

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 1

Vorlesung „Methoden des Software Engineering“

Block B „Software Architektur“
Komponenten und Muster

Dr. Harald Störrle

Einheit B.4, 23.11.2004

Methoden des Software Engineering (c) 2004, Dr. Harald Störrle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 2

Kernpunkte heute

- **Es gibt sehr verschiedene Arten von Komponenten.**
 - Sie unterscheiden sich nach Größe, Stellung im Sw.-Lebenszyklus, Abgeschlossenheit und Art ihrer Verwendung.
- **Komponenten sind abgeschlossene, wiederverwendbare Einheiten.**
 - Manchmal spricht man von **Commercial Off The Shelf**-Komponenten.
 - Komponenten können auf unterschiedliche Arten realisiert werden.
- **Klassen sind keine (starken) Komponenten.**
 - Dafür sind sie zu klein, zu konkret, zu offen usw.
- **Muster sind Wissens-„komponenten“.**
 - Sie beschreiben eine bewährte Lösung für ein spezielles Problem.

Methoden des Software Engineering (c) 2004, Dr. Harald Störrle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 3

Gliederung Block B Teil 4: Muster & Komponenten

The diagram shows a vertical stack of six colored boxes labeled A through F, representing a framework. To the right, five green arrows point from this stack to five corresponding colored boxes. The connections are: A to Methodischer Entwurf; B to Systemarchitektur; C to Middleware; D to Muster & Komponenten; and E to Formale Architektur-Beschreibungssprachen. Box F (Web-Engineering) has no arrow pointing to a component on the right.

Methoden des Software Engineering (c) 2004, Dr. Harald Störrle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 4

Komponenten: Historischer Hintergrund

1968, NATO-Konferenz in Garmisch
 Douglas McIlroy: Mass Produced Software Components

Methoden des Software Engineering (c) 2004, Dr. Harald Störrle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 5

Motivation – Warum überhaupt Komponenten?

Der Einsatz von Komponenten ist gleichbedeutend mit der Standardisierung von Bauelementen.

The diagram is a diamond shape with four nodes: Standardisierung at the top, Qualität on the left, Produktivität at the bottom, and Wiederverwendung on the right. All four nodes are connected to each other by lines with a '+' sign at the intersection, indicating a positive relationship between all four factors.

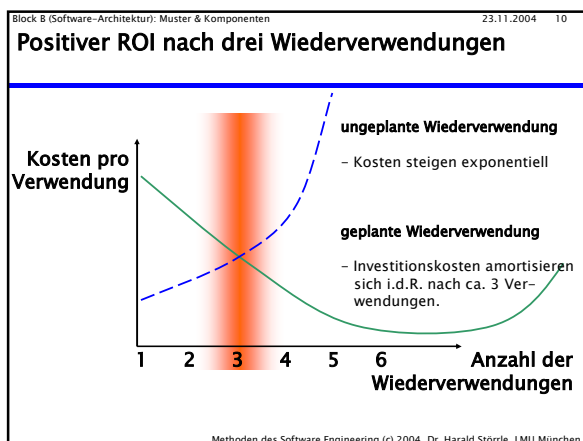
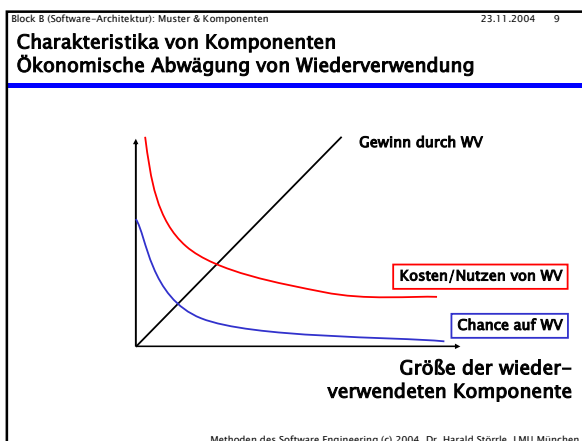
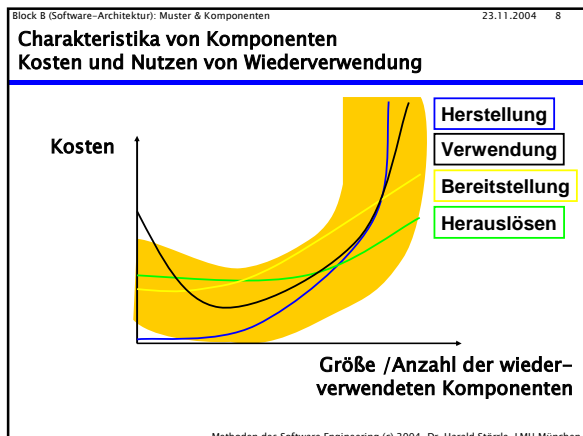
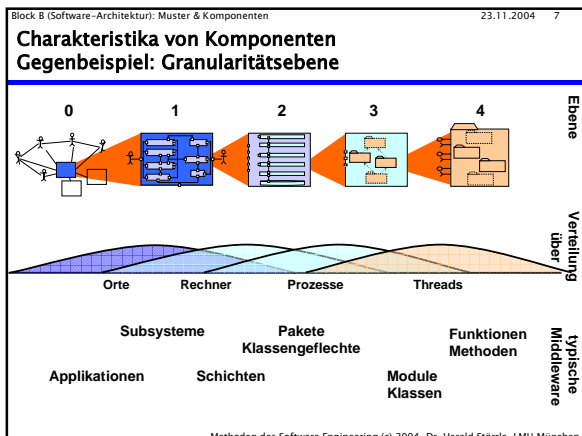
Methoden des Software Engineering (c) 2004, Dr. Harald Störrle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 6

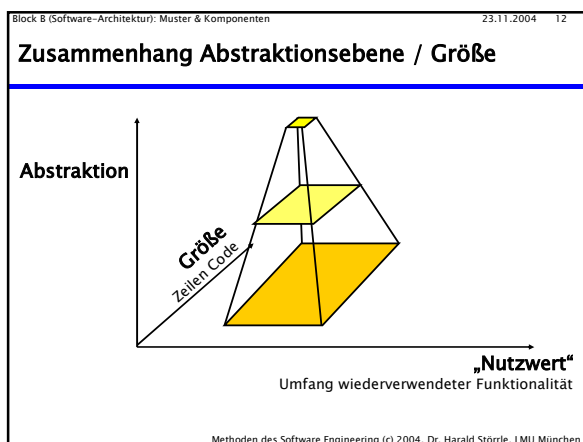
Charakteristika von Komponenten
Beispiele

- **Komponenten sollen komponentiert werden können.**
 - Sie kommen immer in Gruppen und sie müssen miteinander und mit anderen Programmen/Systemen einfach verbunden werden können
- **Komponenten sollen mehrfach benutzt werden können.**
 - Sie sollen in möglichst vielen verschiedenen Kontexten wiederverwendbar sein, daher müssen sie attraktiv und anpassbar sein.
- **Eine Komponente soll stabil sein („Abgeschlossenheit“).**
 - Es soll weder unerwünschte Einwirkungen vom Einsatzkontext auf die Komponente geben, noch umgekehrt.
- **Jede Komponente soll für sich stehen („Eigenständigkeit“)**
 - Komponenten sollen getrennt und unabhängig voneinander und vom Einsatzkontext hergestellt, geändert, benutzt, verteilt usw. werden können.

Methoden des Software Engineering (c) 2004, Dr. Harald Störrle, LMU München



- Block B (Software-Architektur): Muster & Komponenten 23.11.2004 11
- ### Charakteristika von Komponenten Gegenbeispiele
- **Größe**
 - „Reisesystem“ . . . „Container-Klasse“ . . . „Sinus-Funktion“
 - **Abstraktionsebene**
 - Anwendungsfachliches Wissen . . . Entwurfsmuster . . . Code
 - **Ziel / Motivation für Komponenten**
 - Qualität, Produktivität, Time-To-Market, Mitarbeiterzufriedenheit, . . .
 - **Stellung im Entwicklungsprozeß und in der Organisation**
 - Steinbruch . . . Investitionsgut . . . Machtinstrument
- Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München



Block 8 (Software-Architektur): Muster & Komponenten 23.11.2004 13

Die Nutzung von Komponenten erfordert spezielle Schritte im Entwicklungsprozeß

Herstellen

- Anforderungen erheben
- besondere Qualitätsmaßstäbe erfüllen (z.B. zus. Dokumentation, Optimierungen, Fehlerbehandlung etc.)

Bereitstellen

- Qualitätskontrolle
- Katalogisieren
- In Bestand übernehmen
- Komponenten vorhalten
- System pflegen

Einsetzen

- Anforderungen erheben
- Kandidaten sichten
- Kandidaten evaluieren
- Komponente anpassen
- Umgebung anpassen
- Einbauen

Ableiten

- aus existierendem System herauslösen
- von Projektspezifika säubern
- besondere Qualitätsmaßstäbe erfüllen (s.o.)

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block 8 (Software-Architektur): Muster & Komponenten 23.11.2004 14

Beispiele für Komponenten

Anlysemodell	Algorithmus	Modul-Spez.	Fachkonzept	Life Cycle Components
Entwurfsmuster	Idiom	Design-Patterns	Architekturmuster	
Entwurfsmodell	z.B. Aktivitätsdiagramm	Paket mit KD+ADs	kleines vollst. Modell	vollst. UML-Modell
Source-Code	Methode	Klasse, COM, JB	Framework-Paket	Applikation
ausführbarer Code		jar, COM-Komponente, exe-Datei		Business Obj.
	Funktion	Modul	Subsystem	Applikation

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block 8 (Software-Architektur): Muster & Komponenten 23.11.2004 15

Beispiele für Komponenten

Anlysemodell	Algorithmus	Modul-Spez.	Fachkonzept	Life Cycle Components
Entwurfsmuster	Idiom	Design-Patterns	Architekturmuster	
Entwurfsmodell	z.B. Aktivitätsdiagramm	Paket mit KD+ADs	kleines vollst. Modell	vollst. UML-Modell
Source-Code	Methode	Klasse, COM, JB	Framework-Paket	Applikation
ausführbarer Code		jar, COM-Komponente, exe-Datei		Business Obj.
	Funktion	Modul	Subsystem	Applikation

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block 8 (Software-Architektur): Muster & Komponenten 23.11.2004 16

Eigenheiten bei der Wiederverwendung von ausführbarem Code

Herstellen

- ✓ Anforderungen erheben
- ✓ besondere Qualitätsmaßstäbe erfüllen (z.B. zus. Dokumentation, Optimierungen, Fehlerbehandlung etc.)

Bereitstellen

- ✓ Qualitätskontrolle
- ✓ Katalogisieren
- ✓ In Bestand übernehmen
- ✓ Komponenten vorhalten
- ✓ System pflegen

Einsetzen

- ✓ Anforderungen erheben
- ✓ Kandidaten sichten
- ✓ Kandidaten evaluieren
- ✓ Komponente anpassen
- ✓ Umgebung anpassen
- ✓ Einbauen

Ableiten

- ? aus existierendem System herauslösen
- ✓ von Projektspezifika säubern
- ? besondere Qualitätsmaßstäbe erfüllen (s.o.)

- Qualitätsansprüche schwierig zu überprüfen bei .exe bzw. obfuscated class
- Katalogisieren und Finden/Sichten per API, Name, Zweck
- Evaluieren: Test-Suite/Dokumentation. Lohnt sich der Aufwand im Vergleich zum Nutzen?
- Anpassen: unmöglich, ggf. Wrapper im Nutzungskontext
- Einbauen einfach: Linken

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block 8 (Software-Architektur): Muster & Komponenten 23.11.2004 17

Komponenten im Sinne von ausführbarem Code: COM, JavaBeans (JB), Enterprise Java Beans (EJB)

- COM und JB sind lediglich ausführbare Code-Fragmente im Umfang einer Klasse, mit Einschränkungen bezüglich ihrer Schnittstellen und einer geringfügigen Infrastruktur zu ihrer Ausführung.
- Auch EJBs sind zunächst nur spezielle Klassen, haben aber ein sehr ausgeprägtes Programmiermodell und benötigen eine spezielle und umfangreiche Infrastruktur (Applikationsserver).
- Details zu EJBs von Nora Koch im Teil F.3.

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block 8 (Software-Architektur): Muster & Komponenten 23.11.2004 18

Wiederverwendung von Klassengeflechten

Wie implementiert man einen Software-Baustein?

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 19

Beispiele für Komponenten

Abstraktion	Analysemodell	Algorithmus	Modul-Spez.	Fachkonzept	Life Cycle Components
	Entwurfsmuster	Idiom	Design-Patterns	Architektur-muster	UML-Modell
	Entwurfsmodell	z.B. Aktivitätsdiagramm	Paket mit KD+ADs	kleines vollst. Modell	vollst. UML-Modell
	Source-Code	Methode	Klasse, COM, JB	Framework Paket	Applikation
	ausführbarer Code	Jar, COM-Komponente, exe-Datei			Business Obj.
	Funktion	Modul	Subsystem	Applikation	

Granularität

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 20

Eigenheiten bei der Wiederverwendung von Source-Code (Methode, Klasse, Paket)

Herstellen

- ✓ Anforderungen erheben
- ? - besondere Qualitätsmaßstäbe erfüllen (z.B. zus. Dokumentation, Optimierungen, Fehlerbehandlung etc.)

Bereitstellen

- ✓ Qualitätskontrolle
- ✓ Katalogisieren
- ✓ In Bestand übernehmen
- ✓ Komponenten vorhalten
- ✓ System pflegen

Einsetzen

- ? - Anforderungen erheben
- ? - Kandidaten sichten
- ? - Kandidaten evaluieren
- ? - Komponente anpassen
- ✓ Umgebung anpassen
- ✓ Einbauen

• Qualität erfordert (erheblichen) Mehraufwand (z.B. Einhaltung von Programmierrichtlinien, Optimierung, Dokumentation, ...). Ist dieser durch Nutzen gerechtfertigt?

• Katalogisieren und Finden/Sichten per API, Name, Zweck. Aber lohnt sich der Aufwand?

• Anpassen: möglich, aber aufwändig

• Einbauen einfach: Übersetzen

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 21

Eigenheiten bei der Wiederverwendung von Source-Code (Methode, Klasse, Paket)

- Warum sind hier so viele Probleme? Sind nicht Objekte der Inbegriff von Komponenten?
- Ja, aber...
- Auf der Ebene von Funktionsbibliotheken und Container-Typen sind Klassen brauchbar – genauso wie Module.
- Klassen sind aber keine guten Kandidaten für Wiederverwendung größerer Einheiten, da ihre Kapselung relativ schwach ausfällt.
 - Nur der Zugriff auf die Klasse wird eingeschränkt, nicht der aus der Klasse heraus.
 - Nur die Signatur wird kontrolliert, nicht das Verhalten.

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 22

Vererbung bricht Kapselung

FunTours

Reise

reservieren()
löschen()
...

Zustand: String

?

Angebot

reservieren,
löschen

$\Sigma_{\text{Reise}} = \{\text{reservieren, löschen}\}$

$= \{r, l\}$

$L_{\text{Reise}} = \{r, l\}^*$

Ebene: 1 | Phase: Architektur-Entwurf | Autor: HS | Version: 0.1 | Sicht: Struktur/Schnittstellen

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 23

Vererbung bricht Kapselung

Reise

↑

AndereReise

?

Angebot

reservieren,
löschen

?

Buchung

buchen
umbuchen
verlängern,
hochstufen,
erweitern

```

public class Reise {
    public int zustand = 0;

    public bool buchen(...) {
        if (!zustand.eq(„Angebot“))
            {throw...}
        zustand = „Buchung“;
        ...
    };

    public bool umbuchen(...) {
        if (!zustand.eq(„Buchung“))
            {throw...}
        zustand = „Angebot“;
        ...
    };
    ...
}

```

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 24

Vererbung bricht Kapselung

FunTours

Reise

↑

AndereReise

?

Angebot

reservieren,
löschen

buchen

Buchung

umbuchen
verlängern,
hochstufen,
erweitern

$\Sigma_{\text{AR}} = \{\text{reservieren, löschen, buchen, umbuchen, verlängern, hochstufen, erweitern}\}$

$= \{r, l, b, u, v, h, e\}$

$L_{\text{AR}} = (\{r, l\}^*(b(e, h, v)^*u)^*)^*b$

Ebene: 1 | Phase: Architektur-Entwurf | Autor: HS | Version: 0.1 | Sicht: Struktur/Schnittstellen

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 25

Vererbung bricht Kapselung

$\Sigma_{Reise} = \{\text{reservieren, löschen}\} = \{r, l\}$
 $\Sigma_{AR} = \{\text{reservieren, löschen, buchen, umbuchen, verlängern, hochstufen, erweitern}\} = \{r, l, b, u, v, h, e\}$
 $L_{Reise} = \{r, l\}^*$
 $L_{AR} = \{(r, l)^*(b\{e, h, v\}^*u)^*\}$

Instanzen von AR sind nicht mehr für den Typ Reise substituierbar.

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 26

Beispiele für Komponenten

	Algorithmus	Modul-Spez.	Fachkonzept	Life Cycle Components
Analysemodell	Idiom	Design-Patterns	Architekturmuster	UML-Modell
Entwurfsmuster	z.B. Aktivitätsdiagramm	Paket mit KD+ADs	kleines vollst. Modell	vollst. UML-Modell
Entwurfsmodell	Methode	Klasse, COM, JB	Framework-Paket	Applikation
Source-Code				
ausführbarer Code	Jar, COM-Komponente, exe-Datei			Business Obj.
	Funktion	Modul	Subsystem	Applikation
				Granularität

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 27

Wiederverwendung von Klassengeflechten

- Abstrakt wird eine Komponente durch einen Part beschrieben.
- Damit ist sowohl die Bedeutung Modell i.S.v. Paket abgedeckt, als auch ein ausführbares Stück Code.
- Aber wie kann ich dann ein Part mit Ports implementieren?
- Klassengeflechte mit Port-Klassen, unter Verwendung bestimmter Muster, z.B.
 - Business-Delegate
 - Proxy
 - Broker

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 28

Eigenheiten bei der Wiederverwendung von Klassengeflechten

- Wenn die Muster im Original schon enthalten sind, ist die Extraktion einfach – ansonsten kann der Aufwand extrem groß werden.

Herstellen ✓ Anforderungen erheben ✓ besondere Qualitätsmaßstäbe erfüllen (z.B. zus. Dokumentation, Optimierungen, Fehlerbehandlung etc.)	Bereitstellen ? Qualitätskontrolle ? Katalogisieren ✓ In Bestand übernehmen ✓ Komponenten verhalten ✓ System pflegen	Einsetzen ✓ Anforderungen erheben ✓ Kandidaten sichten ✓ Kandidaten evaluieren ✓ Komponente anpassen ✓ Umgebung anpassen ✓ Einbauen
--	--	--

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 29

Beispiele für Komponenten

	Algorithmus	Modul-Spez.	Fachkonzept	Life Cycle Components
Analysemodell	Idiom	Design-Patterns	Architekturmuster	UML-Modell
Entwurfsmuster	z.B. Aktivitätsdiagramm	Paket mit KD+ADs	kleines vollst. Modell	vollst. UML-Modell
Entwurfsmodell	Methode	Klasse, COM, JB	Framework-Paket	Applikation
Source-Code				
ausführbarer Code	Jar, COM-Komponente, exe-Datei			Business Obj.
	Funktion	Modul	Subsystem	Applikation
				Granularität

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 30

Muster-Beschreibungsschema nach Go4 (1)

- Musternamen**
 - treffend, knapp und präzise
- Klassifizierung**
 - Klassifizierung (nach Sammlungs-spezifischem Schema)
- Zweck (Intent)**
 - Was macht das Entwurfsmuster?
- Auch bekannt als**
 - Andere bekannte Namen (Synonyme) für das Muster
- Motivation**
 - Szenario für Entwurfsproblem und Lösung durch das Muster
- Anwendbarkeit**
 - Situationen, in denen das Pattern angewendet werden kann
- Struktur**
 - statische Struktur beteiligter Klassen (Klassendiagramm)

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 31

Muster-Beschreibungsschema nach Go4 (2)

- **Teilnehmer**
 - Beteiligte Klassen und Objekte und deren Zuständigkeiten
- **Interaktionen**
 - Zusammenarbeit der Teilnehmer (Interaktionsdiagramm)
- **Konsequenzen**
 - Vor- und Nachteile, Variationsmöglichkeiten
- **Implementierung**
 - Fallen, Tipps und Techniken
- **Beispielcode**
 - Code-Fragmente
- **Bekannte Verwendungen**
 - Beispiele aus echten Systemen (mind. 2 versch. Domänen)
- **VerwandtePatterns**
 - Beziehung/Unterschiede/Interferenzen zu anderen Patterns

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 32

Observer-Muster Zweck, Klassifizierung, ...

- **Zweck (Intent)**
 - Definiere eine 1-zu-n-Abhängigkeit zwischen Objekten, so dass die Änderungen des Zustands eines Objekts dazu führt, dass alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.
- **Klassifizierung**
 - Verhaltensmuster (Behavioral Pattern), Objekt-bezogen
- **Auch bekannt als (Also Known As)**
 - Dependents, Publish-Subscribe (Go5-Buch)
 - Beobachter

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 33

Observer-Muster Motivation

- Die Zustände von Objekten, die untereinander Beziehungen besitzen, müssen konsistent gehalten werden.
- Klassen sollen lose gekoppelt sein, um sie besser wiederverwenden zu können.
- Beispiel: GUI-Toolkit mit Trennung von Präsentation und zugrunde liegenden Anwendungsdaten:

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 34

Observer-Muster Motivation 2

- Die Klassen mit den Anwendungsdaten und den Präsentationen können unabhängig voneinander wiederverwendet werden, sie können aber auch zusammenarbeiten.
- Zwei verschiedene Präsentationen können die selben Daten anzeigen.
- Verschiedene Präsentationsobjekte kennen sich untereinander nicht.
- Sie verhalten sich aber so, als ob sie voneinander wissen würden.
- Das Observer Pattern beschreibt, wie die Beziehungen zwischen den Objekten aufgebaut werden.
- Dabei kann ein Subject eine beliebige Anzahl von Observern besitzen.

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 35

Observer-Muster Anwendbarkeit

- **Verwende das Observer Pattern in den folgenden Situationen:**
 - Wenn eine Abstraktion mehr als zwei Aspekte besitzt, von denen der eine vom anderen abhängt. Die Kapselung der Aspekte in unterschiedlichen Objekten ermöglicht es, diese zu variieren und unabhängig voneinander wiederzuverwenden.
 - Wenn die Änderung eines Objekts die Änderung anderer Objekte verlangt und nicht bekannt ist, wie viele Objekte geändert werden müssen.
 - Wenn ein Objekt in der Lage sein soll, andere Objekte zu benachrichtigen, ohne Annahmen darüber machen zu dürfen, wer diese anderen Objekte sind.

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 36

Observer-Muster Struktur

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 37

Observer-Muster Teilnehmer

- **Subject:**
 - kennt seine *Observer*. (beliebige viele);
 - stellt eine Schnittstelle zum An- und Abmelden der *Observer* zur Verfügung.
- **Observer:**
 - definiert eine Update-Schnittstelle für Objekte, die von Änderungen am *Subject* benachrichtigt werden sollen.
- **ConcreteSubject:**
 - hält den Zustand, an dem *ConcreteObserver* Objekte interessiert sind.;
 - schickt eine Nachricht an seine *Observer*, wenn sich der Zustand ändert.
- **ConcreteObserver:**
 - hält eine Referenz auf ein *ConcreteSubject* Objekt;
 - besitzt einen Zustand, der konsistent zum Zustand des *Subjects* bleiben soll;
 - implementiert die Update-Schnittstelle des *Observers* um seinen Zustand mit dem des *Subjects* konsistent zu halten.

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 38

Observer-Muster Interaktionen

- Das *ConcreteSubject* benachrichtigt seine *Observer* bei jeder Zustandsänderung, die den Zustand der *Observer* inkonsistent zu seinem eigenen machen könnte.
- Nachdem ein *ConcreteObserver* Objekt über eine Änderung im *ConcreteSubject* informiert wurde, kann ein *ConcreteObserver* Objekt weitere Informationen von *ConcreteSubject* holen.

```

sequenceDiagram
    participant CS as @ConcreteSubject
    participant CO as @ConcreteObserver
    participant ACO as anotherConcreteObserver
    CS->>CO: SetState()
    CS->>CO: Notify()
    CS->>CO: Update()
    CO->>CS: GetState()
    CS->>ACO: Update()
    ACO->>CS: GetState()
  
```

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 39

Observer-Muster Konsequenzen 1

- **Abstrakte Kopplung zwischen Subject und Observer:**
 - Das *Subject* kennt die konkreten Klassen der *Observer* nicht.
 - *Subject* und *Observer* können zu unterschiedlichen Software-Schichten gehören.
 - Ein *Subject* in einer tieferen Schicht kann einen *Observer* in einer höheren Schicht über Änderungen informieren, ohne die Schichtung zu verletzen.

```

classDiagram
    class ConcreteObserver
    class ConcreteSubject
    class Observer
    class Subject
    ConcreteObserver --> ConcreteSubject
    ConcreteObserver --> Observer
    ConcreteSubject --> Subject
    Observer o-- Subject
  
```

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 40

Observer-Muster Konsequenzen 2

- **Broadcast-Kommunikation wird unterstützt**
 - Änderungsmittelungen werden automatisch an alle registrierten *Observer* übermittelt.
 - Es ist dem *Observer* überlassen, ob und wie er mit der Nachricht behandelt.
- **Unerwartete Updates**
 - *Observer* kennen sich untereinander nicht.
 - Eine scheinbar harmlose Operation auf einem *Subject* kann eine Kaskade von Updates an *Observern* und deren abhängigen Objekten zur Folge haben.
 - Abhängigkeitsbeziehungen, die nicht gut definiert und gepflegt werden, führen im Allgemeinen zu störenden und schwer nachvollziehbaren Updates.

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 41

Observer-Muster Implementierung 1

- **Mapping von Subjects auf Observer**
 - Einfachste Lösung: *Subject* hält Referenzen auf *Observer*.
 - Für den Fall vieler *Subjects* und weniger *Observer* besser: Mapping-Tabelle für *Subject*-zu-*Observer* Mapping.
- **Observer für mehr als ein Subject**
 - Das Update-Interface muss dem *Observer* die Information, welches *Subject* die Nachricht geschickt hat, mitgeben (z. B. in Form einer Referenz auf das *Subject*).
- **Wer stößt den Update an?**
 - Methoden, die den *Subject*-Zustand ändern, rufen nach der Änderung `notify()` auf.
 - Klienten sind für den Aufruf von `notify()` zum richtigen Zeitpunkt verantwortlich.
- **Dangling References auf gelöschte Subjects**
 - *Subjects* müssen ihre *Observer* vor dem Löschen darüber informieren.

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 42

Observer-Muster Implementierung 2

- **Vermeide Observer-spezifische Update-Protokolle**
 - Push-Modell: Das *Subject* verschickt beim `Notify()` detaillierte Informationen darüber, was sich geändert hat.
 - *Observer* sind schwerer wiederverwendbar.
 - Pull-Modell: Das *Subject* informiert nur darüber, dass sich der Zustand geändert hat.
 - Evtl. ineffizient, weil die *Observer* ohne Hilfe des *Subjects* herausfinden müssen, was sich geändert hat.
- **Explizite Angabe der Interessierenden Änderungen**
 - *Observer* können sich nur für bestimmte Zustandsänderungen registrieren.
 - *Subjects* besitzen bestimmte Aspects, für die sich *Observer* registrieren können.

```

public class Subject {
    void attach(Observer observer, Aspect interest){...};
}

public class Observer {
    void update(Subject subject, Aspect interest){...};
}
  
```

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 43

Observer-Muster Implementierung 2

- Komplexe Update-Semantik in einem Change Manager kapseln**
 - Mapping zwischen Subject und Observern,
 - definiert Update-Strategie,
 - Durchführung des Update für das Subject

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 44

Observer-Muster Einbettung

- Bekannte Verwendungen**
 - Smalltalk Model/View/Controller (Model = Subject, View = Observer)
 - ET++ (Erich Gamma)
 - THINK Klassenbibliothek
 - User Interface Toolkits: InterViews, Andrew Toolkit, Unidraw
 - java.util-Klassen Observer, Observable
 - JavaBeans-Framework (PropertyChangeListener, PropertyChangeSupport)
- Verwandte Muster**
 - Mediator: Falls komplexe Update-Semantik in einer zentralen Instanz gekapselt wird, nimmt der ChangeManager die Rolle eines Mediators zwischen Subjects und Observern ein.
 - Singleton: Der Change Manager kann mit Hilfe des Singleton Patterns ein-Instanzig und global zugreifbar gemacht werden.

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 45

Eigenheiten bei der Wiederverwendung von Mustern

<p>Herstellen</p> <ul style="list-style-type: none"> - Anforderungen erheben - besondere Qualitätsmaßstäbe erfüllen (z.B. zus. Dokumentation, Optimierungen, Fehlerbehandlung etc.) 	<p>- Das Herstellen neuer Muster ist ein sehr sensibler Vorgang, und dauert sehr lange. Dies ist kein Vorgang, den eine einzelne Organisation wesentlich beeinflussen könnte.</p>
<p>Bereitstellen</p> <ul style="list-style-type: none"> ✓ - Qualitätskontrolle ✓ - Katalogisieren ✓ - In Bestand übernehmen ✓ - Komponenten vorhalten ✓ - System pflegen 	<p>Einsetzen</p> <ul style="list-style-type: none"> ✓ - Anforderungen erheben ✓ - Kandidaten sichten ✓ - Kandidaten evaluieren ✓ - Komponente anpassen ✓ - Umgebung anpassen ✓ - Einbauen
<p>Ableiten</p> <ul style="list-style-type: none"> ✓ - aus existierendem System herauslösen ✓ - von Projektspezifika säubern ✓ - besondere Qualitätsmaßstäbe erfüllen (s.o.) 	<p>- Der Einsatz erfordert sehr gute Kenntnisse der zur Verfügung stehenden Muster. Das Abrufen aus einer Bibliothek o.ä. führt in der Regel nicht zum Erfolg, und ist unter Projektsituationen auch keine Option.</p>

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 46

Eigenheiten bei der Wiederverwendung von Mustern (Spezialfall Frameworks)

<p>Herstellen</p> <ul style="list-style-type: none"> - Anforderungen erheben - besondere Qualitätsmaßstäbe erfüllen (z.B. zus. Dokumentation, Optimierungen, Fehlerbehandlung etc.) 	<p>- Abgesehen von Standardfällen ist nicht klar, wie überhaupt Anforderungen sinnvoll erhoben werden können. Spektakuläres Gegenbeispiel ist IBM San Francisco.</p>
<p>Bereitstellen</p> <ul style="list-style-type: none"> ? - Qualitätskontrolle ✓ - Katalogisieren ✓ - In Bestand übernehmen ✓ - Komponenten vorhalten ✓ - System pflegen 	<p>Einsetzen</p> <ul style="list-style-type: none"> ✓ - Anforderungen erheben ✓ - Kandidaten sichten ✓ - Kandidaten evaluieren ✓ - Komponente anpassen ✓ - Umgebung anpassen ✓ - Einbauen
<p>Ableiten</p> <ul style="list-style-type: none"> ? - aus existierendem System herauslösen ✓ - von Projektspezifika säubern ? - besondere Qualitätsmaßstäbe erfüllen (s.o.) 	<p>- Ähnlich wie bei Mustern: man muß das Framework gut kennen, dann ist der Einsatz sehr leicht. Ansonsten nicht.</p>

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 47

CALL FOR PAPERS EuroPloP 2005

- 10th year of the major European patterns conference
- 6th – 10th July 2005 Irsee, Bavaria, Germany
- EuroPloP is the European member of the PloP(tm) conference series.
- The focus at EuroPloP is on learning, discussion and reflection about patterns.
- The conference offers a variety of workshops that allow you to learn about patterns, to receive feedback on your own patterns, and to discuss patterns with fellow pattern enthusiasts.
- [http:// hillside . net / europlop](http://hillside.net/europlop)

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 48

Beispiele für Komponenten

Abstraktion	Algorithmus	Modul-Spez.	Fachkonzept	Life Cycle Components
Analysemodell	Idiom	Design-Patterns	Architektur-muster	
Entwurfsmuster				
Entwurfsmodell	z.B. Aktivitätsdiagramm	Paket mit KD-ADs	kleines vollst. Modell	vollst. UML-Modell
Source-Code	Methode	Klasse, COM, JB	Strukturiertes Paket	Applikation
ausführbarer Code				
	Jar, COM-Komponente, exe-Datei			Business Obj.
	Funktion	Modul	Subsystem	Applikation
				Granularität

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 49

Eigenheiten bei der Wiederverwendung von ausführbaren Entwurfsmodellen

Herstellen

- ✓ Anforderungen erheben
- ✓ besondere Qualitätsmaßstäbe erfüllen (z.B. zus. Dokumentation, Optimierungen, Fehlerbehandlung etc.)

Ableiten

- ? aus existierendem System herauslösen
- ✓ von Projektspezifika säubern
- ✓ besondere Qualitätsmaßstäbe erfüllen (s.o.)

- Das existierende und das zu bauende System müssen ebenfalls als ausführbares Modell vorliegen – das ist heute in der Regel nicht der Fall.

Bereitstellen

- ✓ Qualitätskontrolle
- ✓ Katalogisieren
- ✓ In Bestand übernehmen
- ✓ Komponenten vorhalten
- ✓ System pflegen

Einsetzen

- ? Anforderungen erheben
- ✓ Kandidaten sichten
- ✓ Kandidaten evaluieren
- ✓ Komponente anpassen
- ✓ Umgebung anpassen
- ✓ Einbauen

- Ansonsten: optimale Lösung, wenn Code-Generatoren vorliegen.

- Leider gibt es heute die hierzu erforderlichen Werkzeuge noch nicht. Aber wir arbeiten dran: Diplomarbeiten und FoPras auf Anfrage!

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 50

Eigenheiten bei der Wiederverwendung von Analysemodellen

Herstellen

- ✓ Anforderungen erheben
- ✓ besondere Qualitätsmaßstäbe erfüllen (z.B. zus. Dokumentation, Optimierungen, Fehlerbehandlung etc.)

Ableiten

- ✓ aus existierendem System herauslösen
- ? von Projektspezifika säubern
- ✓ besondere Qualitätsmaßstäbe erfüllen (s.o.)

- Da das Modell abstrakt ist, ist das Herauslösen und Einbauen in der Regel nicht sehr schwer.

- Allerdings kann die Anpassung erheblich sein.

Bereitstellen

- ? Qualitätskontrolle
- ✓ Katalogisieren
- ✓ In Bestand übernehmen
- ✓ Komponenten vorhalten
- ✓ System pflegen

Einsetzen

- ✓ Anforderungen erheben
- ✓ Kandidaten sichten
- ✓ Kandidaten evaluieren
- ✓ Komponente anpassen
- ✓ Umgebung anpassen
- ✓ Einbauen

- Nur wenn das wiederzuverwendende Modell sehr gut passt, kann man sinnvoll wiederverwenden.

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 51

Beispiele für Komponenten

Abstraktion Analysemodell Entwurfsmuster Entwurfsmodell Source-Code ausführbarer Code	Algorithmus	Modul-Spez.	Fachkonzept	Life Cycle Components
	Idiom	Design-Patterns	Architektur-muster	UML-Modell
	z.B. Aktivitätsdiagramm	Paket mit KD+ADs	kleines vollst. Modell	vollst. UML-Modell
	Methode	Klasse, COM, JB	Framework-Paket	Applikation
	Jar, COM-Komponente, exe-Datei			Business Obj.
	Funktion	Modul	Subsystem	Applikation

Granularität

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 52

Literatur

- **Reuse**
 - Zendler: Konzepte, Erfahrungen und Werkzeuge zur Software-Wiederverwendung. Tectum-Verlag, 1997
 - Schäfer, Prieto-Diaz, Matsumoto: Software Reusability (Proc. 1st Intl Conf. Sw.Reusability, 1991), Ellis Horwood, 1994
- **Business Objects:**
 - Herzum, Sims: Business Component Factory. John Wiley & Sons, 1999
- **Muster**
 - Buschmann, Meunier, et al.: Pattern-Oriented Software Architecture: A system of Patterns. Wiley, 1996 [„Gang of five“]
 - Gamma, Vlissides, et al. 1997 [„Gang of four“]
 - Linda Rising: The Patterns Almanach

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 53

Ausblick auf Block B Teil 5: Formale ADLs

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

Block B (Software-Architektur): Muster & Komponenten 23.11.2004 54

Facettenklassifikation

- **Kommt aus dem Bibliothekswesen**
- **Was is es**
 - semantisches Netz, Ähnlichkeitsfunktion [-> Funktions-Bsp aus Paper, Tafel!]
 - dynamisch sich ändernde Klassifikationsschemata
 - Ober-/Unterbegriffe
- **Wie funktioniert**
- **Vor-/Nachteile**
 - unscharf, aber simpl. Einzig realistische Option

Methoden des Software Engineering (c) 2004, Dr. Harald Störle, LMU München

