Software-Engineering

Vorlesung 6 vom 22.11.2004

Sebastian Iwanowski
FH Wedel

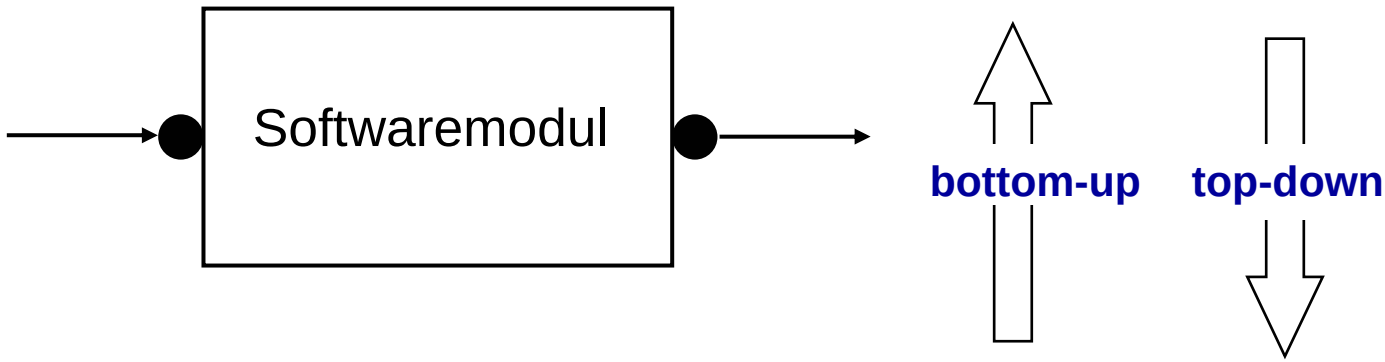
Software-Engineering

Vorlesungsthemen:

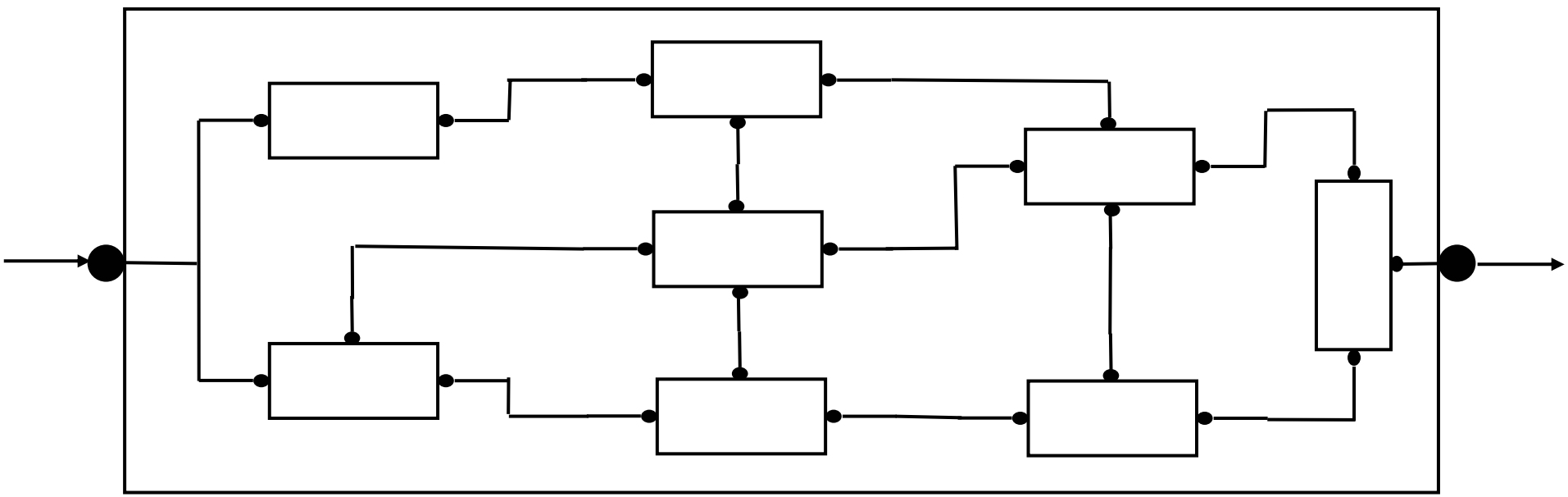
1. Überblick über das Thema und die Vorlesung
2. Grundlegende Prinzipien
3. Softwareplanung
4. Systemanalyse
5. Softwareentwurf
6. CASE-Tools
7. Aufwandsabschätzung
8. Qualitätsmanagement
9. Projektmanagement

Softwareentwurf: Vorgehensrichtung

Entwurfsebene 1:



Entwurfsebene 2:



Software-Module

Der Begriff des Moduls (nach Goos / Dennis 1973):

- Ein **Modul** ist die Zusammenfassung von Operationen und Daten zur Realisierung einer **in sich geschlossenen Aufgabe**.
- Die **Kommunikation** eines Moduls mit der Außenwelt darf nur über eine **eindeutig spezifizierte Schnittstelle** erfolgen.
- Zur **Integration** eines Moduls in ein Programmsystem muss die **Kenntnis des inneren Aufbaus entbehrlich** sein.
- Zum **Nachweis der Korrektheit** eines Moduls muss die **Kenntnis der Einbettung** des Moduls in das Softwaresystem **entbehrlich** sein.

Leitlinien für die Modularisierung

Die Aspekte bei der Modularisierung:

- Modulbindung
- Modulkopplung
- Schnittstellen
- Modulgröße
- Interferenzeigenschaften
- Importzahl
- Verwendungszahl

Leitlinien für die Modularisierung

Modulbindung

- Grad des Zusammenhangs zwischen den einzelnen Daten und Operationen desselben Moduls
- eng ↔ lose

Modulkopplung

- Grad des Zusammenhangs zwischen verschiedenen Modulen
- eng ↔ lose

Schnittstellen

- Datenaustauschstellen zwischen Modul und dem umgebenden System
- viele ↔ wenige

Leitlinien für die Modularisierung

Modulgröße

- Anzahl der verwendeten Daten bzw. Operationen in einem Modul
- groß ↔ klein

Interferenzeigenschaften

- Grad der Beeinflussung anderer Module durch Seiteneffekte
- hoch ↔ niedrig

Importzahl

- Anzahl der in diesem Modul verwendeten Module
- viele ↔ wenige

Verwendungszahl

- Anzahl der Module, die diesen Modul verwenden
- viele ↔ wenige

Leitlinien für die Modularisierung

Folgende Kompromisse sollten geschlossen werden:

- Ausgewogenes Verhältnis zwischen Modulbindung und Modulkopplung
- Minimale Schnittstellen (Ausnahme: wenn sonst das Modul zu groß wird)
- Modulgröße maximal so groß, dass eine Person das Modul in einem überschaubaren Zeitraum bearbeitet
- Mittlere Import- und Verwendungszahlen

1. Bsp. für eine Modulart: Bibliotheksmodule

Bereitstellung einer Sammlung von Algorithmen / Funktionalitäten zu einem Aufgabenbereich

Besonderheiten:

- Umfangreiche Schnittstelle (eine weitere Ausnahme)
- Kein interner Zustand (Modul erfüllt keine Gesamtfunktionalität)

Beispiele:

- Module mit mathematischen Funktionen
- Module für Hilfsfunktionen, die nicht problemspezifisch sind (z.B. Modul Util mit Umwandlungsfunktionen, Datumsfunktionen, etc.)

2. Bsp. für eine Modulart: Datenkapselung

Zweck:

Verhinderung des direkten Zugriffs auf Daten von anderen Programmteilen aus

Aufbau:

Die Datenkapsel besteht aus abstrakten Datenstrukturen/-typen und zugehörigen Zugriffsfunktionen.

„Abstrakt“ heißt: Der konkrete Aufbau der Datenstrukturen/-typen ist außerhalb des Moduls unbekannt.

Bsp.: Wörterbuchproblem (Dictionary)

Abstrakter Datentyp: `Dictionary`

Zugriffsfunktionen:

```
getDictionary()
insert(dict, key, value)
delete(dict, key)
search(dict, key)
```

Softwareentwurf

hier noch nicht angesprochen:

Entwurf von Bedienungsoberflächen

➔ ***Thema der Vorlesung Software-Ergonomie***

Software-Engineering

Vorlesungsthemen:

1. Überblick über das Thema und die Vorlesung
2. Grundlegende Prinzipien
3. Softwareplanung
4. Systemanalyse
5. Softwareentwurf
6. CASE-Tools
7. Aufwandsabschätzung
8. Qualitätsmanagement
9. Projektmanagement

CASE-Tools

CASE = Computer Aided Software-Engineering

UML: Unified Modelling Language

- entworfen von James Rumbaugh, Ivar Jacobsen, Grady Booch
- führt Konzepte zusammen, die ursprünglich unabhängig voneinander entwickelt worden sind
- wird durch Softwarewerkzeug Rational Rose unterstützt
- Kern ist die Beschreibung von Systemen (real und Software) mit objektorientierter Terminologie
- bietet zusätzlich Beschreibungsmöglichkeiten für Prozesse
- Anspruch: Software wird nach Beschreibung in UML automatisch erstellt
- Konkurrenzprodukte: Together, etc. (im Wesentlichen gleiche Standards)

CASE-Tools

CASE = Computer Aided Software-Engineering

ARIS

- entstanden bei der IDS Scheer (Begründer der Wirtschaftsinformatik)
- geht aus von Beschreibung von Geschäftsprozessen
- integriert auch die UML-Funktionalität
- ist integriert in SAP / R3

Bestandteile eines UML-Tools

Die wichtigsten Funktionalitäten von Rational Rose und ähnlichen Werkzeugen:

- objektorientierte Systembeschreibung: Klassendiagramme mit Beziehungen und Hierarchien
- Beschreibung von Anwendungsszenarien (Use Cases)
- Beschreibung von Prozessabfolgen (Sequenzdiagramme)
- Beschreibung von Zustandsabfolgen (Zustandsübergangsdiagramme)
- Aktivitätsdiagramme: spezielle Zustands-Aktivitäts-Beschreibungen

UML: Klassendiagramme

Anmerkungen zu Assoziationen:

- Allgemeine Assoziationen beschreiben beliebige Beziehungen. In der Regel werden Beziehungen von einer Klasse A zu den anderen Klassen beschrieben, deren Kenntnis in den Methoden von A benötigt wird.

- Die spezielle Assoziation “Aggregation” beschreibt Teilobjekte: Es ist sinnvoll, dass solche Teilobjekte die Werte von Attributen des zusammengesetzten Objektes sind.

- Die spezielle Assoziation “Komposition” bedeutet: Ein Wegfall des zusammengesetzten Objekts hat den Wegfall des Teilobjekts zur Folge.

- Die spezielle Assoziation “Generalisierung” hat die Vererbung der Attribute und Methoden auf die spezialisierten Klassen zur Folge.

UML: Klassendiagramme

Unterschied in UML-Klassendiagrammen zwischen Systemanalyse und Softwareentwurf:

- Richtungspfeile bei allgemeinen Assoziationen sind erst im Softwareentwurf sinnvoll: Sie entsprechen der Erreichbarkeit eines implementierten Objekts von dem anderen Objekt aus.
- Eine gerichtete Assoziation wird genau dann benötigt, wenn ein Attribut oder eine Methode die Kenntnis einer anderen Klasse benötigt.

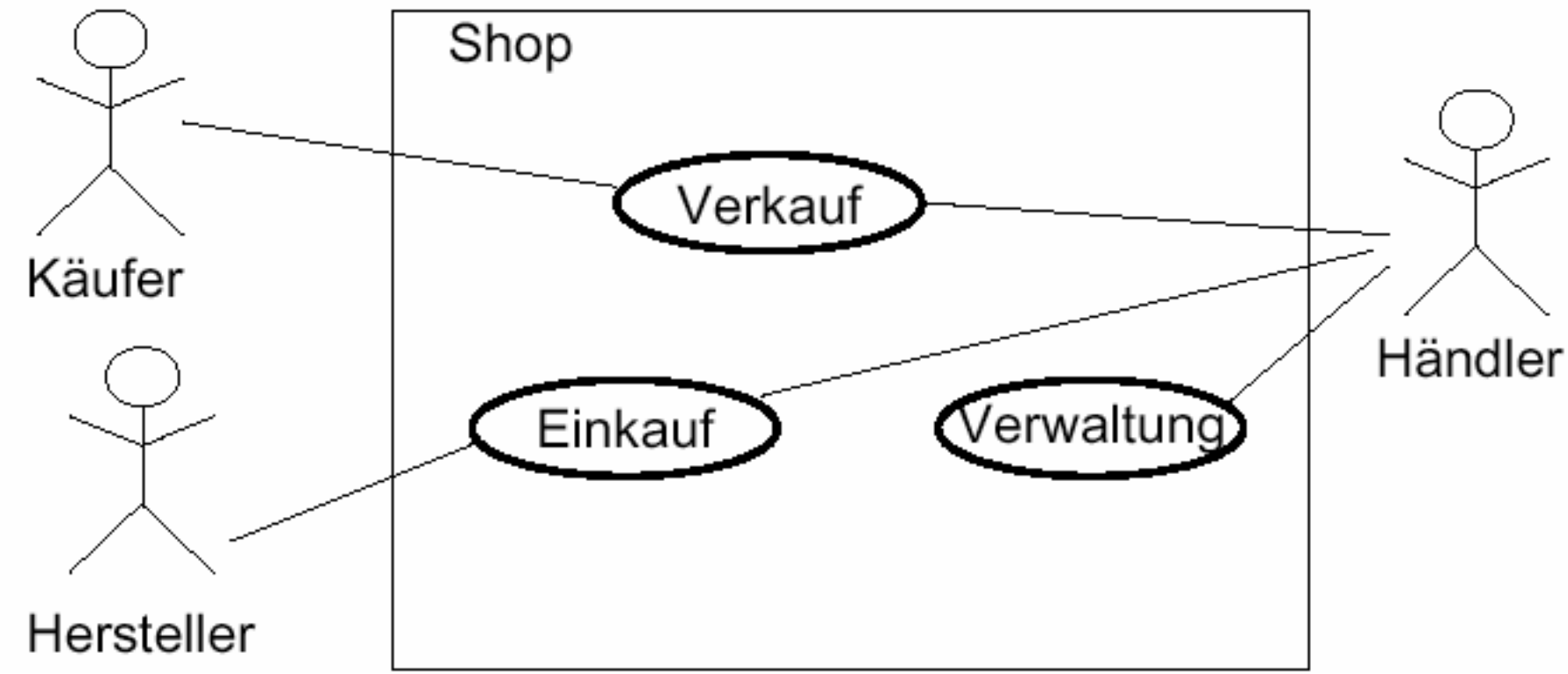
Zusammenfassung:

- In der Systemanalyse sollten grundsätzlich ungerichteten Assoziationen verwendet werden.
- Im Softwareentwurf sollten grundsätzlich gerichtete Assoziationen (uni- oder bidirektional) verwendet werden.
- Bei den Assoziationstypen Aggregation, Komposition und Generalisierung gibt es keine Unterschiede zwischen Systemanalyse und Softwareentwurf

UML: Use-Case-Diagramme

zur Beschreibung und Spezifikation von Systemen

Beispiel:



UML: Use-Case-Diagramme

Erklärung der Symbole:

Aktor (Strichmännchen):

- Jemand der mit dem System interagiert (d. h. außerhalb des Systems): muss keine natürliche Person sein

Use case (Ellipse):

- „Anwendungsfall“: Nach außen sichtbare Teilfunktionalität des Systems
- Kann in nachfolgenden Beschreibungen hierarchisch verfeinert werden
- Zwischen zwei Aktoren muss mindestens ein use case sein !

System (Rechteck):

- Unterscheidung von „innen“ und „außen“

UML: Use-Case-Diagramme

Ziele einer Use-Case-Beschreibung:

- Zur hierarchischen Beschreibung eines Systems
- Grobe Beschreibung einzelner Transaktionen

Weitere Beschreibungsmöglichkeiten für Beziehungen zwischen use cases:

- “extends”: Beschreibung einer Teilfunktionalität, die aber nicht immer ausgeführt wird (verzweigt sich bei so genannten Extension points des ursprünglichen use cases
- “includes”: Teilprozess, der von mehreren use cases benutzt werden kann

Beim nächsten Mal:
UML-Funktionalitäten (Fortsetzung)
ARIS