http://www articles/Article-TPTP-Profili tptpProfilingArticle.html http eclipse.org/tptp/home/documen profilingtool/profilingexampl`
- Installation: Eclipse → Help → Update
- im Pool vorbereitet, benötigt aber genau Eclipse-Installation und java-1.5

```
export PATH=/home/waldmann/bu
```

Kosten von Algorithmen sch

big-Oh-Notation zum Vergleich des Wachstur
Funktionen kennen und anwenden

- einfache Schleife
- geschachtelte Schleifen
- binäres Teilen
- (binäres) Teilen und Zusammenfügen
- Kombinatorische Explosion

(diese Liste aus Pragmatic Programmer, p. 1

die asymptotischen Laufzeiten lassen sich du
Optimierungen *nicht* ändern, also: vorher na
lohnt sich

Gefährliche „Optimierunge

Gefahr besteht immer, wenn die Programm-S
als die Denk-Struktur ist.

- anwendungsspezifische Datentypen vermie
  ausgepackt → primitive obsession

  (Indikator: String und int)
- existierende Frameworks ignoriert

  (Indikatoren: kein import java.util.*; sort sell
  geschrieben, XML-Dokument als String)
- Unterprogramm vermieden bzw. aufgelöst
  Methode (bei 5 Zeilen ist Schluß)

(später ausführlicher bei *code smells* → Refa

Code-Metriken: Halstaed

(zitiert nach Balzert, Softwaretechnik II)

- O Anzahl aller Operatoren/Operationen (Ak
- o Anzahl unterschiedlicher Operatoren/Ope
- A Anzahl aller Operanden/Argumente (Dat
- a Anzahl unterschiedlicher Operanden/Argu
- ($o + a$ Größe des Vokabulars, $O + A$ Größe
  Implementierung)

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

OO-Metriken

- Attribute bzw. Methoden pro Klasse
- Tiefe und Breite der Vererbungshierarch
- Kopplung (zwischen Klassen) wieviele a
  sind in einer Klasse bekannt? (je wenig
- Kohäsion (innerhalb einer Klasse): häng
  eng zusammen? (je enger, desto besse

Code-Optimierungen

Tony Hoare first said,
and Donald Knuth famously repeated,
Premature optimization is the root of all evil.

- erste Regel für Code-Optimierung: *don't do*
- zweite Regel: *. . . yet!*

Erst korrekten Code schreiben, *dann*
Ressourcenverbrauch messen (profiling),
dann eventuell kritische Stellen verbessern.

Besser ist natürlich: kritische Stellen vermeid
Bibliotheksfunktionen benutzen!
Die sind nämlich schon optimiert (Ü: sort, bin

Code-Transformationen zur Opti

(Jon Bentley: Programming Pearls, ACM Press

- Zeit sparen auf Kosten des Platzes:
  - Datenstrukturen anreichern (Komponente
  - Zwischenergebnisse speichern
  - Cache für häufig benutzte Objekte
- Platz sparen auf Kosten der Zeit:
  - Daten packen
  - Sprache/Interpreter (Bsp: Vektorgrafik sta
- Schleifen-Akrobatik, Unterprogramme auflö
  überlassen wir mal lieber dem Compiler/de
  Maschine

Code-Metriken

Welche Code-Eigenschaften kann man mess
sagen sie aus?

- Anzahl der Methoden pro Klasse
- Anzahl der . . . pro . . .
- textuelle Komplexität: Halstaed
- strukturelle Komplexität: McCabe
- OO-Klassenbeziehungen

Code-Metriken: McCa

(zitiert nach Balzert, Softwaretechnik II)
zyklomatische Zahl (des Ablaufgraphen G
$|E| - |V| + 2c$ wobei $c =$ Anzahl der
Zusammenhangskomponenten
(Beispiele)
Idee: durch Hinzufügen einer Schleife, Ve
steigt dieser Wert um eins.

Kohäsion: Henderson-Sell

- M Menge der Methoden
- A Menge der Attribute
- für $a \in A$: $Z(a) =$ Menge der Methoden, die
- z Mittelwert von $|Z(a)|$ über $a \in A$

fehlende Kohäsion: $\frac{|M|-z}{|M|-1}$
(kleinere Werte sind besser)

Auswertung der Umfra

Richtig beobachtet (Teil I

Der Dozent scheint davon auszugehen, dass
Student alle mathematischen Sachverhalte d
Semester 100%-ig eingeprägt hat und diese
herunterbeten kann.
Er scheint offenbar recht genervt von der Tat
das dem nicht so ist wie sich in den Seminare
Nicht die Tatsache, dass er diese Sachverhal
fragwürdig, sondern seine Reaktion auf die T
die Studenten diese Sachverhalte oft nicht oc
bruchstückhaft erklären können.

Punktzahlen

`http://www.imn.htwk-leipzig.de/~w`
`edu/ss08/st2/umfrage/Report_`
`Softwaretechnik2_Waldmann.pdf`

Refaktoring: Definiti

Martin Fowler: *Refactoring: Improving the Existing Code*, A.-W. 1999,
`http://www.refactoring.com/`
Def: Software so ändern, daß sich

- externes Verhalten nicht ändert,
- interne Struktur verbessert.

siehe auch William C. Wake: *Refactoring* 2004 `http://www.xp123.com/rwb/`
und Stefan Buchholz: Refactoring (Semir
`http://www.imn.htwk-leipzig.de`
`edu/current/se/talk/sbuchhol/`

Kohäsion: Chidamber und Ke

(Anzahl der Paare von Methoden, die kein ge
Attribut benutzen) − (Anzahl der Paare von M
ein gemeinsames Attribut benutzen)
bezeichnet fehlende Kohäsion, d. h. kleinere
besser.

Code-Metriken (Eclipse)

Eclipse Code Metrics Plugin installieren und
Projekt anwenden.

- `http://eclipse-metrics.sourcefor`
- Installieren in Eclipse: Help → Software Up
  → Search for New → New (Remote/Local)
- Projekt → Properties → Metrics → Enable,
  → Build, dann anschauen

Richtig beobachtet (Teil I

Das Skript ist eigentlich sinnlos, da sehr wen
vorhanden ist.
Man findet oft nur Überschriften mit halbfertig
und Literaturverweise

Refactoring

Herkunft

Kent Beck: *Extreme Programming*, Addis

- Paar-Programmierung (zwei Leute, ein
- test driven: erst Test schreiben, dann P
  implementieren
- Design nicht fixiert, sondern flexibel

Refaktorisierungen

- Entwurfsänderungen ...

  verwende Entwurfsmuster!

- „kleine“ Änderungen
  - Abstraktionen ausdrücken:
    neue Schnittstelle, Klasse, Methode, (tem
  - Attribut bewegen, Methode bewegen (in a

Duplicated Code → Schablo

duplizierter Code wird verhindert/entfernt dur

- *Schablonen* (beschreiben das Gemeinsam
- mit *Parametern* (beschreiben die Unterschi

Beispiel dafür:

- Unterprogramm (Parameter: Daten, Result
- polymorphe Klasse (Parameter: Typen, Re
- Unterprogramm höherer Ordnung (Parame
  Programm, Resultat: Programm)

wenn Programme als Parameter nicht erlaub
dann werden sie als Methoden von Objekten
Entwurfsmuster Besucher)

Primitive Daten (*primitive obse*

Symptome: Benutzung von `int, float, S`
Ursachen:

- fehlende Klasse:

  z. B. String → FilePath, Email, ...

- schlecht implementiertes Fliegengewicht

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

- simulierter Attributname:

  z. B. `Map<String,String> m; m.get(`

Behebung: Klassen benutzen, Array durch Ol
(z. B. `class M { String foo; ...}`)


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

Benutzung:

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


Refactoring anwenden

- mancher Code „riecht“(schlecht)

  (Liste von *smells*)

- er (oder anderer) muß geändert werden

  (Liste von *refactorings*, Werkzeugunterstütz

- Änderungen (vorher!) durch Tests absicher

  (JUnit)

Code Smell # 1: Duplicated C

jede Idee sollte an *genau einer* Stelle im Coc
werden:
Code wird dadurch

- leichter verständlich
- leichter änderbar

Verdoppelter Quelltext (copy–paste) führt imr
Wartungsproblemen.

Size does matter

weitere Code smells:

- lange Methode
- große Klasse
- lange Parameterliste

oft verursacht durch anderen Smell: Primitive

Typsichere Aufzählun

Definition (einfach)

```
public enum Figur { Bauer, Turm
```

Definition mit Attribut (aus JLS)

```
public enum Coin {
    PENNY(1), NICKEL(5), DIME(
    Coin(int value) { this.valu
    private final int value;
    public int value() { return
}
```

Definition mit Methode:

```
public enum Figur {
  Bauer { int wert () { return
  Turm  { int wert () { return
```

Datenklumpen—Beispie

Beispiel für Datenklumpen und -Vermeidung:

```
java.awt

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

Vergleichen Sie die Lesbarkeit/Sicherheit vor

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

Vergleichen Sie:

```
java.awt.Graphics: drawRectangle(
java.awt.Graphics2D: draw (Shape)
    class Rectangle implements Sh
```


Aufgabe Refactoring

Würfelspiel-Simulation:

Schummelmex: zwei (mehrere) Spieler, ein W
Spielzug ist: aufdecken oder (verdeckt würfel
ansagen, weitergeben) bei Aufdecken wird v
mit vorigem Wurf verglichen, das ergibt Verlu
den Aufdecker oder den Aufgedeckten

- Vor Refactoring:
  `http://www.imn.htwk-leipzig.de/~`
  `edu/ss07/case/programs/refactor/`
  Welche Code-Smells?
- Nach erstem Refactoring:
  `http://www.imn.htwk-leipzig.de/~`

Temporäre Attribute

Symptom: viele `if (null == foo)`

Ursache: Attribut hat nur während bestimmte
Programmteile einen sinnvollen Wert

Abhilfe: das ist kein Attribut, sondern eine ter
Variable.

Name enthält Typ

Symptome:

- Methodenname bezeichnet Typ des Arg
  Resultats

  `class Library { addBook( Book`
- Attribut- oder Variablenname bezeichne
  Ungarische Notation) z. B. `char ** p`

  siehe
  `http://ootips.org/hungarian-n`
- (grundsätzlich) Name bezeichnet Imple
  Bedeutung

Verwendung von Daten: Datenk

Fehler: Klumpen von Daten wird immer geme

```
String infile_base; String infile
String outfile_base; String outfi

static boolean is_writable (Strin
```

Indikator: ähnliche, schematische Attributnam
Lösung: Klasse definieren

```
class File { String base; String

static boolean is_writable (File
```


Verwendung von Daten: Data

Fehler:

Klasse mit Attributen, aber ohne Methoden.

```
class File { String base; String
```

Lösung:

finde typische Verwendung der Attribute in Cl
(Bsp: `f.base + "/" + f.ext`)
schreibe entsprechende Methode, verstecke
deren Setter/Getter)

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


`edu/ss07/case/programs/refactor/`
Was wurde verbessert? Welche Smells ver

- Nach zweitem Refactoring:
  `http://www.imn.htwk-leipzig.de/~`
  `edu/ss07/case/programs/refactor/`
  Was wurde verbessert? Welche Smells ver

Nichtssagende Nam

(Name drückt Absicht nicht aus)

Symptome:

- besteht aus nur einem oder zwei Zeich
- enthält keine Vokale
- numerierte Namen (`panel1, panel2`
- unübliche Abkürzungen
- irreführende Namen

Behebung: umbenennen, so daß Absicht
(Dazu muß diese dem Programmierer se
Werkzeugunterstützung!

Programmtext

- Kommentare

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

- komplizierte Boolesche Ausdrücke

  → umschreiben mit Verzweigungen, sinnvo bezeichneten Hilfsvariablen

- Konstanten (*magic numbers*)

  → Namen für Konstanten, Zeichenketten e (I18N)

Mehrfachverzweigunger

Symptom: `switch` wird verwendet

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

Ursache: Objekte der Klasse sind nicht ähnlic
Abhilfe: Kompositum-Muster

```
interface C { void foo (); }
class Foo implements C { void foo
class Bar implements C { void foo
```


Richtig refaktorisieren

- immer erst die Tests schreiben
- Code kritisch lesen (eigenen, fremden), ein Anrüchigkeiten entwickeln (und für perfekte
- jede Faktorisierung hat ein Inverses.

  (neue Methode deklarieren ↔ Methode inli expandieren)

  entscheiden, welche Richtung stimmt!

- Werkzeug-Unterstützung erlernen

Aufgaben zu Refaktori

Refactoring-Unterstützung in Eclipse:

```
package simple;

public class Cube {
    static void main (String [
        System.out.println (3.
        System.out.println (5.
    }
}
```

extract local variable, extract method, ad

Behebung: umbenennen (wie vorige Folie)

Größe und Komplexität

- Methode enthält zuviele Anweisungen (Zeil
- Klasse enthält zuviele Attribute
- Klasse enthält zuviele Methoden

Aufgabe: welche Refaktorisierungen?

null-Objekte

Symptom: `null` (in Java) bzw. 0 (in C++) bez besonderes Objekt einer Klasse, z. B. den le oder die leere Zeichenkette
Ursache: man wollte Platz sparen oder „Kom vermeiden.
Nachteil: `null` bzw. `*0` haben keine Methode
Abhilfe: ein extra Null-Objekt deklarieren, das der Klasse gehört.

Aufgaben zu Refaktori

- Code Smell Cheat Sheet (Joshua Kerie `http://industriallogic.com/pa smellstorefactorings.pdf`
- Smell-Beispiele `http://www.imn.ht de/~waldmann/edu/ss05/case/rw` Refactoring Workbook von William C. W `http://www.xp123.com/rwb/`)

  ch6-properties, ch6-template, ch14-ttt

Class Design

Klassen-Entwurf

- benutze Klassen! (sonst: primitive obsessio
- ordne Attribute und Methoden richtig zu

  (Refactoring: move method, usw.)
- dokumentiere Invarianten für Objekte, Kont Methoden
- stelle Beziehungen zwischen Klassen durc dar

  (. . . Entwurfsmuster)

Immutability

- immutable Objekte können mehrfach benut (sharing).

  (statt Konstruktor: statische Fabrikmethode Beispiele in Java-Bibliothek)
- auch die Attribute der immutable Objekte k nachgenutzt werden (keine Kopie nötig)

  (Beispiel: negate für BigInteger)
- immutable Objekte sind sehr gute Attribute Objekte:

  weil sie sich nicht ändern, kann man die In Objektes leicht garantieren

Vererbung bricht Kapselu

Joshua Bloch (Effective Java):

- design and document for in heritance
- . . . or else prohibit it

API-Beschreibung muß Teile der Implementie dokumentieren (welche Methoden rufen sich auf), damit man diese sicher überschreiben k (Das ist ganz furchtbar.)

statt Vererbung: benutze Komposition (Wrapp Delegation.

Abstand v. Dokumenation

. . . je größer, desto gefährlicher! (d. h. int Dokumentation ist noch relativ sicher)
wenn möglich, externe Dokumente darau Werkzeuge generieren.
warum nur unsicher: der Compiler kann z prüfen und ausführen, aber nicht die Kom Ideal: schreibe *selbst-dokumentierenden*

Aufgaben zu Refaktoring (

- Eclipse → Refactor → Extract Interface
- “Create Factory”
- Finde Beispiel für “Use Supertype”

Immutability

(Joshua Bloch: Effective Java, Addison Wesl
immutable = unveränderlich
Beispiele: String, Integer, BigInteger

- keine Set-Methoden
- keine überschreibbaren Methoden
- alle Attribute privat
- alle Attribute final

leichter zu entwerfen, zu implementieren, zu

Vererbung bricht Kapselu

(Implementierungs-Vererbung: bad,
Schnittstellen-Vererbung: good.)

Problem: `class B extends A` ⇒
B hängt ab von Implementations-Details von

⇒ Wenn man nur A ändert, kann B kaputtgel

(Beispiel)

Code- und Interface-Doku

Code dokumentiere

warum?

- nach innen (Implementierung)

  für Wartung, Weiterentwicklung
- nach außen (Schnittstelle)

  soll ausreichen für Benutzung

wo?

- intern (im Quelltext selbst)
- extern (separates Dokument)

(Literatur: Steve McConnell, Code Compl

Selbst-dok. Code: Method

- hat wohlbestimmten Zweck?
- Name beschreibt Zweck?
- ist weit genug zerlegt?

Selbst-dok. Code: Datenorgan

- zur Verdeutlichung zusätzliche Variablen (K
- Benutzungen einer Variablen stehen eng b
- Komplexe Daten nur durch Zugriffsfunktion

Selbst-dok. Code: Desig

- versteht man den Code?
- ist er frei von Tricks?
- Details soweit wie möglich versteckt?
- benutzte Begriffe stammen aus Anwendung nicht aus Informatik oder Programmiersprac

Selbst-dok. Code: War

Stelle dir beim Programmieren vor, daß n gewalttätiger Psychopath mit deinem Cod auch weiß, wo du wohnst.
(anonym, zitiert in McDonnell)

Selbst-dok. Code: Klasse

- Schnittstelle ist konsistente Abstraktion?
- Name beschreibt Zweck?
- Benutzung der Schnittstelle offensichtlich?
- Black Box?

Selbst-dok. Code: Daten

(Attribute, Variablen)

- Name beschreibt Zweck?
- nur zu einem Zweck benutzt?
- Aufzählungstypen anstelle von Flags oder Z
- Benannte Konstanten anstelle magischer Z

Selbst-dok. Code: Ablau

- normaler Ausführungsweg ist deutlich?
- zusammengehörende Anweisungen stehen
- gruppenweise unabhängige Anweisungen i Unterprogramme?
- Normalfall bei Verzweigung nach if (nicht na
- jede Schleife hat nur einen Zweck?
- nicht zu tief geschachtelt?
- Boolesche Ausdrücke vereinfacht?

Kommentare

- Wiederholung des Codes (unsinnig und

  (. . . debug only the code, not the comm can be terribly misleading.)
- Erklärung des Codes

  (. . . don't *document* bad code — *rewrite*
- Markierung im Code (TODO, FIXME, us Eclipse unterstützt)
- Zusammenfassung des Codes
- Absichtserklärung

JavaDoc

http://java.sun.com/j2se/javadoc/writingdoccomments/index.html

```
/**
 * Returns an Image object that can ther
 * The url argument must specify an absc
 * argument is a specifier that is relat
 *
 * @param  url  an absolute URL giving t
 * @param  name the location of the imag
 * @return      the image at the specifi
 * @see         Image
 */
 public Image getImage(URL url, String r
```


Doxygen

http://doxygen.org/

```
export PATH=/home/waldmann/built/

doxygen -g dox.conf # erzeugt Def
emacs dox.conf # Parameter einste
    # PROJECT_NAME, OPTIMIZE_OUTP
    # INPUT, FILE_PATTERNS, RECUR
doxygen dox.conf # Dokumente hers
```


Das Haltepoblem

- *Definition:* H alle $(x, y) \in \mathbb{N}^2$ mit der Eigens Turingmaschine Nr. x führt bei Eingabe y n viele Schritte aus
- *Satz:* (Church, Turing, 193*): Die Menge H entscheidbar.

Das ist kein Fehler der Turingmaschinen, son Aussage gilt für *alle* vernünftigen Berechnung

Automaten und Formale S

Beschreibung von Prozessen durch Meng von Ereignissen.
Einfachste Prozesse:

- kein Ereignis ϵ
- ein Ereignis a

Kombination von Prozessen:

- sequentiell
- iterativ
- alternativ (entweder — oder)
- parallel (verschränkt gleichzeitig)

Schnittstellen-Dokumentat

Zur Benutzung des Codes (einer Klasse/Metl Kenntnis der *Schnittstelle* ausreichen.
Diese muß den *Kontrakt* dokumentieren.
wesentlicher Teil des Kontraktes ist der Typ (Typen von Argumenten und Resultat)
man kann aber nicht alles (*) durch statische ausdrücken, deswegen Beschreibung hinzufü

(*) aber doch sehr viel. Wenn nicht, liegt das primitive obsession, Datenklumpen, lange Pa

Übung Javadoc

- in Eclipse unterstützt (Project → Generate Window → Show View → JavaDoc)
- Probieren Sie JavaDoc-Annotationen aus! Tippe `@` und danach CTRL-SPACE = auto-vervollständigen),
- generierte HTML-Dateien exportieren (Exp System) und in HTML-Browser betrachten

ähnliche Werkzeuge für andere Sprachen

Theorie und Praxis

„Theoretische“ Informatik u Softwaretechnik

- Berechenbarkeit: Halteproblem
- Automaten und formale Sprachen: reguläre
- Komplexitätstheorie: Scheduling-Aufgaben

Das Haltepoblem (Folger

- alle interessanten Programmeigenscha unentscheidbar
- Automatische Quelltextanalyse ist *unm*
- Programmierer muß die interessanten Programmeigenschaften (Korrektheit, L beweisen.
- dafür gibt es Sprach- und Werkzeugunt die wesentliche Arbeit bleibt

Scheduling-Probleme

Die Termine für die Vorgänge sind so zu plan

- die Abhängigkeiten (siehe Netzplan) erfülle
- mit vorgegebenen Ressourcen (Mitarbeiter ausgeführt werden können.

Diese Aufgabe erscheint in verschiedensten (Stundenpläne, Raumpläne, Fahrpläne, Betri Multiprozessor-Systeme . . .).

- d. h. es gibt (*) keinen Algorithmus, der in v (polynomialer) Zeit eine optimale Lösung fi
- d. h. man muß Näherungs-Algorithmen finc benutzen

Eine Liste von Scheduling-Aufgaben ist:
`http://www.nada.kth.se/~viggo/pro compendium.html`
Lese-Übung: Erklären Sie Unterschiede zwis Flow und Job Shop Scheduling.

(*) sehr wahrscheinlich – das ist ein “million d `http://www.claymath.org/millenniu`

zusätzliche Eingabe: eine Zahl $T \in \mathbb{N}$
Frage: gibt es einen Plan mit Gesamtlaufzeit

Scheduling-Aufgaben

Scheduling: Ressourcenzuordnung unter Bea Nebenbedingungen
Bsp: Stundenplan, Raumplan, Transportplan, Auslastung von Maschinen, Mitarbeitern
verschiedene mathematische Modelle,
u. a. Bin Packing

Komplexität von Scheduling-Pro

Mathematisch gehört Ressourcen-Scheduling Graphentheorie/Optimierung (siehe auch ent Lehrveranstaltungen)
Die algorithmische Komplexität ist gut untersu meisten interessanten Varianten gilt aber:

- die Aufgabe ist NP-vollständig

  (N: es ist ein Suchproblem, P: der Suchbau polynomiell tief, d. h. exponentiell breit)

  NP ist *nicht* die Abkürzung für „nicht polync die Tiefe der Suchbäume ist eben *doch* pol beschränkt! (N bedeutet „nicht- deterministi wäre der Baum ein Pfad)

Open-Shop Scheduling

als Optimierungsproblem:

- *Eingabe*: Anzahl $m \in \mathbb{N}$ von Prozessoren, M Jobs, jedes $j \in J$ besteht aus m Operation jedes $o_{i,j}$ eine Dauer $l_{i,j}$,
- *Lösung*: ein *Open-Shop-Plan* für J, d. h. fü Prozessor i eine Funktion $f_i : J \to \mathbb{N}$, so da $f_i(j) > f_i(j') \Rightarrow f_i(j) \geq f_i(j') + l_{i,j'}$ und für je halboffenen Intervalle $[f_i(j), f_i(j) + l_{i,j})$ sinc
- *Kriterium*: möglichst geringe Gesamtlaufzei $\max_{1 \leq i \leq m, j \in J} f_i(j) + l_{i,j}$

Als Entscheidungsproblem:

Scheduling-Aufgaben

die meisten dieser Aufgaben sind schwer

- genauer: NP-vollständig.
- d. h. (wahrscheinlich) exponentielle Lau

man hätte gern Näherungsverfahren

- in Polynomialzeit
- mit beschränktem Fehler

Bsp. Bin-Packing: First Fit, First Fit Decre