

Softwarearchitektur

(Architektur: *αρχή* = Anfang, Ursprung + *tectum* = Haus, Dach)

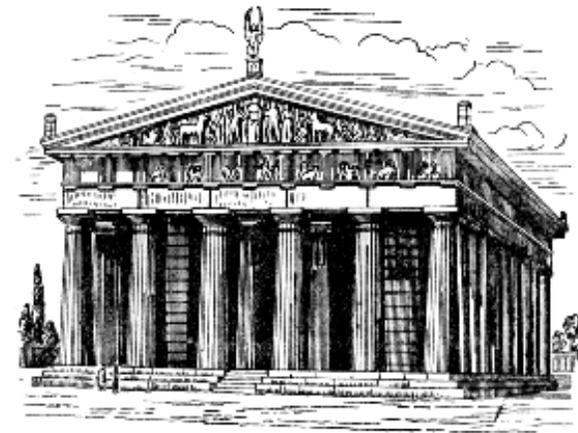
6. Ausprägungen und Wiederverwendung I

Vorlesung Wintersemester 2008 / 09

Technische Universität München

Institut für Informatik

Lehrstuhl von Prof. Dr. Manfred Broy



Dr. Klaus Bergner, Prof. Dr. Manfred Broy, Dr. Marc Sihling

Inhalt

- Wiederverwendung
- Schnittstellen
- Frameworks
- Aspect-oriented Programming
- Zusammenfassung

- *Nächste Stunde: Wiederverwendung per*
 - *Komponenten*
 - *Architekturmuster*
 - *Referenzarchitektur*
 - *Produktlinienarchitektur*

Inhalt

- Wiederverwendung
- Schnittstellen
- Frameworks
- Aspect-oriented Programming
- Zusammenfassung

- *Nächste Stunde: Wiederverwendung per*
 - *Komponenten*
 - *Architekturmuster*
 - *Referenzarchitektur*
 - *Produktlinienarchitektur*

Wiederverwendung als Weg aus der Softwarekrise?

- Seit vielen Jahren sieht man in der Wiederverwendung von Software-Artefakten die Lösung zu vielen Problemen der Softwareentwicklung
 - bessere Qualität durch Nutzung eines „gereiften“ Produkts
 - geringere Kosten durch mehrfache Verwendung
 - kürzere Entwicklungszeiten aufgrund weniger Eigenentwicklung
- Beispiel Kostenreduktion:

Reuse rates	0%	25%	50%
Time [m.m.]	81,5	45	32
# Developers	8	6	5
Costs per l.o.c. [DM]	70	37	28
Lines per m.m.	165	263	370
Savings [DM]	-	~ 400.000	~ 560.000
Savings	-	~ 45%	~ 60%

From: C. Jones, *The Impact of Reusable Modules and Functions*, in *Programming Productivity*, Mc Graw Hill, 1986

Warum Wiederverwendung oft nicht klappt...

- Technische Probleme
 - Wiederverwendung oft nur auf Code-Ebene
 - Vernachlässigte Dokumentation und Spezifikationen
 - Korrektheit der wiederverwendeten Teile nicht gesichert
- Organisatorische Probleme
 - Wiederverwendung selten geplant, zu spät gemacht
 - Motivation und Anreize für Wiederverwendung
 - Kaum Handelsplätze für Software, Anbieter und Nutzer finden nur schwer zusammen
 - Aufwand für Wiederverwendung u. U. höher als Nutzen



Formen der Wiederverwendung

- **Ad Hoc Wiederverwendung** : Ursprünglich nicht für Wiederverwendung vorgesehene und nicht dafür vorbereitete Artefakte werden wiederverwendet
- **Systematische Wiederverwendung**: Artefakte werden systematisch für Wieder-/Mehrfachverwendung aufbereitet

Formen der systematischen Wiederverwendung

- **Produktlösungen:** Es wird ein Produkt geschaffen, das über Parametrisierung angepasst werden und so ohne Änderung des Codes in vielen Einsatzumgebungen und Unternehmen eingesetzt werden kann (Beispiele: div. Produkte von SAP, Microsoft, Entwicklungswerkzeuge von IBM, etc.).
- **Produktlinien:** Hier wird ein Baukasten entwickelt, der durch Kombination Lösungen nach einheitlicher Architektur zu bauen erlaubt, wobei kaum Änderungen im Code erforderlich sind (oft werden Techniken der Codegenerierung eingesetzt – eine weitere Richtung sind domänenspezifische Sprachen).
- **Standardarchitekturen:** Hier wird ein einheitlicher Architekturansatz geschaffen (oft Referenzarchitektur genannt), bei dem bestimmte Teile (Komponenten) mehrfach (einheitlich) verwendbar sind und andere Teile individuell entwickelt werden.
- **Componentware:** Es werden Softwarekomponenten geschaffen, die ohne Änderung des Codes als Komponenten von individuell entwickelten Softwaresystemen eingesetzt werden.
- **Frameworks:** Frameworks sind meist objektorientiert und liefern eine Zahl von Klassen, die teilweise unvollständig sind, und aus denen durch Ergänzung und Änderung Applikationen aufgebaut werden können. Man unterscheidet Glass Box Frameworks (auch White Box Frameworks) und Black Box Frameworks. Bei Black Box Frameworks werden die Klassen über Schnittstellen spezifiziert und wie Komponenten eingesetzt, ohne dass ihr innerer Aufbau bekannt und verstanden sein muss. Bei Glass Box Frameworks werden Klassen angepasst und erweitert (etwa durch Vererbung), wobei der Code explizit betrachtet wird. Eine Mischform sind Grey Box Frameworks.

Wo Wiederverwendung erfolgreich ist...

- Erfolgsfaktoren
 - Technische Erfolgsfaktoren
 - präzise definierte, möglichst standardisierte Schnittstellen
 - abgeschottete Implementierung (Geheimnisprinzip)
 - niedrige Einsatzschwelle: wenig Konfiguration, verständliche Dokumentation
 - Organisatorische Erfolgsfaktoren
 - Systematisches Vorgehen bei der Entwicklung und Nutzung
 - Anreize und Motivation für die Wiederverwendung
 - niedrige Einsatzschwelle: Preis, Lizenzmodell
- Gelungene Beispiele für Wiederverwendung
 - Schnittstellen und Protokolle Heutige Vorlesung
 - Bibliotheken und Frameworks (z.B. Java-Bibliotheken)
 - Middleware, Componentware, Container (z.B. COM, J2EE)
 - Architekturmuster, Referenzarchitekturen, Produktlinien-Architekturen

Kommerzieller Marktplatz für Komponenten

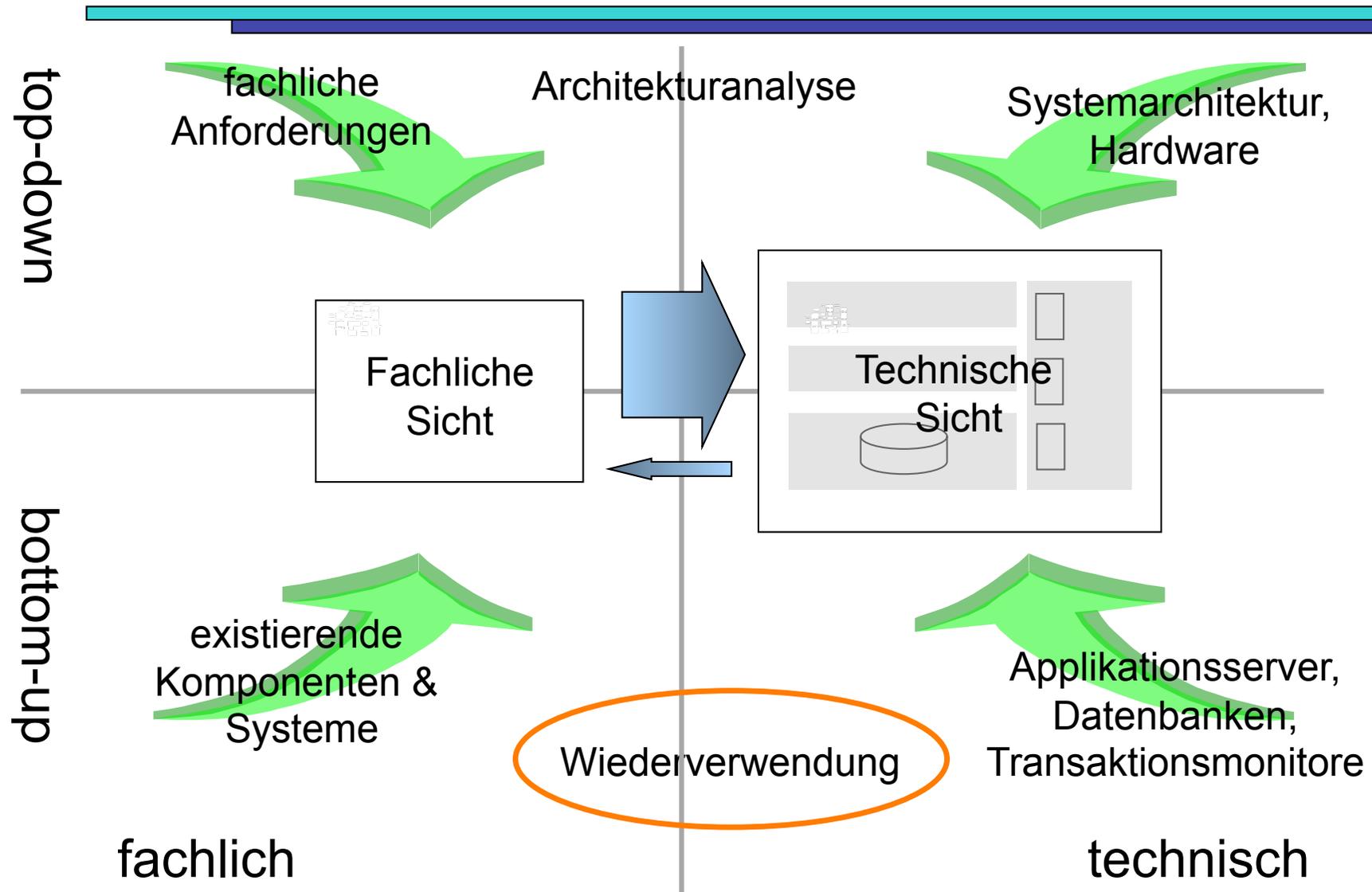
The screenshot displays the ComponentSource website, a marketplace for software components. The page is organized into several sections:

- Navigation:** Includes links for Advanced Search, Shopping Cart, My Account, Contact Us, Logon, Register, and Sign In.
- Header:** Features the ComponentSource logo and a list of supported technologies: .NET, COM, MTS, Java, J2EE, VCL, C++, Tools, and Other.
- Product Search:** A search bar with a dropdown menu set to 'All Products' and a 'Go' button.
- SALES & SUPPORT:** Links for Call Toll Free, Email Sales, and Email Support.
- Featured Products:**
 - LEADTOOLS Raster Imaging Pro:** An API, C++ Class libs, ActiveX and VCL toolkit with 80+ raster file formats (JPEG, GIF, TIFF...), 70+ Image Transforms and Filters, extensive display options, TWAIN scanning, printing, thumbnail browser, internet and database imaging functions and much more.
 - TX Text Control ActiveX:** Build your own word processor. TX Text Control ActiveX gives your application the ability to read, edit, and create documents in Microsoft Word formats including Word 2000.
 - Spread:** Add spreadsheet functionality to your applications. Use FarPoint's Spread 6 to easily incorporate advanced grid and high-level spreadsheet features into your applications.
- What's on the site:**
 - New Visitor:** If you are new to the ComponentSource Web site, the following links may help you find the information you are looking for.
 - Other Sections...:**
 - About Components
 - Build Components
 - News Center
 - SAVE-IT™ Enterprise Reuse
 - Sell Components
- Browse Stores:**
 - All Products:** .NET, .NET Beta, .NET Ready (COM), C++, C++Builder, COM, Component Building Tools, Delphi, DLL & Libraries, J2EE, Java, JavaBeans, JBuilder, Linux, MTS, Tools, VBA, VCL, Visual Basic, Visual Studio .NET, Visual Studio, Windows CE
 - Stop Press:**
 - Get ComponentOne V8 Power!** Give your software projects V8 power from ComponentOne. Now available for immediate download are new Version 8 updates for the following best selling ComponentOne products: True DBGrid Pro 8.0, VSFlexGrid Pro 8.0, VSView 8.0 Classic Edition, VSView 8.0 Reporting Edition, ComponentOne Chart 8.0, ComponentOne WebChart 8.0, ComponentOne Query 8.0, TrueDataControl 8.0 and True DBLIST 8.0
 - New ComponentOne Studio Enterprise** Add grid, reporting, charting, data manipulation and user interface capabilities to your .NET, ASP.NET and ActiveX applications. ComponentOne Studio Enterprise is a combination of three individual ComponentOne Studio subscriptions delivering you 30 components.
 - Top Downloads:**
 - Developer Express XtraGrid .NET Suite** Buy Now | Other Grid Components Add a bound or unbound, native .NET grids to your Visual Studio.NET applications. Developer Express XtraGrid .NET Suite is a 100% C# Grid, CardView, and Editors Library built specifically for Visual Studio .NET.
 - Sax ActiveX SmartUI** Buy Now | Other User Interface Components Build a complete application user interface with a single comprehensive component. Sax ActiveX SmartUI provides all the functionality required for a modern application front-end (Office 2000/Win2000) including: MenuBars, ToolBars, StatusBars, OutlookBars, TreeViews, ListBoxes, OptionLists, PropertyLists, Property Toolboxes, flat Tabstrips and much more.
 - AddFlow** Buy Now | Other Diagramming Components Create interactive flowcharts and workflow diagrams. AddFlow is an ActiveX component that is useful each time you need to display and use relationships between objects in your application.
 - ASPxGrid** Buy Now | Other Grid Components Add Outlook style grids to your ASP.NET applications. The ASPxGrid by Developer Express is a 100% C# based grid control for ASP.NET.
 - RAD XML Persistent Tools** Buy Now | Other Data Storage Components With almost no coding create sophisticated TreeViews, and VB-Like data-entry Property Dialogs, that are XML-ready and can share their data through built-in multi-user XML-Database engine.
 - VARCHART XGantt LC** Buy Now | Other Charting and Graphing Components Add interactive Gantt diagram functionality to your desktop or web application. VARCHART XGantt LC is an ActiveX control which lets you graphically display, edit and print your data for project or resource planning in Gantt diagrams or control panels.
 - DockStudioXP** Buy Now | Other Toolbar Components Provide Visual Studio.NET or Microsoft XP/2000 style toolbars and dockable windows in your applications. DockStudioXP allows you to implement docking flexibility with no complex manual coding.
 - ComponentOne Studio for .NET** Buy Now | Other Button and Cursor Design Components Maintain a complete collection of components for all aspects of application development. ComponentOne Studio for .NET provides existing .NET tools, any new .NET tools and updates released within your 1 year subscription period, plus email support.
 - ImagXpress**
- Sponsored Links:**
 - IP*Winkl .net
 - Contour CUBE
 - Xceed
- Best Sellers:**
 - Janus GridEX
 - InfraLogics NetAdvantage Suite
 - ActiveReports for .NET
 - ActiveReports
 - Janus Controls Suite
- Microsoft .net msdn dev2dev**
- New Products:**
 - PowerTCP SSL Sockets for .NET
 - Transcender WANCert/Security (640-442)

Wiederverwendung von Architekturen

- Beobachtung: mit wachsender Komplexität eines Systems ist das Zusammenspiel der Komponenten kritischer als ihre Funktionalität.
- Die Vision einer „plug-and-play“ Softwareentwicklung kann nur real werden, wenn neben einzelnen Komponenten auch die Organisation und das Zusammenspiel der Komponenten (also die Architektur) einfach wiederverwendet werden können
- Im Bereich Softwarearchitektur ist Wiederverwendung an vielen Stellen wünschenswert:
 - Spezifikation (z.B. Schnittstellen, Protokolle, Contracts, Strukturen etc.)
 - Implementierung (z.B. elementare Komponenten einer Architektur)
 - Konzepte und Zusammenhänge (z.B. Architekturtreiber)
 - Methodik (z.B. „Best Practice“, Cookbook-Ansätze)
- Diese Artefakte verbessern nicht zuletzt die Kommunikation unter den Entwicklern („klassische Drei-Schichten-Architektur“).

Wiederverwendung in fachlicher und technische Sicht



Inhalt

- Wiederverwendung
- **Schnittstellen**
- Frameworks
- Aspect-oriented Programming
- Zusammenfassung

- *Nächste Stunde: Wiederverwendung per*
 - *Komponenten*
 - *Architekturmuster*
 - *Referenzarchitektur*
 - *Produktlinienarchitektur*

Schnittstelle - Definitionen

- „Eine Schnittstelle ist ein **definierter Übergang** zwischen Datenübertragungseinrichtungen, Hardware-Komponenten, logischen Softwareeinheiten oder zwischen Menschen und Computern.“
- „Eine Schnittstelle ist eine **definierte Übergabestelle** innerhalb eines Systems.“
- „Eine Schnittstelle im Sinne der objektorientierten Programmierung ist eine **Sammlung von Operationssignaturen**, welche durch Methoden in unterschiedlichen Klassen implementiert werden können.“
- „Eine Schnittstelle beschreibt einen **Teil des extern sichtbaren Verhaltens** eines Systems.“
- „Eine Schnittstelle beschreibt einen **Typ von Instanzen**, die auf unterschiedliche Art und Weise realisiert sein können.“

Beispiel: List- und Iterator-Schnittstellen in Java (Ausschnitt)

```
public interface List {  
    public boolean add(Object o);  
    public boolean remove(Object o);  
    public boolean contains(Object o);  
    public int indexOf(Object o);  
    public Object get(int index);  
    public Object set(int index, Object element);  
    public int size();  
    public Iterator iterator();  
}
```

veränderbare
Liste von Objekten

```
public interface Iterator {  
    public boolean hasNext();  
    public Object next();  
    public void remove();  
}
```

Durchblättern
einer Liste

Beispiel: Probleme mit diesen Schnittstellen (1)

- Verhalten nicht formal spezifiziert
 - keine Informationen über erlaubte Parameter-Werte
 - keine Informationen über nichtfunktionale Eigenschaften
 - keine Informationen über erlaubte Aufrufreihenfolgen
 - stattdessen informelle Beschreibung in Textform (JavaDoc)
- Nur Spezifikation der exportierten Operationen
 - Keine Import-Spezifikation der benötigten Operationen nicht
- Implementierungen können unvollständig sein
 - müssen nicht sämtliche Operationen unterstützen
 - müssen stattdessen UnsupportedOperationException werfen
- nicht alle Exceptions sind an der Schnittstelle sichtbar

Beispiel: Probleme mit diesen Schnittstellen (2)

Interface Iterator

- public Object **next()** throws
 - NoSuchElementException – if the iteration has no more elements.
- public void **remove()** throws
 - UnsupportedOperationException - if operation is not supported by this Iterator
 - IllegalStateException - if the next method has not yet been called, or the remove method has already been called after the last call to the next method.

Implizites Verhalten / Probleme:

- es bleibt unklar, ob mehrere Iteratoren auf einer Liste möglich sind (ja!)
- nach remove() muss immer next() folgen
- remove-Operation ist optional implementiert
- ConcurrentModificationException
 - wenn die Liste des Iterators während des Durchblätterns verändert wurde, außer durch die remove-Operation des Iterators selbst
 - ob der Iterator im Zustand „Concurrent Modification“ ist, kann beim Iterator nicht einfach überprüft werden (außer über die Exception)

Schnittstellen und Contracts

- Ideale Schnittstellenbeschreibungen enthalten:
 - Signaturen von Operationen, inklusive der Parameter
 - Erbrachte Funktionalität, Vor- und Nachbedingungen
 - Aufrufreihenfolgen, Protokolle
 - Angaben zu Nichtfunktionalen Eigenschaften
 - ggf. zusätzliche Aspekte wie Ansprechpartner, Aufrufkosten etc.
- Beschreibungstechniken
 - Klassendiagramme zur Beschreibung von Parametern
 - Sequenzdiagramme, Aktivitätsdiagramme
 - Formal definierte Pre- und Postconditions

Design by Contract

Analogie zu Verträgen im juristischen Sinn

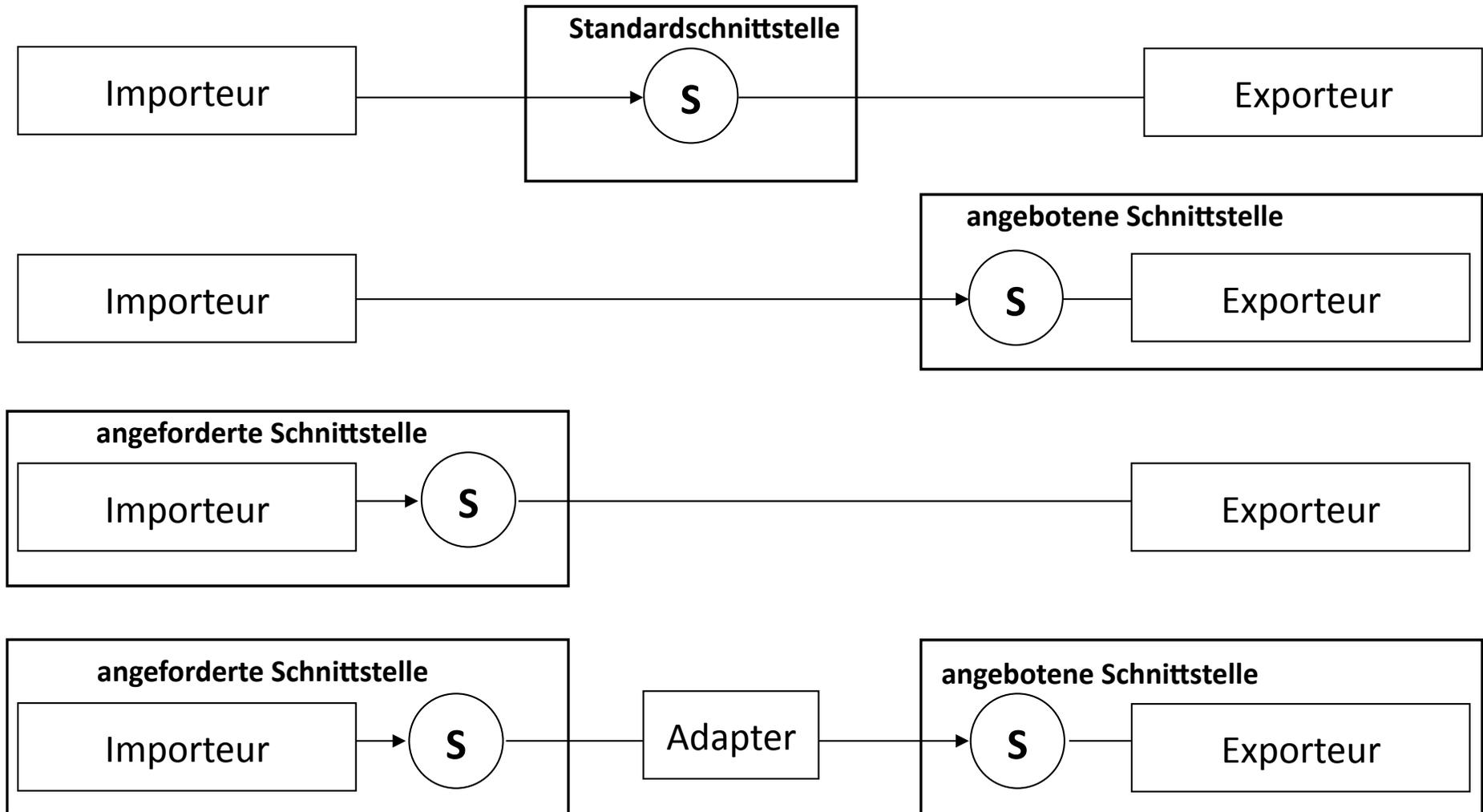
- Komponenten schließen untereinander Verträge ab, indem sie die zugehörigen Schnittstellen implementieren.
- Ein Vertrag bedeutet das Versprechen einer Komponente, bestimmte Leistungen zu erfüllen, wenn bestimmte Voraussetzungen gegeben sind.
- Ein System wird so aus Komponenten zusammengesetzt, dass es richtig funktioniert, wenn alle ihre Verträge einhalten.



Möglichkeiten der Realisierung von Contracts

- Ausprogrammieren
 - `if (kontoStand < 0) throw new IllegalStateException`
 - Vorteil: Einheitliche Sprache, überall verwendbar
 - Nachteil: Umständlich, uneinheitlich, können Seiteneffekte haben, kein Teil der Schnittstelle
- Nutzung von Assertion-Statements (z.B. in Java)
 - Beispiel: `assert (kontoStand >= 0)`
 - Vorteil: Einheitliche Sprache, einfach anzuwenden
 - Nachteil: Können durch Konfiguration ausgeschaltet werden, können Seiteneffekte haben, kein Teil der Schnittstelle
- Spracherweiterung für Design by Contract
 - Eigene Regelsprache für Vor- und Nachbedingungen
 - Vorteil: Mächtigkeit, Seiteneffektfreiheit, echter Teil der Schnittstelle
 - Nachteil: Integration, Komplexität

Wo werden Schnittstellen definiert?



Eigenschaften sauber entworfener Schnittstellen

- Klare Bezeichnungen
 - Verwendung von Namenskonventionen (getX, addX, removeX)
 - Sprechende Namen: `exec_pers_rd()` → `getPerson()`
 - Fachlich statt technisch getriebener Schnittstellen
- Schmale Schnittstellen
 - Erleichtert Verständnis und Nutzung
 - Erleichtert Implementierungsänderungen
- Interface Segregation
 - Eine Abstraktion pro Schnittstelle – Nutzer sollten nicht gezwungen sein, sich auf Operationen abzustützen, die sie gar nicht benötigen.
- (Möglichst) kontextlos
 - Nutzer sollten nicht gezwungen sein, komplexe Voraussetzungen zu erfüllen, um die Operationen der Schnittstelle aufrufen zu können.

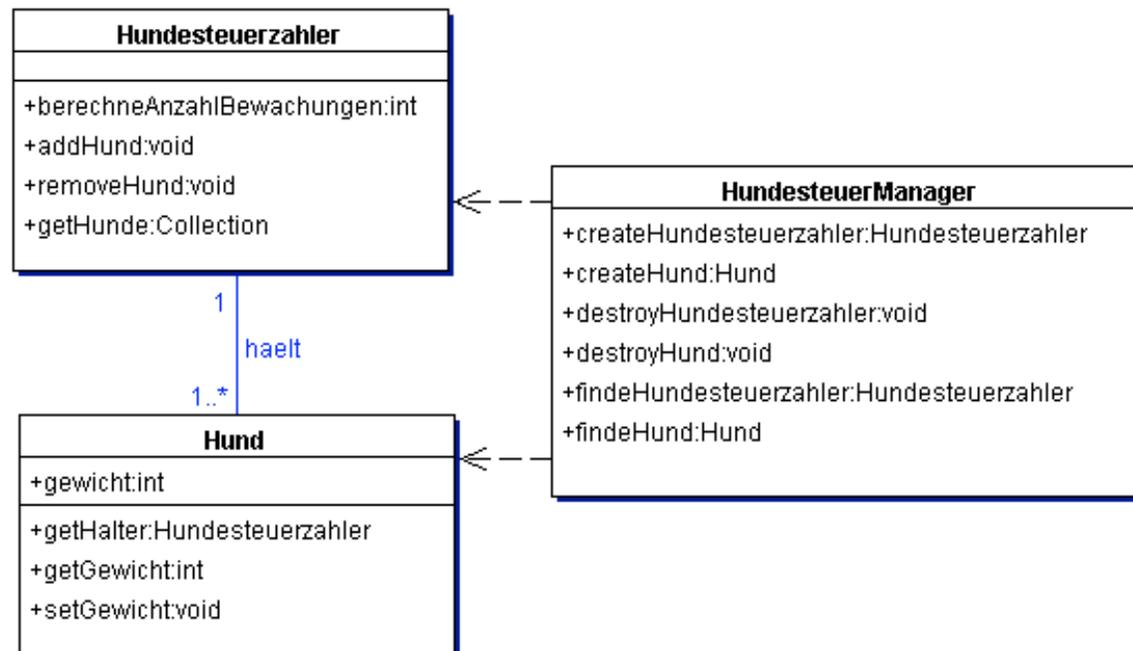
Fachlich vs. technisch getriebene Schnittstellen

- **Technisch getriebene Schnittstelle**
 - `bearbeiteKunde(aktion, id, name, adresse, konto)`
- **Fachlich getriebene Schnittstellen**
 - `erzeugeKunde(name, adresse, konto)`
 - `leseKunde(id)`
 - `aendereKonto(id, konto)`
 - `aendereAdresse(id, adresse, pruefen, korrigieren)`
 - `loescheKunde(id)`
- Fachliche Schnittstelle, aber technisch motiviert
- Direkt in Datenbank-Zugriffe umsetzbar
- Fünf verständliche Signaturen statt einer globalen, jeweils nur mit den benötigten Parametern)
- Lese- und Schreiboperationen mit unterschiedlicher Granularität (unabhängiges Ändern von Adresse und Konto)
- Zusätzliche Parameter bei einzelnen Operationen möglich (Prüfung oder Korrektur bei Adressänderung)

Aus: [Jos08]

Granularität von Schnittstellen (1)

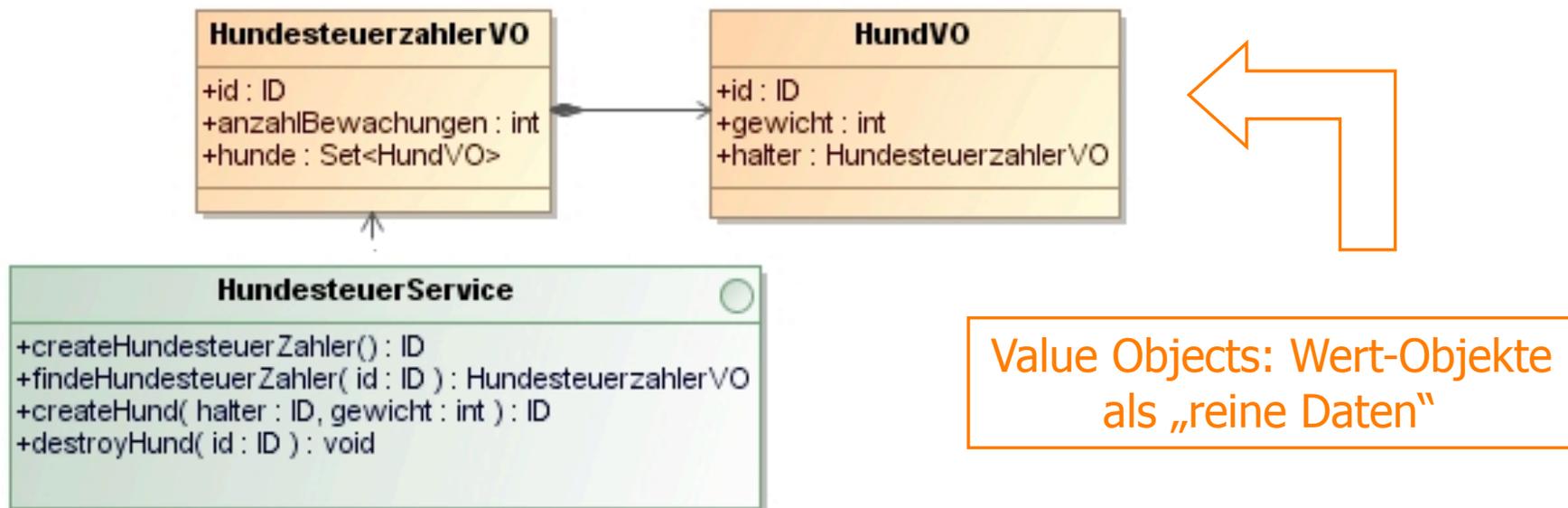
- Feingranulare, „objektorientierte“ Schnittstelle



- Viele „kleine“ Aufrufe, jeweils wenige übertragene Daten
- Gut für Zugriffe innerhalb einer Komponente wegen hoher Flexibilität
- Schlecht für Zugriffe über Netzwerke wegen hoher Latenz

Granularität von Schnittstellen (2)

- Grobgranulare, „serviceorientierte“ Schnittstelle



- Wenige „große“ Aufrufe, jeweils viele übertragene Daten
- Zu umständlich für Zugriffe innerhalb von Komponenten
- Gut für Zugriffe über Netzwerke und Anwendungen

Kontextlosigkeit von Schnittstellen

- Nutzung einfacher, überall bekannter Datentypen
 - Im Extremfall nur Zeichenketten
- Zustandslosigkeit als „Dogma“ der Serviceorientierung
 - Einfach zu erreichen bei lesenden Services
 - Inhärent unmöglich bei schreibenden Services ...
- Idempotenz
 - Mehrfache Aufrufe (z.B. bei Retrys aufgrund unsicherer Netze) haben die gleiche Wirkung wie ein einzelner Aufruf.
 - Einfaches Beispiel: `remove(hund:ID)` hat keine Auswirkung, wenn der Hund schon gelöscht wurde
 - Was tun bei `ueberweise(geldbetrag, kontoNr)`? Z.B. Nummer des Überweisungsformulars als zusätzlicher Parameter.
- Kein impliziter Kontext
 - Alle Informationen werden explizit über Parameter übergeben

Implizite versus Explizite Kontexte

Explizite Kontexte

- Vorteil: alle Parameter-Informationen sind sichtbar
- Nachteile: lange Parameterlisten; mühsames manuelles „Durchschleifen“ von Parametern

Implizite Kontexte

- Vorteil: wesentlich einfachere Nutzung der Schnittstelle
- Nachteile: Informationen werden implizit „unsichtbar“ übergeben; erfordert entsprechende Infrastruktur

Beispiel: Transaktionskontext: explizit oder thread-bezogen

```
tx := new Transaction();  
tx.begin();  
a.doSomething(3, p2, tx);  
b.remove(tx);  
c.update(1, p4, tx);  
tx.commit();
```

```
Transaction.begin();  
a.doSomething(3, p2);  
b.remove();  
c.update(1, p4);  
Transaction.commit();
```

Inhalt

- Wiederverwendung
- Schnittstellen
- **Frameworks**
- Aspect-oriented Programming
- Zusammenfassung

- *Nächste Stunde: Wiederverwendung per*
 - *Komponenten*
 - *Architekturmuster*
 - *Referenzarchitektur*
 - *Produktlinienarchitektur*

Frameworks: Definition

- Definition: Ein Framework ist ein „halbfertiges“ objektorientiertes Anwendungssystem.
- Frameworks
 - definieren in ihrem Anwendungsbereich die Softwarearchitektur
 - definieren die Regeln, nach denen sie zur vollständigen Anwendung weiterentwickelt werden
 - enthalten eine wiederverwendbare, allgemeingültige und durch das Zusammenspiel einzelner Instanzen des Frameworks vordefinierte Verarbeitungslogik
 - beeinflussen den gesamten Software-Entwicklungsprozess
 - erlauben die Wiederverwendung von Spezifikation und Code
 - realisieren das „Hollywood-Prinzip“ („Don't call us – we call you!“)

Blackbox und Whitebox-Frameworks

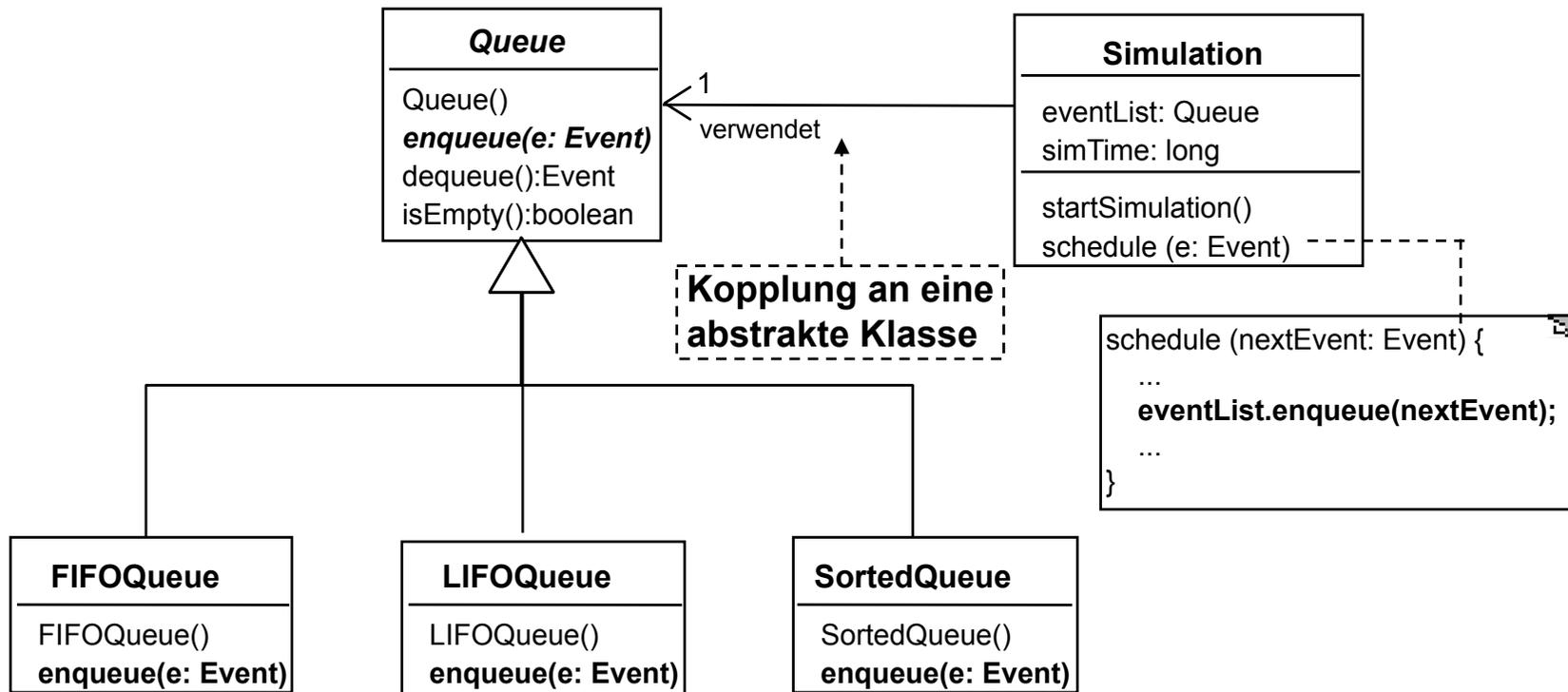
- **Anpassung durch Vererbung oder Komposition**
 - Whitebox-Frameworks erlauben Wiederverwendung durch Ableiten von abstrakten Klassen
 - Anwender muss Verhalten abstrakter Klassen implementieren
 - Blackbox-Frameworks ermöglichen Wiederverwendung durch Parametrisierung oder Komposition von Klassen
 - basieren auf einer Menge schon implementierter Klassen
 - Anwender bestimmt Verhalten durch Einfügen bestimmter Klassen
- **Interaktion mit dem Framework**
 - "Calling" Frameworks sind aktiv, d.h. beinhalten den zentralen Programmablauf und rufen andere Teile der Anwendung auf
 - "Called" Frameworks sind passiv und werden von anderen Teilen der Anwendung aufgerufen (= Bibliotheken)

Objektorientiert oder komponentenbasiert

- Traditionelle, objektorientierte Frameworks basieren stark auf dem Prinzip der Erweiterung und Wiederverwendung durch (Code-) Vererbung. Dies erweist sich in der Praxis als nachteilig:
 - „Vererbungsschnittstellen“ sind meist unpräzise
 - durch Vererbung ergeben sich starke Bindungen im Code
- Durch die Verwendung von Delegation anstelle von Vererbung ergeben sich flexible, objektorientierte Frameworks.
- Auf der Ebene von Komponenten wird ebenfalls der Mechanismus der Delegation genutzt – es ergeben sich komponentenorientierte Frameworks oder kurz Komponentenframeworks.

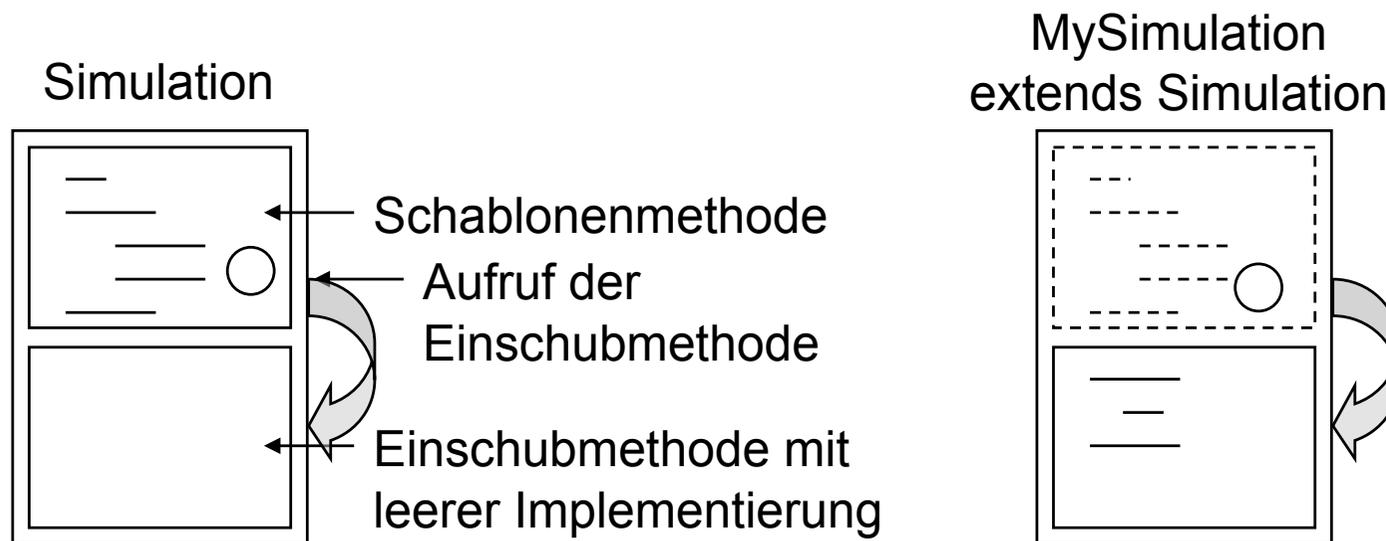
Abstrakte Kopplung [Pre97]

- abstrakte Klassen definieren eine Schnittstelle, zu der abgeleitete, implementierte Klassen kompatibel sind (Polymorphismus)



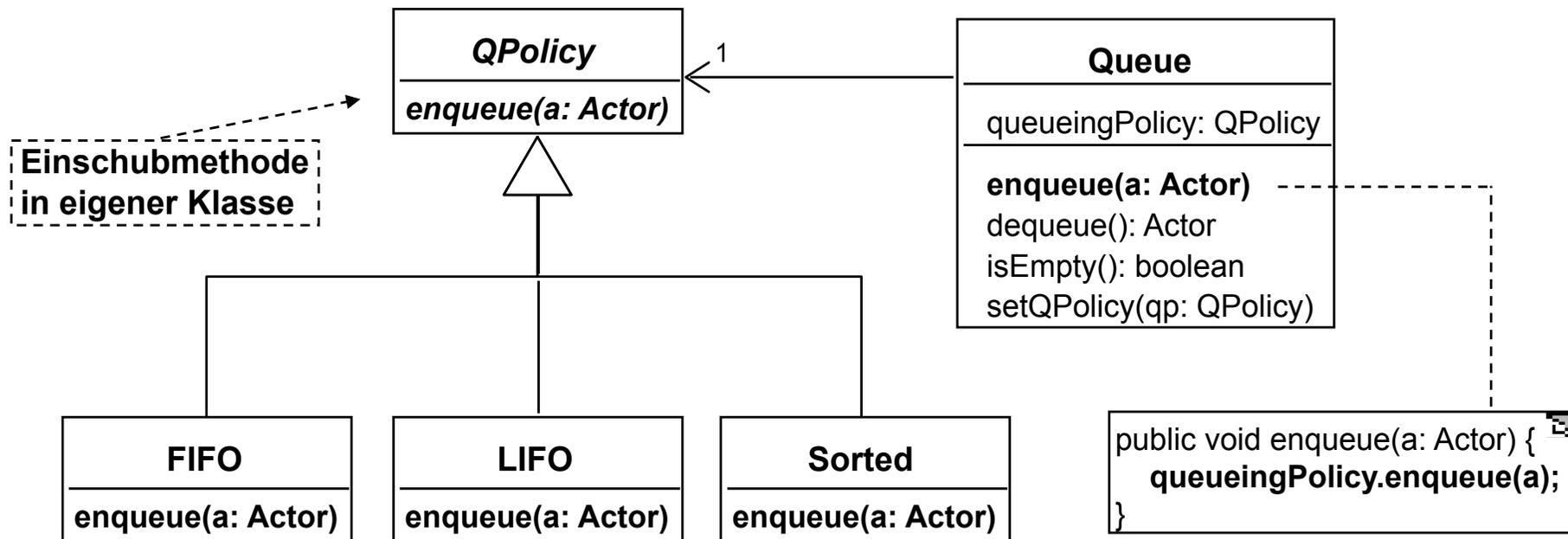
Einschubmethoden / Hot Spots [Pre97]

- Flexibilität durch Platzierung von "Einschubmethoden"-Aufrufen in "Schablonenmethode"
- Grundidee: Verhalten eines Objektes verändern, ohne den Quellcode der zugehörigen Klasse ändern zu müssen.



Einschubklassen [Pre97]

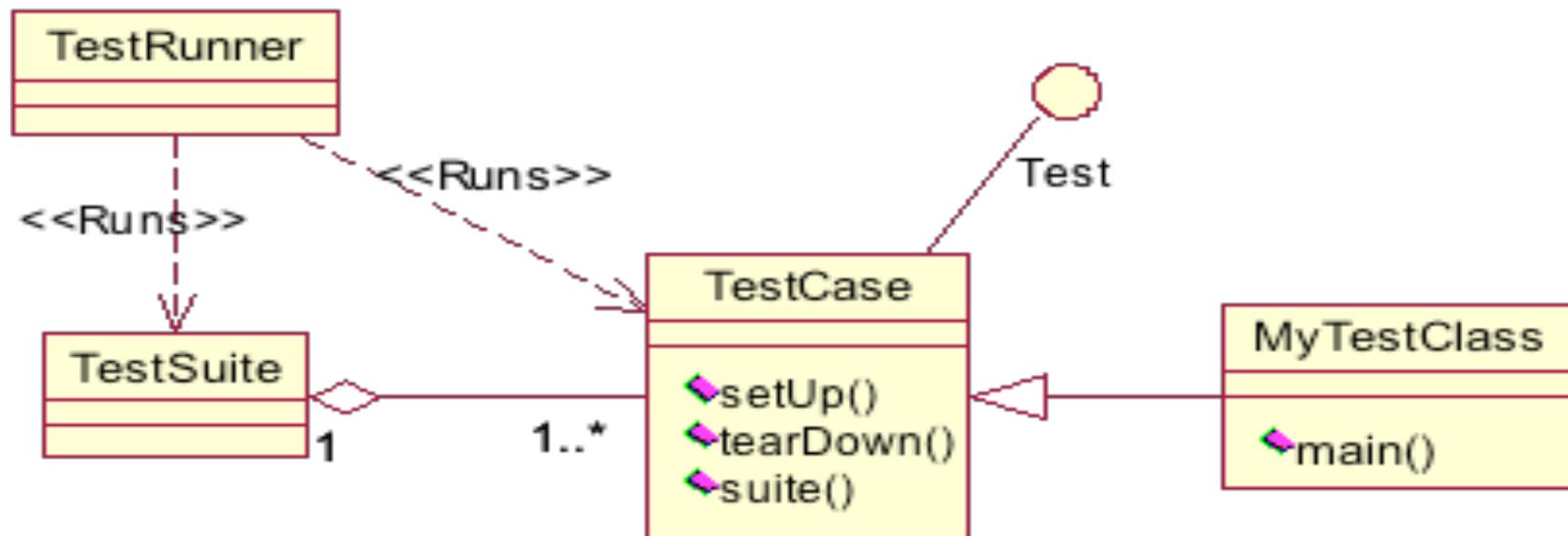
- Problem : Schablonen- und Einschubmethode in einer Klasse erlauben keine Anpassung zur Laufzeit!
- Lösung : Komposition statt Vererbung. Einschubmethode in eigener Klasse kapseln (Einschubklasse)



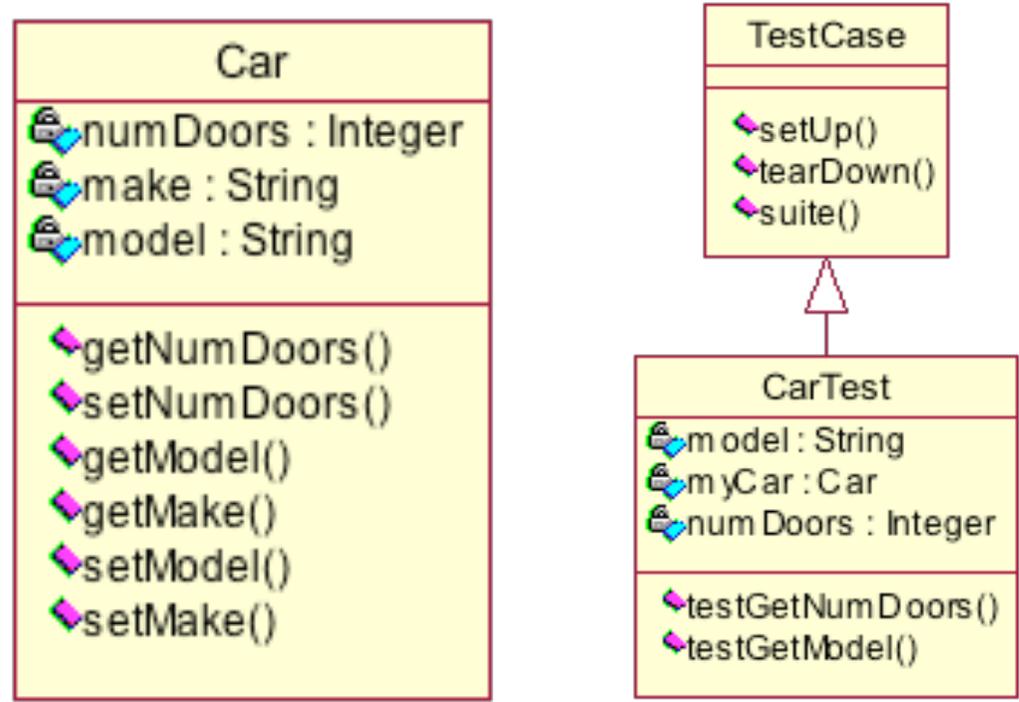
Beispiel: JUnit

- JUnit ist ein handliches Test-Framework in Java
- Eine Reihe von erweiterbaren Klassen realisiert die Testfunktionalität (Fehler zählen, auswerten, darstellen, Testfälle durchspielen)
- Es bietet eine einfache, doch komfortable Oberfläche
- Grundlage einer Infrastruktur für "extreme testing" (nightly build, nightly test)
- Entwickelt von
 - Kent Beck, kentbeck@compuserve.com
 - Erich Gamma, erich_gamma@acm.org

JUnit: Klassendiagramm



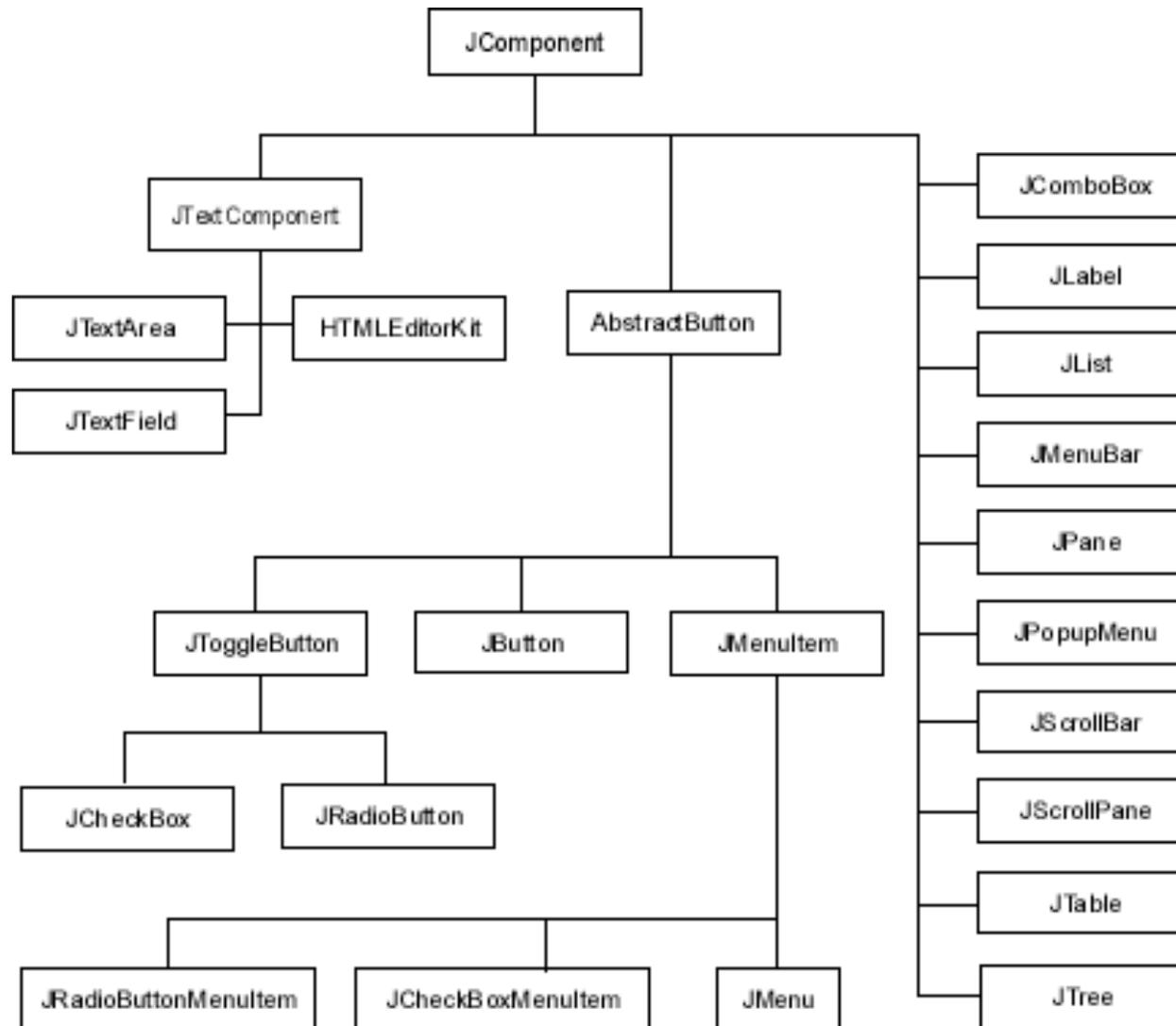
JUnit: Testklasse und Testfall



JUnit: Realisierung Testfall-Klasse „CarTest“

```
public void setUp() {
    myCar = new Car ();
    numDoors = 4;
    model = "BMW 318i";
}
public void testGetNumDoors() {
    assert (numDoors == myCar.getNumDoors());
}
public void testGetModel() {
    assert(myCar.getModel().equals(model));
}
public static void main(String[] args) {
    junit.textui.TestRunner.run(suite());
}
```

Beispiel: Das Swing-Framework



Beispiel: Nutzung von Swing

```
1: import java.awt.event.*;
2: import javax.swing.*;
3:
4: public class Framework extends JFrame {
5:     public Framework() {
6:         super("Application Title");
7:         // Hier die Komponenten einfügen
8:     }
9:
10:    public static void main(String[] args) {
11:        JFrame frame = new Framework();
12:
13:        WindowListener l = new WindowAdapter() {
14:            public void windowClosing(WindowEvent e) {
15:                System.exit(0);
16:            }
17:        };
18:        frame.addWindowListener(l);
19:
20:        frame.setVisible(true);
21:    }
22: }
```

Entkopplung von Swing

- Die wünschenswerte Entkopplung von Objektmodell und Swing wird über das Adapter-Pattern realisiert

```
public class AddressBook{
    List personList;

    public int getSize(){...}
    public int addPerson(...){...}
    public Person getPerson(...){...}
}
```

```
public interface TableModel
{
    // return number of rows
    getRowCount();

    ...
}
```

```
public class AddressBookTableAdapter
implements TableModel
{
    AddressBook ab;

    public AddressBookTableAdapter(
        AddressBook ab ){
        this.ab = ab;
    }

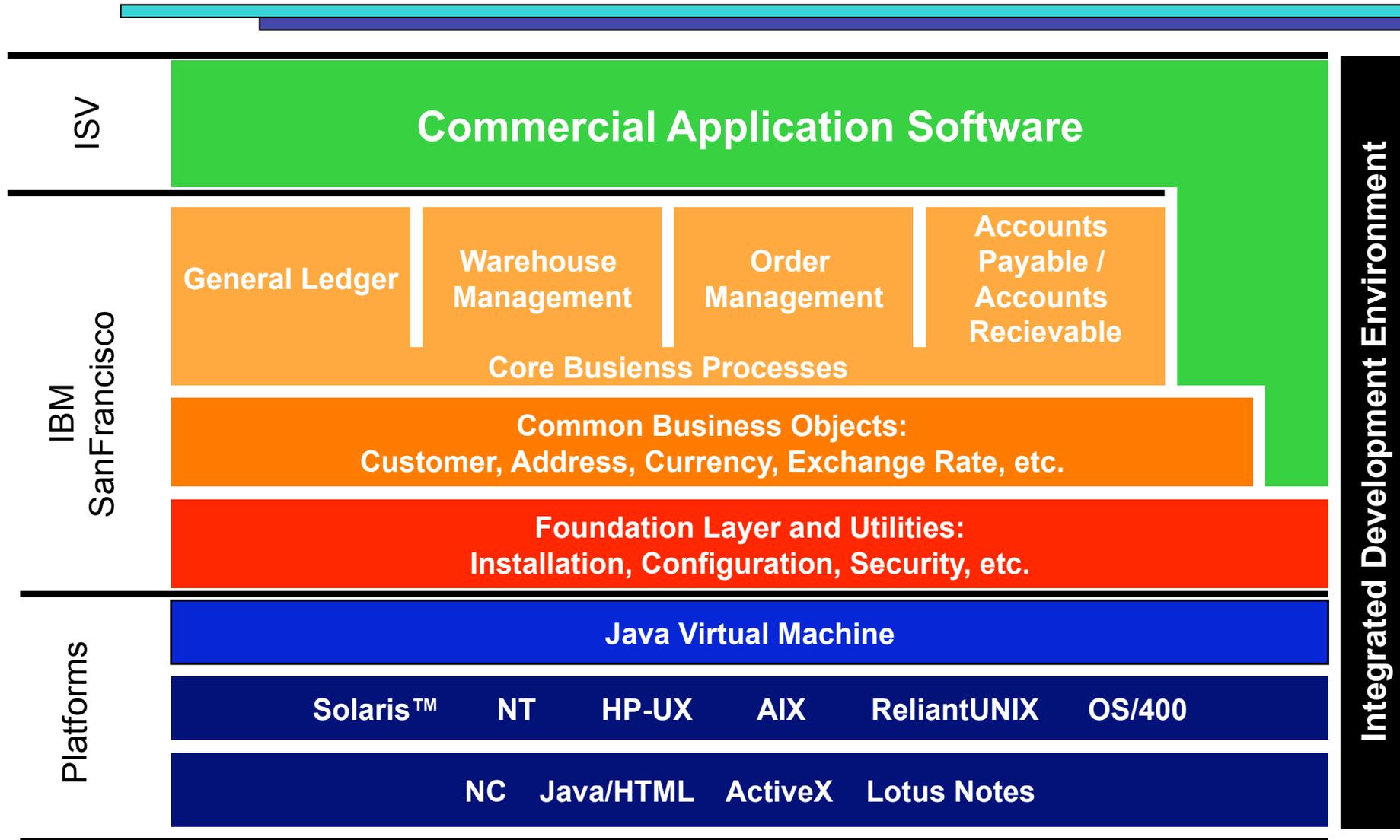
    //TableModel impl
    public getRowCount(){
        ab.getSize();
    }

    ...
}
```

Grenzen von Frameworks: San Francisco von IBM

- Gescheiterter Ansatz zur Standardisierung von Geschäftsobjekten und –prozessen im Rahmen eines Frameworks für Informationssysteme.
- Das Framework enthielt
 - Geschäftsobjekte, die domänenübergreifend benötigt werden
 - Breites Spektrum an Konstrukten für unterschiedliche Geschäftsfelder
 - Schnittstellen zu zentralen Geschäftsprozessen des Finanzmanagements
 - übliche Funktionalitäten eines Informationssystems
- Wurde inzwischen vom Markt genommen ...

San Francisco: Architektur

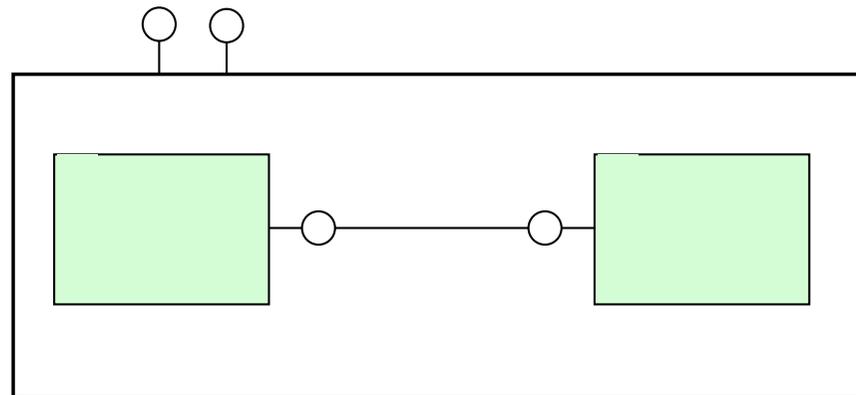


Gründe für das Scheitern

- Wie viele andere Frameworks strebt auch SF nach „alleiniger Herrschaft“ in der Anwendung.
- Die Komplexität des Frameworks war für nicht-spezialisierte Fach-Entwickler nicht mehr zu beherrschen
 - Bis zu 30-stufige Ableitungshierarchien – auf jeder Ebene kommen neue fachliche Konzepte und Mechanismen hinzu
 - Adressierung von technischen Mechanismen im Framework
- Wünschenswert wäre eine Möglichkeit, einfachere Frameworks leicht miteinander zu kombinieren.
 - Vorteil: Reduzierte Komplexität der Einzellösungen
 - Nachteil: Zusammenspiel und Aufteilung des Kontrollflusses kompliziert. Ansätze aus dem Bereich der Feature Interaction.

Ausblick: Komponentenframeworks

- Ein Komponentenframework ist eine besondere Komponente
- Ihre Realisierung sieht freie Stellen („plugs“) vor zum Einfügen bestimmter bedarfsabhängiger Teilkomponenten.
- In der Blackbox-Sicht werden nur die Plugs der Komponente dargestellt.
- In der Glassbox-Sicht werden zudem interne Komponenten des Frameworks repräsentiert, die einen Teil seiner Realisierung darstellen.



Bewertung: Frameworks

- + Hohe Wiederverwendung nicht nur von Design, sondern gleichfalls von Artefakten der Implementierung.
- + Verkürzung der Entwicklungsdauer für Anwendungen.
- + Weniger fehleranfällige Anwendungen durch Wiederverwendung bewährter Mechanismen.
- + Führen langfristig zu einer Standardisierung des jeweiligen Anwendungsbereichs.
- + Konzentration auf die wesentlichen Teile der Anwendung
- Hoher Einarbeitungsaufwand für den Nutzer des Frameworks.
- Frameworks geben meist Programmiersprache und Umgebung vor.
- Nur für einen bestimmten Problembereich anwendbar.
- Hoher Entwicklungsaufwand für das Framework.
- Frameworks „reißen die Kontrolle an sich“.

Inhalt

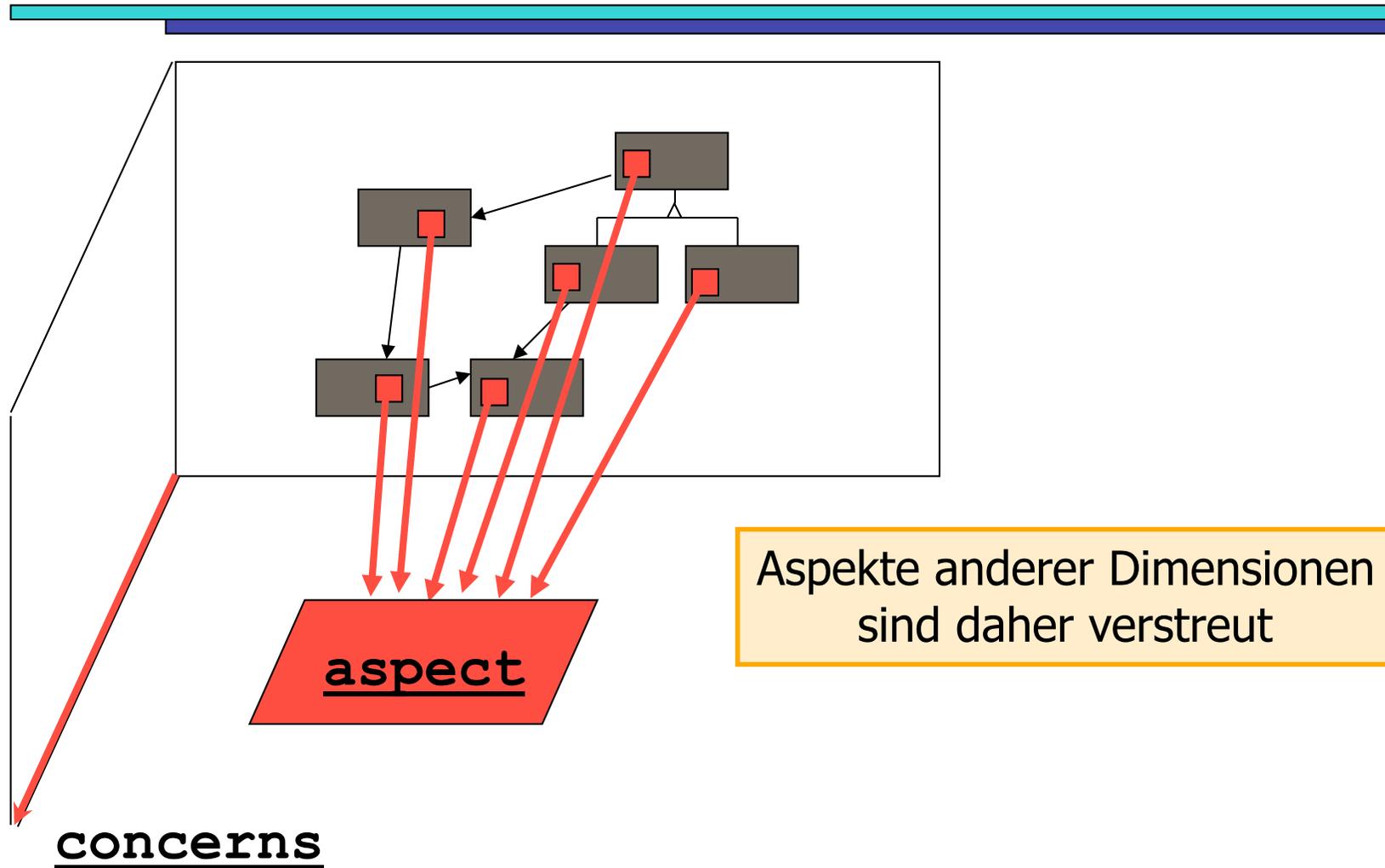
- Wiederverwendung
- Schnittstellen
- Frameworks
- **Aspect-oriented Programming**
- Zusammenfassung

- *Nächste Stunde: Wiederverwendung per*
 - *Komponenten*
 - *Architekturmuster*
 - *Referenzarchitektur*
 - *Produktlinienarchitektur*

Hierarchische Organisation

- Funktionen, Prozeduren und Objekte unterstützen die hierarchische Organisation von Systemen
- In der Regel bestimmen die fachlichen Anforderungen die Hierarchie
- Aspekte liegen "quer" zu diesen Hierarchien
- ☞ AOP ist für alle verallgemeinerten prozeduralen Sprachen möglich

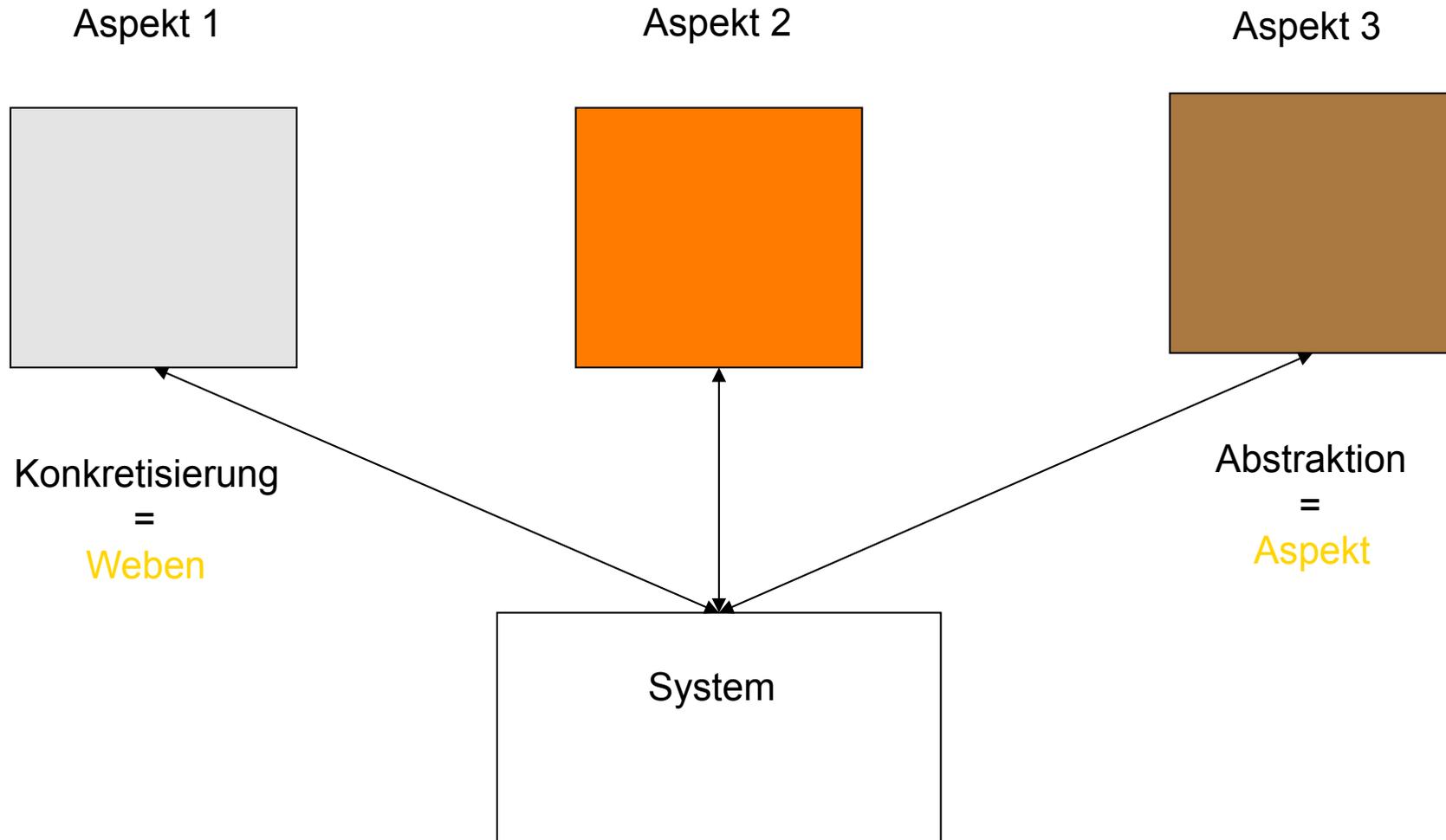
AOP: Opening a New Dimension in Software Development



Merkmale von Aspekten

- Die Anweisungen zur Protokollierung treten in mehreren Funktionen an bestimmten Stellen auf (Cross-Cutting)
- Sie können mit herkömmlichen Sprachmitteln aus den Funktionen nicht völlig herausgelöst werden (Modularisierung)
- Die Vorausplanung von Aspekten fällt schwer (Pre-Planning Problem)

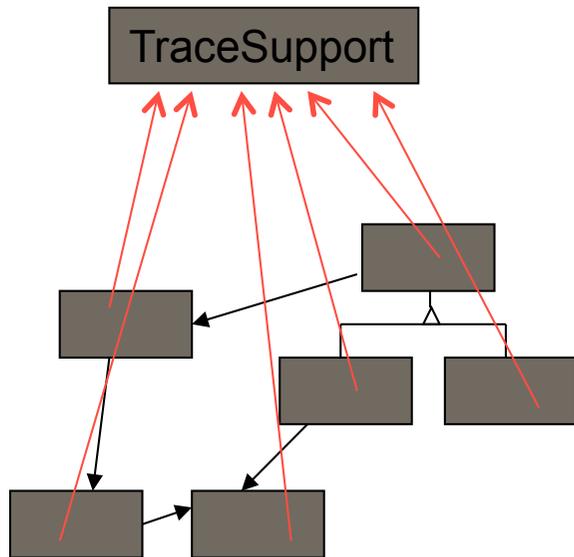
Zusammenführung von Aspekten in einem System



Aspect-oriented programming (AOP) [KHH+94]

- Verallgemeinerung von Architektursystemen:
 - Architektursysteme sind spezielle Aspektsysteme: Ihre beiden Aspekte sind anwendungsspezifische Funktionalität und Kommunikation
 - Weitere Aspekte: Synchronisation, Verteilung, Persistenz etc.
- Transparenz durch aspektbeschreibende Sprachen
Basiskommunikation: ein Aspekt
- Generierung: „Weben“ aus Aspektspezifikationen
- Schnittstellen: Join points, an denen Aspekte verwoben werden
- Interne Adaption: Aspekte bilden Teile von Komponenten, d.h. interne Adaption durch Austausch von Aspekten
- Externe Adaption: Weben von Klebecode um Komponenten herum

AOP: Beispiel



```
class TraceSupport {
    static int TRACELEVEL = 0;
    static protected PrintStream stream = null;
    static protected int callDepth = -1;

    static void init(PrintStream _s) {stream=_s;}

    static void traceEntry(String str) {
        if (TRACELEVEL == 0) return;
        callDepth++;
        printEntering(str);
    }

    static void traceExit(String str) {
        if (TRACELEVEL == 0) return;
        callDepth--;
        printExiting(str);
    }
}
```

```
class Point {
    void set(int x, int y) {
        TraceSupport.traceEntry("Point.set");
        _x = x; _y = y;
        TraceSupport.traceExit("Point.set");
    }
}
```

Probleme

- **Orthogonale Aspekte**
 - Keine Interaktionen zwischen den Aspekten
 - Weben einfach
- **Nicht-orthogonale Aspekte**
 - Z.B. Funktionalität und Synchronisation, Funktionalität und Verteilung, Protokolle und Synchronisation etc.
 - Getrennte Spezifikation unmöglich
 - Weben unklar
- **Aspekt als Sicht**
 - Abstraktion von Systemeigenschaften
 - Umkehrungsfunktion nicht immer
 - Eindeutig
 - Widerspruchsfrei mit anderen Aspekten

Inhalt

- Wiederverwendung
- Schnittstellen
- Frameworks
- Aspect-oriented Programming
- Zusammenfassung

- *Nächste Stunde: Wiederverwendung per*
 - *Komponenten*
 - *Architekturmuster*
 - *Referenzarchitektur*
 - *Produktlinienarchitektur*

Zusammenfassung

- Die pragmatische Wiederverwendung von Softwarearchitekturen in Form von Design, Implementierung und Erfahrung ist essentielle Voraussetzung für die effektive Softwareentwicklung
- Schnittstellen und Contracts sind die Basis für Software-Architekturen und Wiederverwendung.
- Frameworks stellen „halbfertige Anwendungen“ dar und ermöglichen die Wiederverwendung von Architekturen.
- Aspect-oriented Programming ermöglicht die Wiederverwendung orthogonaler Querschnittsaspekte.

Literaturhinweise

- [HNS99] Christine Hofmeister, Robert Nord, Dilip Soni: *Applied Software Architecture*, Addison Wesley – Object Technology Series, 1999.
- [Jos08] Nicolai Josuttis: *SOA in der Praxis* – dpunkt.verlag, 2008
- [DW98] Desmond Francis D'Souza, Alan Cameron Wills: *Objects, Components, and Frameworks With UML: The Catalysis Approach*, Addison Wesley Publishing Company, 1998
- [KHH+94] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, William G. Griswold: *An Overview of AspectJ*, ECOOP 2001, 2002
- [Pre97] Wolfgang Pree: *Komponentenbasierte Softwareentwicklung mit Frameworks*, dpunkt Verlag, 1997.