

Softwaretechnik

## Kapitel 6

1. Wieso Software Engineering?
2. Vom Einzelkämpfer zum Großprojekt
- Systematische Softwareentwicklung**
3. Anforderungen und Test: Basis des Projekts
4. Entwurf: Strukturen und nicht-funktionale Eigenschaften
5. Entwürfe notieren mit UML: Modelle im SE
- 6. Design Patterns: Entwurfserfahrungen nutzen**
7. Management von Technik und Projekt

Leibniz Universität Hannover
SWT 2015/16 231

Softwaretechnik

## Inhalt von Kapitel 6

**Systematische Softwareentwicklung**

### 6. Design Patterns: Entwurfserfahrungen nutzen

- Idee der Design Patterns
- Beispiel „Adapter“-Pattern
- Beispiel „Composite“-Pattern
- Beispiel „Observer“-Pattern
- Model-View-Controller

Leibniz Universität Hannover
SWT 2015/16 232

Softwaretechnik

## Design Patterns

- Prinzip: Entwurfserfahrungen verpacken, wiederholt nutzen
- Ansatz
  - Gewisse Probleme im Entwurf tauchen immer wieder auf
  - Erfahrene Entwickler finden auch ähnliche, bewährte Lösungen
  - Idee: **Problem-Lösungspaar** wird als Muster beschrieben
  - Wenn das Problem noch einmal auftaucht, gleich die Lösung einsetzen
- Vorteile
  - Erfahrung sind in Patterns „codiert“, nicht im Hirn
  - Anwendung auch für nicht so erfahrene Entwickler möglich
  - Nur das beste Muster muss man sich merken (Auslese)
- Hinweis: Muster sind meist abstrakter als Code
  - Daher eher UML-Muster als Code. Nur Fragmente von Code.
  - Vorteil: Nicht nur in einer Programmiersprache anwendbar
  - Problem ist nur teil-formal beschrieben, Entwickler muss es erkennen

Kurt Schneider
Leibniz Universität Hannover
SWT 2015/16 233

Softwaretechnik

## Problem-Lösungspaare ein Beispiel

**Pattern-Name**  
**Observer**  
 Type: Behavioral  
 What it is:  
 Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

**Klassifikation**

**Lösung**

Kurt Schneider
Leibniz Universität Hannover
SWT 2015/16 234

Softwaretechnik

## Flexibilität durch Adapter: Beispiel Wetterstation

- Messstationen in vielen Städten
- Werden automatisiert abgefragt
  - Temperatur, Luftdruck etc.
  - Explizites Logging
  - Zeichnen Verlaufskurven auf

Probleme treten auf, als USA dazukommt ...

Software zum Werteabruf:

```

public class Thermometer {
    ...
    public double temperature () { ... }
    public void logNow (Time now) {...}
}
          
```

Kurt Schneider
Leibniz Universität Hannover
SWT 2015/16 235

Softwaretechnik

## Problem: Wie vereinheitlichen?

**Messzentrale will u.a.:**

- Einzelne Temperaturen abrufen
- Alle Temperaturen abrufen
- Alle Stationen zum Loggen auffordern

**Und insbesondere**

- Dabei nicht zwischen D- und US-Stationen unterscheiden müssen
- Bei nächster Anwendung nicht wieder drüber nachdenken müssen, wie man Fahrenheit umrechnet.

Kurt Schneider
Leibniz Universität Hannover
SWT 2015/16 236

### Lösungsidee: Pattern "Adapter"

**Problem**  
Client möchte von einer Klasse eine Methode benutzen, die diese nicht anbietet.

**Lösung: Adapter Klassen-Adapter (li)**

- Gibt einer Klasse ein neues Methoden-Interface
- Erbt/implementiert Interface von Target, erbt echte Methoden von Adaptee

Blauer Pfeil: kein UML

**Objekt-Adapter (re)**

- Erbt nicht von Adaptee
- Benutzt aber adaptee-Objekte, um Request () doch noch auszuführen

Kurt Schneider Nach Gamma et al. (1994) Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley  
Leibniz Universität Hannover SWT 2015/16 237

### Wo soll Pattern "Adapter" eingesetzt werden?

**Lösung im Beispiel: Objekt-Adapter geplant**

Vorbild: Das Pattern

Kurt Schneider Nach Gamma et al. (1994) Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley  
Leibniz Universität Hannover SWT 2015/16 238

### Pattern "Adapter" im Beispiel in UML umsetzen

Struktur in UML umgesetzt

Zum Vergleich: Das Pattern

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 239

### Pattern "Adapter" fertig eingesetzt

Fertig: Einsatz des Patterns umgesetzt, und mit UML modelliert

Kurt Schneider Nach Gamma et al. (1994) Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley  
Leibniz Universität Hannover SWT 2015/16 240

### Nächstes Pattern: "Composite"

Einheitliche Handhabung, bessere Flexibilität und Änderbarkeit

**Problem**  
Objekte werden möglicherweise gruppiert. Clients, die sie benutzen, sollen aber nicht zwischen atomaren und zusammengesetzten („composite“) Objekten unterscheiden müssen.

**Lösung: Composite**

- Beispiel: graphischer Editor
- Gruppierung wie Linien oder Rechtecke
- Alle kann man löschen, dehnen, färben auf gleiche Weise „Operation()“

Kurt Schneider Nach Gamma et al. (1994) Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley  
Leibniz Universität Hannover SWT 2015/16 241

### Beispiel: Drucken von Dateien und Ordern

- Idee: Befehl „Drucken“ an Ordner wie an Einzeldateien
- Ordner geben den Befehl an ihre Inhalte weiter – und rekursiv weiter

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 242

### Beispiel: Ordner drucken ohne Composite

Naiver Ansatz

```
public class MausMenu {
    public static void main(String[] a) {
        Ordner o1=new Ordner();
        Ordner o2=new Ordner();

        o1.add(new WordDatei());
        o1.add(new WordDatei());
        o1.add(new PdfDatei());

        o2.add(new PdfDatei());
        o2.add(o1);
    }
}
```

**Ordnerstruktur**

Menü will o2 auf „naive Weise“ drucken:  
Alle Elemente von o2 durchgehen  
WENN sie pdf oder Word sind: drucken  
SONST alle Elemente davon durchgehen  
WENN sie pdf oder Word sind: ...

Schon besser: Rekursiv  
**SCHLECHT: bei JEDEM AUFRUF nötig!**  
Aufwändig, fehleranfällig, änderungsfeindlich

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 243

### Beispiel: Ordner drucken

Das möchte man tun können

```
public class MausMenu {
    public static void main(String[] a) {
        Ordner o1=new Ordner();
        Ordner o2=new Ordner();

        o1.add(new WordDatei());
        o1.add(new WordDatei());
        o1.add(new PdfDatei());

        o2.add(new PdfDatei());
        o2.add(o1);

        o2.drucken();
    }
}
```

**Einfache Ordnerstruktur aufbauen**

**Gesamte Struktur drucken (wie eine Einzeldatei)**

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 244

### Beispiel: Ordner drucken

Die Bestandteile vorher und nachher

```
public class WordDatei implements Komponente {
    public void drucken () {
        System.out.println("WordDatei wird ausgedruckt.");
    }
}

public class PdfDatei implements Komponente {
    public void drucken () {
        System.out.println("Pdf-Datei wird ausgedruckt.");
    }
}
```

```
public interface Komponente {
    public void drucken();
}
```

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 245

### Beispiel: Ordner drucken

Bindeglied: Composite/Ordner

```
import java.util.LinkedList;
import java.util.ListIterator;

public class Ordner implements Komponente {
    LinkedList inhalt = new LinkedList();

    public void add(Komponente k) {
        inhalt.add(k);
    }

    public void drucken() {
        Komponente komp;
        for (ListIterator it=inhalt.listIterator(); it.hasNext(); ) {
            komp = (Komponente) it.next();
            komp.drucken();
        }
    }
}
```

**Der Witz:**  
Iteration im Composite (zusammenges. Objekt) statt im Aufrufer

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 246

### Patterneinsatz dokumentieren

Beispiel, ähnlich UML

**Erinnerung**

• Wie wurde Pattern umgesetzt?  
– Konkrete Klassen u. ihre Rollen

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 247

### Pattern "Observer"

Erhöht die Flexibilität, entkoppelt Systemteile

**Problem:** Wenn sich der Zustand eines bestimmten Objekts („Subject“) ändert, soll das zu Reaktionen in anderen Objekten führen. Diese Beziehung soll flexibel sein.

**Lösung:** Observer

**Sequenzdiagramm**

Nach Gamma et al. (1994) Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley  
Leibniz Universität Hannover

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 248

### Beispiel für Patterns Börse

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    249

### Beispiel: Börsenkurse

Ein Kurs, drei Sichten:  
zwei Observer,  
ein Drucker

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    250

### Beispiel: Börsenkurse Ausgangssituation

```

public class Stock {
    private String name;
    private int price; // in cents

    public Stock(String name) {
        this.name = name;
    }

    public String toString() {
        return this.getName() + " (" +
            + String.format("%.2f", this.getPrice() / 100.0) + ")";
    }

    public String getName() { return name; }
    public int getPrice() { return price; }

    public void setPrice(int price) {
        if (price >= 0) {
            this.price = price;
        }
    }
}

```

**Aufruf**  
**Drucker**: `printer1.print(MainFrame.this.getSelectedStock().toString());`  
**MainFrame**: `MainFrame.this.getSelectedStock().setPrice(4711);`

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    251

### Börse, nun mit Observer-Pattern

```

import java.util.Observable;
public class Stock extends Observable {
    private String name;
    private int price; // in cents

    public Stock(String name) {
        this.name = name;
    }

    public String toString() {
        return this.getName() + " (" +
            + String.format("%.2f", this.getPrice() / 100.0) + ")";
    }

    public String getName() { return name; }
    public int getPrice() { return price; }

    public void setPrice(int price) {
        if (this.price >= 0) {
            this.price = price;
            this.setChanged();
            this.notifyObservers();
        }
    }
}

```

**Erbt von Java-Klasse Observable**  
 Kann damit alles, was Observable kann

**Insbesondere:**

1. sich bei Änderungen selbst benachrichtigen
2. `notifyObservers()` meldet an alle Observer weiter
3. Dann müssen sich Observer aktualisieren (`update()`)

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    252

### Observer an- und abmelden beim Observable; der verwaltet sie (geerbt)

Jeder Observer trägt sich beim Observable ein

```

public GraphComponent(Stock stock) {
    super();
    this.stock = stock;
    stock.addObserver(this);
}

public StockQuoteFrame(Stock stock) {
    super();
    initialize();
    this.stockNameLabel.setText(stock.getName());
    this.stockQuoteLabel.setText("+" + stock.getPrice() / 100);
    this.stock = stock;
    this.stock.addObserver(this);
}

```

**Nicht vergessen: nicht mehr nötige abhängen!**

```

public void dispose() {
    super.dispose();
    this.stock.deleteObserver(this);
}

```

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    253

### Was tut „notifyObservers“?

Ganz einfach: schickt an alle eingetragenen Observer eine „update“-Anforderung

```

public void notifyObservers() {
    Observer[] allObservers;

    for (ListIterator it=allObservers.listIterator();
         it.hasNext(); ) {
        obs=it.next();
        obs.update();
    }
}

```

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16    254

### Beispiel: Börsenkurse Die Beobachter

```

public void update(Observable o, Object arg) {
    if (this.stock == o) {
        this.stockQuoteLabel.setText(""+this.stock.getPrice()/100);
    }
}

public void update(Observable observable, Object param) {
    if (observable == this.stock) {
        this.addValue(this.stock.getPrice());
        this.repaint();
    }
}

```

Aktie A 98

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 255

### Patterneinsatz dokumentieren Beispiel, ähnlich UML

• Pattern-Symbol:  
gestricheltes Oval mit Patternnamen; Pfeile mit Rollennamen

- Meist in Kollaborationsdiagramm
- Oder – wie hier – in Klassendiagramm
- Zusätzliche Text-Info ist meist hilfreich (Begründung, Methoden etc.)

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 256

### Methode im Zusammenhang Patterns verwenden

**WENN** eines der Patterns mit seinem Problem-Teil zu Ihrem Problem passt  
**DANN** überlegen Sie, ob Sie es einbauen wollen  
**WENN** es sich zu lohnen scheint, **DANN** bauen Sie es ein  
**SONST** nicht!

- Geeignete Patterns finden: Im Entwurf**
  - Sie sollten einige Patterns auswendig kennen: im Geiste durchgehen
  - Haben Sie ein wichtiges Problem, lesen Sie Problemtile weiterer Patterns durch
- Anwendbarkeit prüfen**
  - Passt der „Problem“-Teil?
  - Wie wichtig ist Ihnen die angestrebte Eigenschaft (oft Flexibilität)?
  - Abwägen und dokumentieren von Nutzen und Aufwand
- Einsatzort festlegen**
  - Genau festlegen, welche konkrete Klasse o.ä. zu welcher Rolle im Pattern passt
- Pattern-Lösungsteil einfügen**
  - In UML-Entwurf, evtl. auch in schon existierenden Code
- In der Regel erst Pattern aussuchen, dann programmieren**

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 257

### Sehr nützlich: MVC Model View Controller

**Problem**  
Ein System mit komplizierter Oberfläche soll gebaut werden, bei dem sich die Interna und die Oberfläche unabhängig flexibel ändern lassen.

**3-Schichten/3-Tier**

**Überblick**

- **WENN** Sie ein Programm entwickeln
  - Mit intensiver Logik/Datenhaltung
  - UND nicht-trivialer Oberfläche
- **DANN** trennen Sie Oberfläche vom Rest  
UND verwenden Sie dabei das MVC-Pattern
- **Ziel:** Entkoppelung zwischen Inhalt (Model) und Oberfläche (View-Controller)
  - Gibt genaue Interaktion vor
  - Damit kann man fast jede Komponente separat ändern (information hiding, divide et impera)

Vgl.: Zuser et al. (2004): Software Engineering mit UML und dem Unified Process. Pearson Studium

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 258

### Lösung: MVC im Detail

**Zuvor**

- MVC aufbauen
- Vs sind Observer (Pattern im Pattern)
- V fragt M nach Werten, stellt sie dar (Ausgangsp.)

**Bsp. Bankomat**

1. Benutzer gibt 4 Ziffern ein, „ok“
2. V informiert C über Eingabe
3. C teilt M die Ziffern als PIN mit
4. M prüft PIN, lässt Benutzer ein
5. M informiert V
6. V fragt nach Kontostand, stellt ihn dar

Zugrunde liegendes Bild: Sun (2002)  
<http://java.sun.com/blueprints/patterns/MVC-detailed.html>

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 259

### Beispiel für MVC im Detail

**Zuvor**

- Zwei Sichten haben sich bei Dateien (z.B. Einführungsveranstaltung-v03.pdf) angemeldet

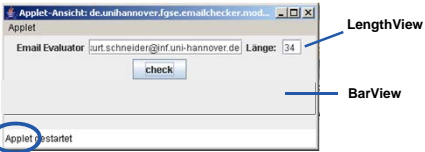
**Interaktion**

- Benutzer ändert Dateinamen:

1. Gibt String ein in C
2. C teilt File M neuen Namen mit
3. File M übernimmt Namen
4. File informiert seine Observer (V1, V2) über Änderung
5. V1, V2 fragen nach Namen, stellen ihn (verschieden) dar

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 260

### Java-Beispiel



- Emailchecker evaluiert Email-Adressen
  - Wie lang sind sie? → LengthView
  - Sind sie zu lang? → BarView
- Ziel dieses Beispiels: „reines MVC“
  - Echtes Java-Beispiel
  - Nutzt MVC in Java
  - Sonst nichts, so klein und simpel wie möglich

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 261

### Relevanter Code

Die Anwendung als Applet, MVC-Teile fett

```
public class EmailCheckerApplet extends JApplet {
    EmailEvaluator model = new EmailEvaluator();
    LengthView numView = new LengthView(model); // Zahl
    BarView barView = new BarView(model); // Balken

    public void init(){
        this.setSize(400,140);
        GridLayout gridBag = new GridLayout(2,1);
        getContentPane().setLayout(gridBag);

        // Views in die GridBag einfügen
        getContentPane().add(numView);
        getContentPane().add(barView);
    }
}
```

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 262

### Relevanter Code

#### Model

```
public class EmailEvaluator extends Observable {
    private String email="";

    public void setEmail(String emailAddress){
        email = emailAddress; // Model aendert sich

        setChanged(); // Auf Aenderung hinweisen
        notifyObservers();
    }
}
```

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 263

### Relevanter Code

#### LengthView

```
public LengthView(EmailEvaluator anEvaluator){
    this(); // baut GUI-Elemente zusammen

    model = anEvaluator;
    model.addObserver(this);
    evalController = new EmailEvalController(model);
    setPreferredSize(new Dimension(350, 100));
}

public void update(Observable obs, Object obj) {
    length.setText(String.valueOf(model.getLength()));
    repaint();
}
```

LengthView weiß selbst  
- interessanten Aspekt in Model  
- wie holen  
- wie darstellen

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 264

### Relevanter Code

#### BarView

```
public class BarView extends JPanel implements Observer{
    private EmailEvaluator model;

    public BarView(EmailEvaluator aModel) {
        super();
        model = aModel;
        model.addObserver(this); // in MVC einbauen
    }

    public void update(Observable ignored, Object alsoIgnored) {
        repaint(); // ruft paint() auf, aktualisiert Balken
    }

    public void paint(Graphics g){
        super.paint(g);
        ...
        // Farben markieren, ob die Länge akzeptabel ist
        if (numOfChars > 23){
            gc.setColor(Color.RED);
            ...
            gc.fillRect(5, barLength, 30); // Email-Balken
        }
    }
}
```

Die Grenzen (23) gehören eigentlich zur „Logik“ und damit zum Model.

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 265

### MVC besser umgesetzt

Model entscheidet, BarView stellt dar

Das Model: Semantik/Logik	Die View: Darstellung
<pre>public class EmailEvaluator extends Observable {     ...     public boolean isTooLong (int chars){         return (chars &gt; 23;     }      public boolean isPerfect (int chars){         return (chars &lt; 15;     } }</pre>	<pre>public class BarView extends JPanel implements Observer{     ...     if (model.isTooLong(numOfChars)){         gc.setColor(Color.RED);     } else     if (model.isPerfect(numOfChars))     {         gc.setColor(Color.GREEN);     } else {         gc.setColor(Color.YELLOW);     } }</pre>

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 266



### Controller und Listener in Java

```

sendComButton.addActionListener (
    new ActionListener(){
        public void actionPerformed(
           (ActionEvent buttonPressedEvent){
                try {
                    // Wenn Button gedrückt: evalController soll
                    // Emailadresse holen, Daten damit aktualisieren
                    evalController.updateRawData(emailAdr.getText());
                }
                catch(RuntimeException e) {
                    System.out.println("Fehler mit dem Button");
                }
            }
        );
    }

```

Hier wird ein Listener erzeugt, der den sendComButton belauscht.

Listener sind wie Observer auf GUI-Elementen in der View: Sie „lauschen“, was dort geschieht, darauf reagiert der Controller

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 267

### Wichtige Variante: MVC im Web

- Beispiel: Play!-Framework
  - Folgt MVC
  - Aber modifiziert
- Kein Observer V→M möglich
  - V bemerkt M-Änderung nicht
- Ähnliche Umsetzung
  - HTTP-Request löst aus
  - Framework sucht zuständigen C
  - Controller vermittelt an M, Vs
  - Antwortet: HTTP Response
- Das ist üblich und ok!

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 268

### Methode im Zusammenhang

#### MVC in die Architektur einbetten

WENN Sie in eine nicht-triviale GUI und mindestens einen der folgenden Aspekte haben:

- Nicht-triviale Logik
- Datenverwaltung/Verteilung

DANN verwenden Sie MVC zur Entkoppelung

- Legen Sie dabei View und Controller in eine Schicht (die oberste)
  - Zunächst: jede Schicht ein Java-Package
  - Darunter: Schicht bei Bedarf noch in feinere Pakete zerlegen
- View und Controller *möglichst* in unabhängige Klassen u. Pakete
  - Aber gerade VC lassen sich nicht immer wirklich trennen
  - In Java arbeiten Sie mit Listenern für Controller; das geht
  - In C# oder Visual Basic ist die Trennung schwerer möglich
  - Andere Sprachen sind bei VC oft auch „eigenwillig“
- Legen Sie das Model in eine andere Schicht
  - Model muss von VC besonders klar getrennt sein, in allen Sprachen
  - Drei-Schichten-Architektur ist ein Spezialfall

Packages (rekursiv) Schicht

Optional: 3. Schicht

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 269

### Es gibt etliche Patterns

Nach Gamma et al. (1994) Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley

- Das Gamma-Buch ist der absolute Klassiker "GoF"
  - Hat eine Disziplin begründet
- Enthält mehr als 20 Patterns
  - Sind Allgemeingut
  - Wenige andere bekannt
  - Jeder kann eigene definieren
- Aber es sind nicht hunderte von Pattern
- Ein "SW-Designer" kann - und sollte! - sie kennen

Kurt Schneider

### Bewertung von Design Patterns

- Größere (und abstraktere) Entwurfsbestandteile als Klassen und Objekte; programmiersprachen-unabhängig
- Patterns sind konstruktive Qualitätssicherung
  - Qualität betroffen (oft: mehr Flexibilität)
  - Rationale/Begründung gehört zum Pattern
  - Erklärungen werden kompakter mit Patterns
  - Effiziente Kommunikation unter Experten möglich
  - Davon profitieren Lesbarkeit, Flexibilität, Wartbarkeit
  - Ein Pattern-Katalog ist wie ein Musterbuch für Architekten
- Sind in der OO-Zunft bekannt und werden vorausgesetzt
  - Fast jeder Entwickler hat davon gehört
  - Profis können sie selbst einsetzen

Kurt Schneider Leibniz Universität Hannover SWT 2015/16 271