



Vorlesung  
Integrierte Modellierung komplexer Systeme  
Anwendungssysteme und UML

Bachelor Studiengang Informatik  
WS 2017-2018

Hugo Colceag | MHP

© 2018 MHP Management- und IT-Beratung GmbH

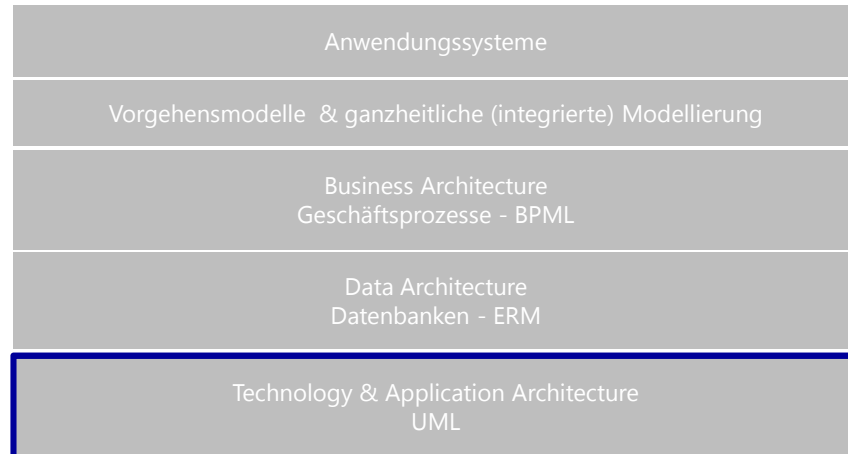


UNIVERSITATEA  
BABEȘ-BOLYAI

Einleitung - Integrierte Modellierung komplexer Systeme



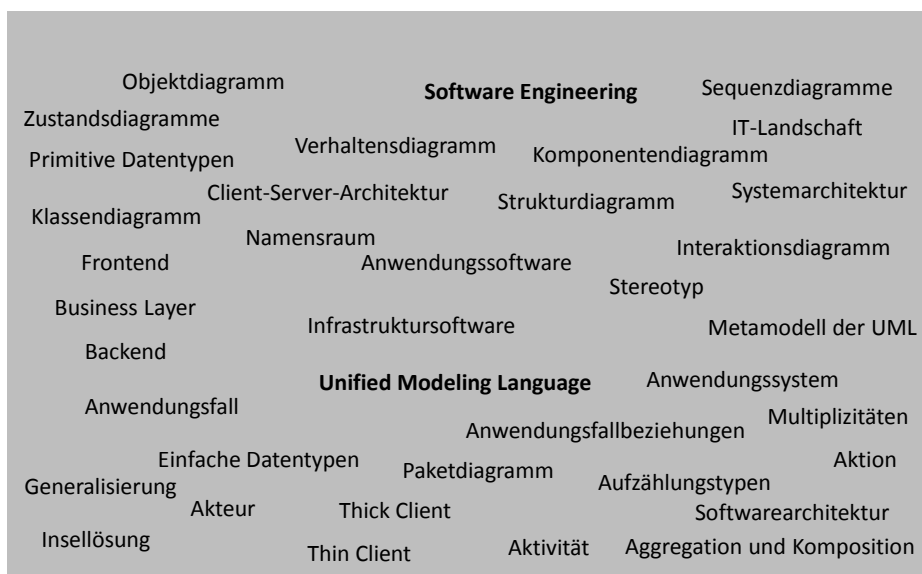
### 3 Vorlesungsinhalte und Aufbau



© 2018 MHP Management- und IT-Beratung GmbH

3

### Mindmap - Anwendungssysteme und UML



© 2018 MHP Management- und IT-Beratung GmbH

4

## Agenda

- 1 Aufbau von Anwendungssystemen
- 2 Methoden der SW Technik & UML

## 6 Aufbau von Anwendungssystemen

### 6.1 Anwendungssoftware und Anwendungssystem

- Daten und Informationen sind eine **wichtige** und **zentrale Ressource in Unternehmen**
- Endbenutzer nicht direkt über einen Datenbankserver auf Daten zu sondern über Anwendungsprogramme
- **Anwendungsprogramme**
  - Auf bestimmte fachliche Aufgaben zugeschnitten
  - darauf abgestimmte Bedienoberflächen
  - Nutzen im Hintergrund Dienste eines Datenbankservers
- **Umgang mit Daten**
  - Auswertung
  - mehrfach Nutzung
  - Austausch zwischen verschiedenen Anwendungssystemen
- **Ziele:**
  - Rationalisierung
  - Qualitätssteigerung
  - Services und neu Geschäftschancen

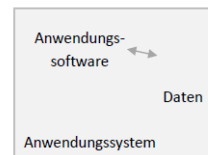
## 6 Aufbau von Anwendungssystemen

## 6.1 Anwendungssoftware und Anwendungssystem

- **Anwendungssoftware:**

Eine (betriebliche) Anwendungssoftware ist eine Software, mit der Endbenutzer eine bestimmte (betriebliche) Aufgabe oder Aufgaben aus einem (betrieblichen) Anwendungsgebiet bearbeiten können.

- **Anwendungssystem (im engeren Sinn):**  
ein Anwendungssystem ist eine Anwendungssoftware zusammen mit den damit erstellten und bearbeiteten Daten



- **Anwendungssystem (im weitergefassten Sinn):**  
ein Anwendungssystem im engeren Sinn **einschließlich** aller Hardware und Systemsoftware, die zum Betrieb der Anwendungssoftware nötig sind.
- **Infrastruktursoftware** (Systemsoftware) stellt Funktionen und Dienste bereit, die die eigentliche Anwendungssoftware nutzen kann, die Endnutzer aber nicht routinemäßig direkt bedienen.

## 6 Aufbau von Anwendungssystemen

## 6.1 Anwendungssoftware und Anwendungssystem

- In Unternehmen sind viele Softwaresysteme im Einsatz
- **IT-Landschaft** oder **Systemlandschaft:** Gesamtheit aller IT-Systeme, die ein Unternehmen verwendet
- **unübersichtliche** „gewachsene“ IT-Landschaften, die aufwändig zu warten sind
- häufig bestimmte Funktionen in verschiedenen Systemen **mehrfach** zu Verfügung
- **Ziel IT-Landschaft :**
  - Integrierte Informationsverarbeitung
  - Vernetzung („integriert“)
  - durchgängige Informationsflüsse über Systemgrenzen
  - schnelle Informationsverarbeitung
  - widerspruchsfreie Daten
  - Vermeidung von Doppelarbeit
- **Insellösung:** Anwendungssystem, das vom Datenaustausch mit den anderen Systemen in der IT-Landschaft abgekoppelt ist

## 6 Aufbau von Anwendungssystemen

## 6.2 Softwarearchitektur: Komponenten von Anwendungssoftware

- Moderne Anwendungssoftware:
  - großen Funktionsumfang
  - vielfältige Einstellungsmöglichkeiten
  - flexibel und komfortabel über grafische Bedienoberflächen zu bedienen
  - Mehrbenutzerbetrieb
  - **komplex** in der
    - Erstellung
    - Wartung
    - Weiterentwicklung
- **Software Engineering:** „Die Kunst und das Handwerk, komplexe Software-Systeme zu entwerfen und bauen“
- Maßnahmen:
  - Funktionalität in einzelne Komponenten verteilen
  - Komponenten unabhängig voneinander entwickelbar
  - Komponenten wirken über fest definierte Schnittstellen zusammen
  - Zusammen bilden die Komponenten die Anwendungssoftware

## 6 Aufbau von Anwendungssystemen

## 6.2 Softwarearchitektur: Komponenten von Anwendungssoftware

- **Software-Architektur**  
Als Software -Architektur bezeichnet man den Aufbau einer Software aus Komponenten und deren Zusammenspiel.
- Sehr verbreitet ist eine Unterteilung der Anwendungssoftware in drei Schichten
  - **Frontend** / Präsentationsschicht
    - dient der Darstellung der Daten und Informationen
    - Interaktion mit dem Benutzer
    - meist grafische Bedienoberfläche
  - **Business Layer** / Logikschicht / Anwendungsschicht / Geschäftsschicht
    - eigentliche Anwendung
    - Regeln, Prozesse, Workflows, etc. ...
  - **Backend** / Persistenzschicht
    - Datenspeicherung und Datenbereitstellung



Abbildung 6-2 Aufbau einer Anwendungssoftware: 3-Schicht-Architektur

## 6 Aufbau von Anwendungssystemen



## 6.2 Softwarearchitektur: Komponenten von Anwendungssoftware

- Beispiel 6.1 Architektur einer Anwendungssoftware „Autohaus“

Eine Anwendungssoftware für ein Autohaus erlaubt es unter anderem, Probefahrten zu terminieren und erfassen.

Das **Frontend** bietet auf einer grafischen Bedienoberfläche eine Eingabemaske, in der der Benutzer Kunde und Auto eingeben oder aus dem Kunden- und Autostamm auswählen kann.

Das **Backend** ruft Daten zu vorhandenen Kunden und Autos aus dem Datenbankmanagementsystem ab und übergibt eingegebene Daten zu neuen Kunden und zur Probefahrt an das Datenbankmanagemt, das die Daten in die Datenbank persistiert.

Die **Geschäftslogik-Schicht** enthält unter anderem Funktionen, die Terminüberschneidungen verhindern und dafür sorgen, dass nicht versehentlich ein Rückgabezeitpunkt eingegeben wird, der vor dem Startzeitpunkt liegt.

## 6 Aufbau von Anwendungssystemen



## 6.3 Systemarchitektur

- „verteilten System“  
Anwendung ist verteilt auf mehrere Rechnern (Hardware)

- **System-Architektur**  
Die Art, wie Software-Komponenten auf Hardware-Komponenten verteilt sind und wie sie zusammenwirken, bezeichnet man als System-Architektur.

## 6 Aufbau von Anwendungssystemen

## 6.4 Client-Server-Architektur

- **Server**
  - Systemkomponente, die Dienste bereitstellt
  - Die Server-Software läuft ständig
  - wartet auf Anforderung von Dienste

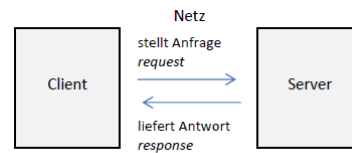


Abbildung 6-3 Client-Server-Architektur

- Kommunikation geht vom Client aus: Anfrage (*request*) an den Server
- Ein Server kann viele Clients bedienen.
- Clients können die Dienste mehrerer Server nutzen

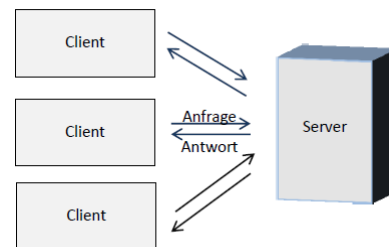


Abbildung 6-4 Ein Server bedient mehrere Clients

## 6 Aufbau von Anwendungssystemen

## 6.4 Client-Server-Architektur

## ▪ Beispiel 6.2 Beispiel Client-Server-Architektur: MySQL Workbench

In einem System aus MySQL Workbench und MySQL Datenbankmanagementsystems hat die MySQL Workbench die Rolle des **Client** inne: sie sendet Befehle (requests) an den MySQL **Server**, der die Befehle ausführt und eine Antwort zurücksendet, beispielsweise in Form einer Datenmenge. Die MySQL Workbench bietet eine grafische Benutzeroberfläche und Anwendungsfunktionen, etwa den Entwurf von Modellen und Lesen und Schreiben von SQL Script-Dateien.

Mehrere MySQL Workbench-Clients können gleichzeitig die Dienste des Datenbankservers nutzen.

## 6 Aufbau von Anwendungssystemen

## 6.4.1 Thin Clients und Thick Clients

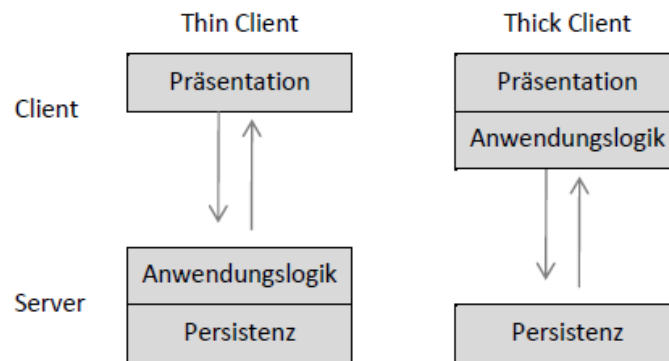


Abbildung 6-5 Thin Client – Thick Client

## 6 Aufbau von Anwendungssystemen

## 6.4.1 Thin Clients und Thick Clients

- Beispiel 6.3 Beispiel „**Thin Client**“: Wikipedia

Wikipedia ist ein System, mit dem Artikel erfasst, editiert, angezeigt und kommentiert werden können. Anwendungsfunktionen wie Benutzerverwaltung, Verwaltung von Versionsständen, Freigabe von Artikeln liegen komplett auf dem Server.

- Beispiel 6.4 Beispiel „**Thick Client**“: MySQL Workbench

Die MySQL Workbench ist eine eigenständige Software, die auch ohne Anbindung an einen Backend-Server sinnvoll genutzt werden kann. Die Anwendungsfunktionen (Modelle erstellen, Skripte speichern und laden) stecken im Client. Der Server übernimmt lediglich die Persistenz.



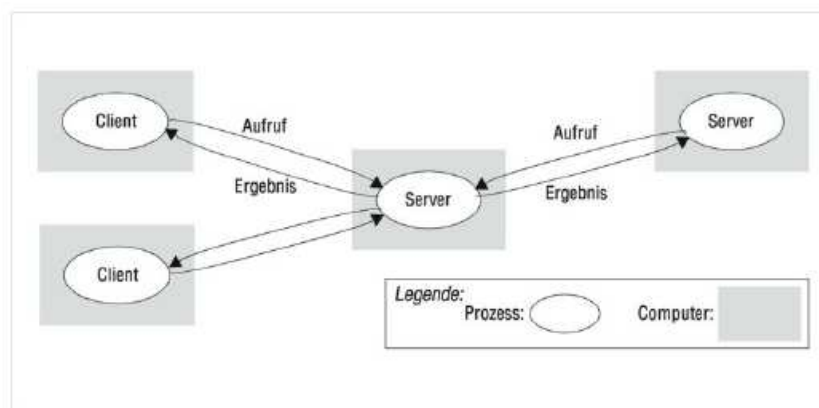
## 6 Aufbau von Anwendungssystemen

## 6.4.2 Vorteile und Nachteile von Thin Clients gegenüber Thick Clients

- **Thin Client:**
  - belastet die Server-Ressourcen generell stärker
  - Hardware-Anforderungen der Server-Seite sind größer
  - Anforderungen an die Hardware der Client-Seite sind geringer
  - weniger komfortabel zu bedienen
  - „Gefühlt“ langsam – Hohe Wartezeiten (*abhängig von Art und Menge der zu übertragenden Daten*)
  - belastet die Netzverbindung stärker und ist abhängig vom Netz.
  - ermöglicht vielen Nutzern gleichzeitig Zugriff auf dieselben Daten
- **Thick Client**
  - bessere Bedienbarkeit für Benutzer
  - offline nutzbar (!!! Datenreplikation und Datensynchronisation)
  - Auf Client-Rechnern muss Software installiert und gewartet werden
  - Client-Software abhängig von Client-Hardware
- **Webbrowser als Thin Client:**
  - Als Webanwendung bezeichnet man eine Anwendung, die auf der Client-Seite komplett im Webbrowser läuft. Die Client-Seite bedient sich eines Standard-Browsers, um eine graphische Benutzeroberfläche anzuzeigen. Sämtliche Anwendungsfunktionen stecken auf der Server-Seite. Hier ist auf der Client-Seite keinerlei Software-Installation und -Wartung nötig.

## 6 Aufbau von Anwendungssystemen

## 6.5 3-tier und n-tier-Architekturen



## 6 Aufbau von Anwendungssystemen

## 6.6 Datenbankanwendungen mit MS Access

- MS Access ist eine (Standard) Software, die es Endnutzer ermöglicht, Datenbanken anzulegen, Daten zu verwalten und Datenbankanwendungen inklusive Bedienoberfläche und Geschäftslogik zu erstellen.
- **Vorteile:**
  - übersichtliche Bedienoberfläche
  - viele Assistenzfunktionen
  - grafische Bedienoberflächen für Datenbanken
  - „Datenbank-Engine“, die auf dem relationalen Datenbankmodell beruht und SQL beherrscht
  - Datenbanken und Anwendungen für andere Systeme kompatibel und zugänglich
  - Zugriff auf externe Datenbanken
  - Erweiterbarkeit durch VBA Programmierung
- **Nachteile:**
  - begrenzte Zahl von Benutzern (maximal 10 bis 20) zur gemeinsamen Benutzung empfohlen
  - Eingeschränkte Funktionen zur Benutzerverwaltung und zum Vergabe von Zugriffsrechten
- MS Access eignet sich daher besonders als Tool für **Rapid Prototyping**, also dazu, schnell und einfach Softwarelösungen zu realisieren und bei Bedarf ganz oder teilweise auf leistungsfähigere Werkzeuge und Technologien umzusteigen.

## 6 Aufbau von Anwendungssystemen

## 6.7 Eine typische Web-Anwendung: Wordpress

- freie quelloffene Software, die dynamisch Webseiten erzeugt
- kostenlos nutzbar
- „Software für Blogs / Artikeln, die in der Reihenfolge ihrer Erstellung veröffentlicht werden
- Erweiterbar durch Plugins
- Unterstützung von Themes: Sammlung von Darstellung- und Layoutanweisungen
- In eCommerce-Anwendungen einsetzbar
- leichtgewichtiges Content Management-System

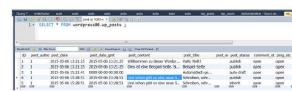


Abbildung 6-7 Wordpress Datenbanktabelle mit Inhalten

▪ **Content Management-System**

Software zur Verwaltung von Webinhalten bezeichnet man Content Management-System

▪ **Beispiele**

- <http://www.sonymusic.com/>
- <http://www.news-sap.com/>

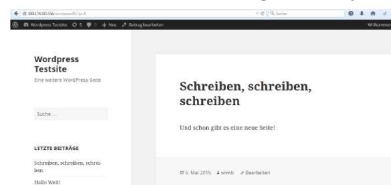


Abbildung 6-6 Die Wordpress-Seite

## 6 Aufbau von Anwendungssystemen

## 6.8 Übungen

- Aufgabe 6.2 Beispielanwendung mit Wordpress

Eine Wordpress-Site hat zwei Ansichten:

1. die öffentliche Ansicht für Leser ist zugänglich unter (hier)  
193.174.193.156/wordpress00/
2. die Admin-Seite für Blog-Betreiber ist zugänglich unter (hier)  
193.174.193.156/wordpress00/wp-admin

a) Sich mit der Anwendung vertraut zu machen:

- Am Blog anmelden
- Theme anpassen
- 2 Seiten erstellen
- 2 Blogeinträge

b) **Anwendungsmöglichkeiten:**

- Projekt/Projektarbeit: Kann ein Blog die Kommunikation per Email ersetzen? Vorteile und Nachteile?
- Wofür kann man einen Blog beruflich nutzen? Bitte nennen Sie 3 Anwendungsbeispiele mit Nennung und Erläuterung von Aufgabe, Anwendern und Nutzern

## Agenda

- 1 Methoden der SW Technik
- 2 Allgemeine Grundlagen UML
- 3 Anwendungsfall-/ Use Case Diagramme
- 4 Aktivitätsdiagramme
- 5 Objekt- und Paketdiagramme
- 6 Komponentendiagramme
- 7 Klassendiagramme
- 8 Sequenzdiagramme
- 9 Objektorientierte Methode

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

### Motivation



Wir müssen das Rad nicht neu erfinden!

Aber: Wir müssen das passende Rad finden!

Wir müssen keine neuen Methoden erfinden!

Aber: Welche Methode in welcher Ausprägung passt zum Projekt und zu den Projektmitarbeitern?

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

### Top-Down-Methode

- Vom Abstrakten zum Konkreten, vom Allgemeinen zum Speziellen.
- Beispiele für die Anwendung:
  - Schichten → Pakete → Komponenten → Klassen und ihre Beziehungen.
  - Modellierung von Zuständen in Zustandsautomaten: Zustand Ein/Aus eines Gerätes (Oberzustand), danach Modellierung der Unterzustände.
  - Identifizieren von Klassen und danach Festlegen der Attribute und Operationen.
  - Spezialisierung: Vom abstrakten Typ zu den konkreten Typen/Klassen.
  - Planung von Phasen und später Detailplanung der Aktivitäten.
- Vorteile und Nachteile
  - + Konzentration auf das Wesentliche möglich.
  - + Strukturelle Zusammenhänge werden leichter erkannt.
  - - Es wird ein hohes Abstraktionsvermögen benötigt.



Wo arbeiten Sie in Ihren Projekten nach der Top-Down-Methode?

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

## Bottom-Up-Methode

- Vom Konkreten zum Abstrakten, vom Speziellen zum Allgemeinen.
- Beispiele:
  - Objektorientierte Modellierung: Identifikation von Klassen und ihren Beziehungen  
→ Zusammenfassung zu Komponenten/Paketen → Bildung von Schichten.
  - Modellierung von Zuständen in Zustandsautomaten: Zuerst Modellierung der Unterzustände danach Zusammenfassen zu Oberzuständen.
  - Identifizieren von Attributen und Operationen danach Identifikation der Klassen.
  - Generalisierung: Vom konkreten Typ/Klasse zum abstrakten Typ.
  - Detailplanung für einzelne Aktivitäten, danach Zusammenfassen zu den Phasen.
- Vorteile und Nachteile
  - + Es ist eine konkrete Ausgangsbasis vorhanden.
  - + Bei Altsystemen ist dies oftmals die einzige Möglichkeit.
  - - Übergeordnete Strukturen/Abstraktionen sind schwierig zu erkennen.



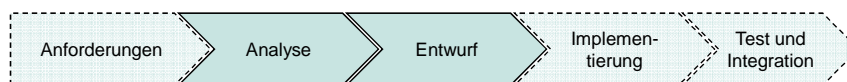
Wo arbeiten Sie in Ihren Projekten nach der Bottom-Up-Methode?

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

## Methode



Methoden dienen als Vorschrift für die Durchführung einer oder mehrerer Aktivitäten/Phasen eines Vorgehensmodells.

**Konzept:****Analyse-/Entwurfsmethode:**

datenorientiert

Datenmodellierung mit ERM (Entity Relationship Model)

funktionsorientiert

Strukturierte Analyse (SA)

Strukturiertes Design (SD)

Strukturierte Modellierung

objektorientiert

Objektorientierte Analyse (OOA)

Objektorientiertes Design (OOD)

Objektorientierte Modellierung (OOM)

Auch  
**klassische Methoden**  
genannt.



Methoden dienen der  
**Modellierung** nach einem  
bestimmten **Konzept**.

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

## Modelle in den unterschiedlichen Methoden



**Methoden setzen Modelle** ein. Ein Modell ist ein Abbild der Realität. Ein Modell besteht aus mehreren Sichten, die gekoppelt das System beschreiben.

Konzept:	Analyse-/Entwurfsmethode:	Modelle und Sichten:
datenorientiert	Datenmodellierung mit ERM (Entity Relationship Model)	<ul style="list-style-type: none"> <li>- ERM,</li> <li>- relationales Datenbankmodell, ...</li> </ul>
funktionsorientiert	<div>Strukturierte Analyse (SA)</div> <div>Strukturiertes Design (SD)</div> <div>Strukturierte Modellierung</div>	<ul style="list-style-type: none"> <li>- DFD: Datenflussdiagramm</li> <li>- PSPECs: Prozessspezifikationen</li> <li>- CFD: Kontrollflussdiagramm</li> <li>- CSPEC: Kontrollspezifikation</li> <li>- Entscheidungstabelle</li> <li>- Aufrufhierarchie, ...</li> </ul>
objektorientiert	<div>Objektorientierte Analyse (OOA)</div> <div>Objektorientiertes Design (OOD)</div> <div>Objektorientierte Modellierung (OOM)</div>	Unified Modelling Language (UML) <ul style="list-style-type: none"> <li>- Klassendiagramm</li> <li>- Anwendungsfalldiagramm</li> <li>- Aktivitätsdiagramm</li> <li>- Kommunikationsdiagramm, ...</li> </ul>

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

## Modelle in den unterschiedlichen Methoden



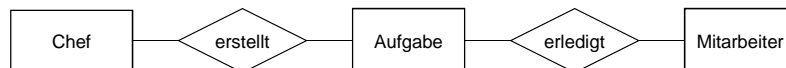
**Methoden setzen Modelle** ein. Ein Modell ist ein Abbild der Realität. Ein Modell besteht aus mehreren Sichten, die gekoppelt das System beschreiben.

Konzept:	Analyse-/Entwurfsmethode:	Modelle und Sichten:
datenorientiert	Datenmodellierung mit ERM (Entity Relationship Model)	<ul style="list-style-type: none"> <li>- ERM,</li> <li>- relationales Datenbankmodell, ...</li> </ul>
funktionsorientiert	<div>Strukturierte Analyse (SA)</div> <div>Strukturiertes Design (SD)</div> <div>Strukturierte Modellierung</div>	<ul style="list-style-type: none"> <li>- DFD: Datenflussdiagramm</li> <li>- PSPECs: Prozessspezifikationen</li> <li>- CFD: Kontrollflussdiagramm</li> <li>- CSPEC: Kontrollspezifikation</li> <li>- Entscheidungstabelle</li> <li>- Aufrufhierarchie, ...</li> </ul>
objektorientiert	<div>Objektorientierte Analyse (OOA)</div> <div>Objektorientiertes Design (OOD)</div> <div>Objektorientierte Modellierung (OOM)</div>	Unified Modelling Language (UML) <ul style="list-style-type: none"> <li>- Klassendiagramm</li> <li>- Anwendungsfalldiagramm</li> <li>- Aktivitätsdiagramm</li> <li>- Kommunikationsdiagramm, ...</li> </ul>

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

## Datenmodellierung mit ERM

- ERM = ältestes und allgemeinstes Modell zur strukturierten Modellierung von Daten.
- definiert 1970 – 1976 von Peter Chen:
  - taiwanischer Informatiker,
  - seit 1983 Professor der Informatik an der Louisiana State University,
  - zählt zu den Pionieren der Informatik.
- Beispiel Chen-Notation:



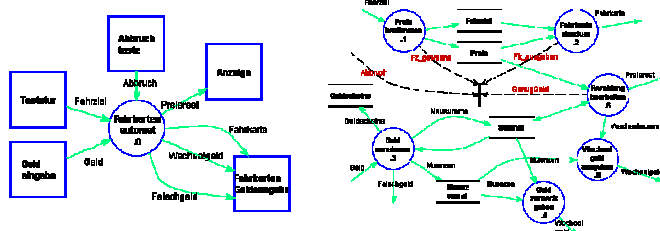
Bedeutung für nachfolgende Methoden:

- ERM-Methode wurde nachträglich in die Strukturierte Analyse aufgenommen.
- Objektorientierte Theorie von Rumbaugh basiert auf ERM.
- Eine Klasse entspricht einem Entitäts-Typ.
- ER-Diagramme und UML-Klassendiagramme sind in der Struktur einander ähnlich.

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

## Strukturierte Analyse (SA)

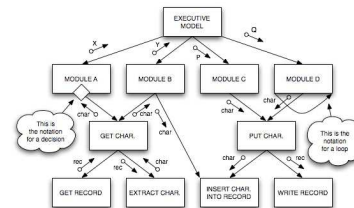
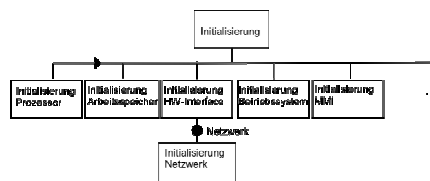
- Strukturierte Analyse (SA) ist eine Top-Down-Methode für die Analyse-Phase einer Software-Entwicklung.
- hauptsächlich von Tom DeMarco von 1970 – 1977 entwickelt:
  - 1940 in Pennsylvania, USA geboren,
  - viele Jahre in der Software-Entwicklung tätig,
  - viele bekannte Bücher (auch „Der Termin“) geschrieben.
- Hierarchische Zerlegung des Systems in immer einfachere Funktionen/Prozesse.
- Gleichzeitig wird eine Datenflussmodellierung durchgeführt.



## 2. Allgemeine Grundlagen - Methoden der SW-Technik

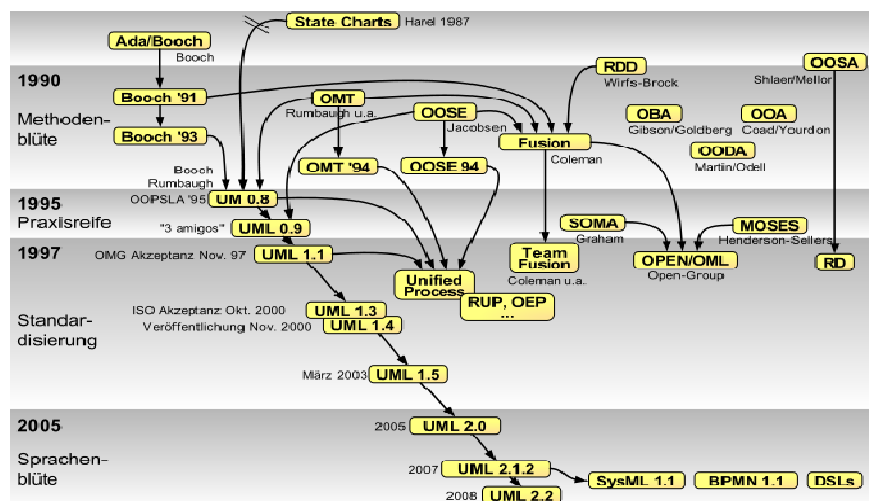
## Strukturiertes Design (SD)

- Schlägt eine Brücke zwischen der technologieneutralen Strukturierten Analyse und der eigentlichen Implementierung.
- entwickelt von
  - Edward Yourdon (geb. 1944)
  - und Larry Constantine (geb. 1943)
  - in den späten 1970ern.
- beschreibt – neben der reinen Funktionshierarchie – den Kontrollfluss in den Kontrollflussdiagrammen sowie die Wechselwirkungen zu übergeordneten Modulen in Structure-Chart-Diagrammen.



## 2. Allgemeine Grundlagen - Methoden der SW-Technik

## Entwicklung von Methoden für die Objektorientierung





## Agenda

- 1 Methoden der SW Technik
- 2 Allgemeine Grundlagen UML
- 3 Anwendungsfall-/ Use Case Diagramme
- 4 Aktivitätsdiagramme
- 5 Objekt- und Paketdiagramme
- 6 Komponentendiagramme
- 7 Klassendiagramme
- 8 Sequenzdiagramme
- 9 Objektorientierte Methode

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

### Was ist UML?

#### Definition:

UML ist eine **standardisierte Notationssprache** zur Beschreibung der **objektorientierten Modellierung** von Systemen.

#### Vorsicht:

Das objektorientierte Konzept und die objektorientierte Methodik beruhen nicht auf UML, sondern können mit Hilfe von UML notiert werden.



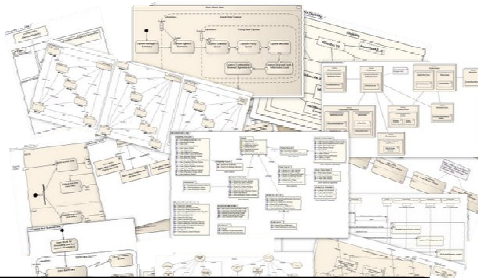
**Ziel der UML** ist es, die **Analyse** und den **Entwurf** von komplexen, **objektorientierten Software-Systemen standardisiert** und **leicht verständlich zu dokumentieren**.



## 1. Was ist die UML?

### Was ist die Unified Modeling Language (UML)?

- Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme
- von der *Object Management Group* (OMG) entwickelt und ist sowohl von ihr als auch von der ISO (ISO/IEC 19505 für Version 2.4.1[2]) standardisiert
- Aktuelle Version: UML 2.5
- Offizielle Seite: [www.uml.org](http://www.uml.org)



„Sichtbar“ von UML ist die **grafische Notation**, es gibt aber auch die Möglichkeit UML rein textuell zu formulieren (XML Metadata Interchange) XML.

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

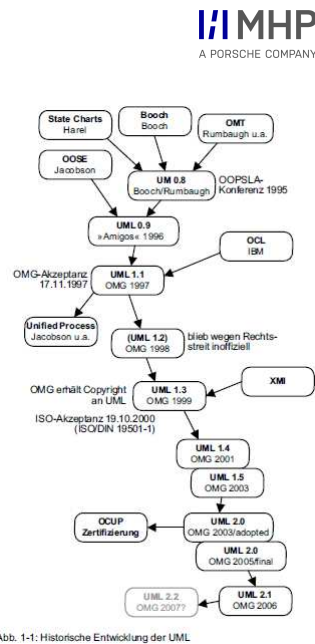
### Wie entstand die UML?

- UML entstand in den frühen 90er-Jahren mit dem Ziel die Methodenkriege, wie man objektorientiert modelliert, zu beenden (es gab über 50 Methoden).
- Die Väter der UML – Grady Booch, James Rumbaugh und Ivar Jacobson (auch als die 3 Amigos bekannt) –, die alle nach und nach bei der Firma Rational beschäftigt wurden, arbeiteten gemeinsam daran, eine gemeinsame Methode – die Unified Method – zu entwickeln.
- Da keine Einigung auf eine gemeinsame Methode möglich war, wurde – als bescheideneres, gemeinsames Ziel – eine grafische Modellierungssprache – die Unified Modelling Language – geboren.

## 1. Was ist die UML?

### Wozu UML?

- Kommunikation – gemeinsame Sprache sprechen
  - zum Auftraggeber
  - in Projektteams (anhand standardisierter Diagramme)
- Abstraktion
  - Code ist zwar präzise, aber zu detailliert
  - UML bietet Abstraktionen und unterschiedliche Sichten auf das zu erstellende System
- Durchgängigkeit
  - UML ermöglicht eine durchgängige und vollständige Systemdokumentation
  - bis hin zur Codegenerierung beim Einsatz von CASE-Tools
- Standard
  - zur Modellierung und Dokumentation objektorientierter Systeme
  - für die Entwicklung von objektorientierten CASE-Tools



© 2018 MHP Management- und IT-Beratung GmbH

37

## 2. Allgemeine Grundlagen - Methoden der SW-Technik

### Standardisierung der UML

- 1997: Die OMG (Object Management Group) nimmt UML 1.1 als Standard an.
- 1999: Veröffentlichung der Version 1.3.
- 2003: Veröffentlichung der noch heute für die Praxis relevante Version 1.5.

Standardisierungsarbeiten an der UML 2, erhebliche Erweiterungen z. B.:

- architektonische Änderungen (Superstructure / Infrastructure)
- standardisiertes Format zum Austausch von Modellen und Diagrammen zwischen Werkzeugen

- 2010: Veröffentlichung der Version 2.3



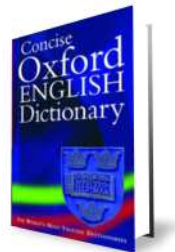
Die UML-Spezifikation sowie weitere Links und Informationen finden Sie unter:  
**<http://www.uml.org>**

© 2018 MHP Management- und IT-Beratung GmbH

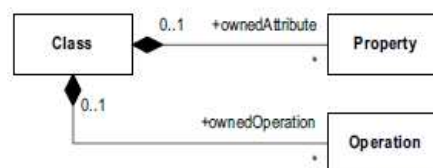
38

## 2. Allgemeine Grundlagen

## Das Metamodell der UML

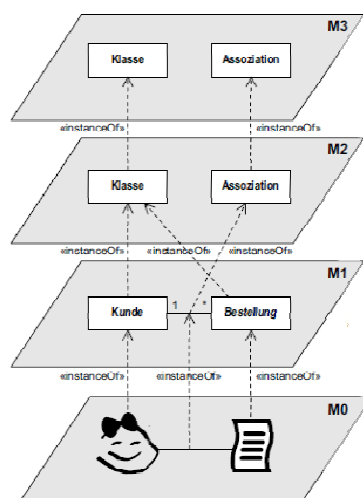


Vereinfachter Ausschnitt aus dem Metamodell



## 2. Allgemeine Grundlagen

## Das Metamodell der UML



Meta-Metamodell, MOF  
Modellierung z.B. des UML-Metamodells

UML-Metamodell,  
gewöhnlich verfügbare UML-  
Modellierungselemente

UML-Modell,  
gewöhnliche UML-Modellierungsebene

Laufzeitmodell  
Das Objekt selbst

## 2. Allgemeine Grundlagen

## Datentypen (1/2)

**UML unterscheidet in:**

- Einfache Datentypen (Data Type)
  - wie beispielsweise Geld, Bankverbindung etc.
  - Einfache Datentypen sind ein Oberbegriff für primitive Datentypen und Aufzählungstypen. Ein einfacher Datentyp ist ein Typ, dessen Werte keine Identität besitzen, d.h., zwei Instanzen eines Datentyps mit denselben Attributwerten können nicht voneinander unterschieden werden. Das ist ein wesentlicher Unterschied gegenüber Klassen.
- Aufzählungstypen (*Enumeration*)
  - wie beispielsweise Familienstand, Anrede etc.
  - Aufzählungstypen sind einfache Datentypen, deren Werte aus einer begrenzten Menge von Aufzählungsliteralen (*EnumerationLiteral*) stammen

## 2. Allgemeine Grundlagen

## Datentypen (2/2)

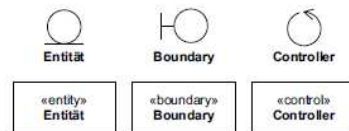
- Primitive Datentypen (PrimitiveType)
  - Ein primitiver Datentyp ist ein einfacher Datentyp ohne Strukturen. Die zum Typ gehörende Algebra ist außerhalb der UML beschrieben, z.B. die Addition ganzer Zahlen wird nicht als Operation eines primitiven Datentyps modelliert.
  - Die UML definiert selbst folgende primitive Datentypen vor:
    - *Integer*  
(Unendliche) Menge ganzer Zahlen: (... , -2, -1, 0, 1, 2, ...)
    - *Boolean*  
true, false
    - *String*  
Beliebige Zeichenkette, auch Umlaute u. Ä. sind zulässig.
    - *UnlimitedNatural*  
(Unendliche) Menge natürlicher Zahlen: (0, 1, 2, ...). Das Symbol für unendlich ist „\*“. Dieser Datentyp wird im Metamodell beispielsweise für die Definition von Multiplizitäten verwendet.

Zusammenfassend:

## 2. Allgemeine Grundlagen

## Stereotypen

- Formale Erweiterungen vorhandener Modellelemente des UML-Metamodells.
- Entsprechend der mit der Erweiterung definierten Semantik wird das Modellierungselement, auf das es angewendet wird, direkt semantisch beeinflusst.
- Klassifizieren die möglichen Verwendungen eines Modellelementes. Dabei handelt es sich nicht um die Modellierung von Metaklassen, vielmehr werden einem oder mehreren Modellelementen bestimmte gemeinsame Eigenheiten zugeschrieben.
- Visuelle und textuelle Stereotypen



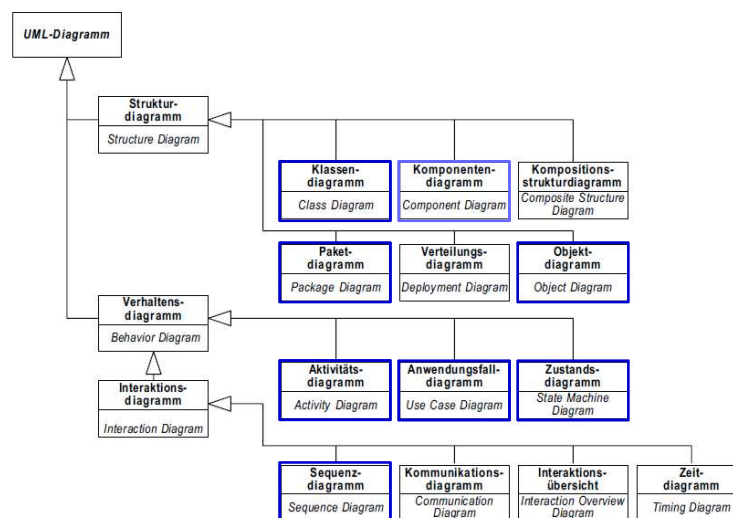
Standard-Stereotypen der UML

Stereotyp	UML-Element	Beschreibung
<call>	Abhängigkeit (Usage)	Aufrufabhängigkeit zwischen Operationen oder Klassen
<create>	Abhängigkeit (Usage)	Quellelement erstellt Instanzen des Zielelementes.
<instantiates>	Abhängigkeit (Usage)	Quellelement erstellt Instanzen des Zielelementes. (Achtung: Diese Beschreibung ist identisch mit der von <create>.)
<responsibility>	Abhängigkeit (Usage)	Quellelement ist verantwortlich für das Zielelement.
<send>	Abhängigkeit (Usage)	Quellelement ist eine Operation, Zielelement ist ein Signal, welches von der Operation gesendet wird.
<derive>	Abstraktion	Quellelement kann festw. durch eine Berechnung vom Zielelement abgeleitet werden.
<refine>	Abstraktion	Verfeinerungsbeziehung, z.B. zwischen einem Design- und zugehörigem Analyseelement.
<trace>	Abstraktion	zur Verfügbarkeit von Anforderungen
<script>	Artefakt	Skriptdatei (ausführbar auf einem Computer)
<auxiliary>	Klasse	Klassen, die andere Klassen («focus») unterstützen, Fokus-Klassen enthalten die primäre Logik.
<focus>	Klasse	siehe «auxiliary»
<implementation-class>	Klasse	Implementierungsklasse speziell für eine Programmiersprache, wobei ein Objekt nur zu einer Klasse gehören darf.
<metaclass>	Klasse	Klasse, deren Instanzen wiederum Klassen sind.
<type>	Klasse	Typen definieren eine Menge von Operationen und Attributen und sind in der Regel abstrakt.
<utility>	Klasse	Hilfsklassen sind Sammlungen von globalen Variablen und Funktionen, die zu einer Klasse zusammengefasst sind und als Klassenattribute-Operationen definiert sind.
<buildComponents>	Komponente	organisatorisch motivierte Komponente
<implements>	Komponente	Komponente, die keine Spezifikation enthält, sondern nur Implementierung.
<framework>	Paket	Paket, das Framework-Elemente enthält.
<modelLibrary>	Paket	Paket, das Modellelemente enthält, die in anderen Paketen wiederverwendet werden.
<create>	Verhaltens-eigenschaft (BehavioralFeature)	Eigenschaft, die Instanzen der Klasse erzeugt, zu der sie gehört (z.B. Konstruktor).
<destroy>	Verhaltens-eigenschaft (BehavioralFeature)	Eigenschaft, die Instanzen der Klasse zerstört, zu der sie gehört (z.B. Destruktor).

Abb. 2-7: Standardstereotypen der UML (Auswahl)

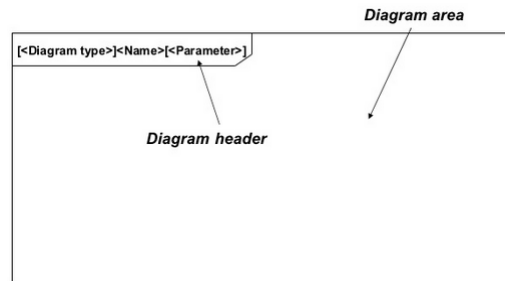
## 2. Allgemeine Grundlagen

## Diagramme im Überblick

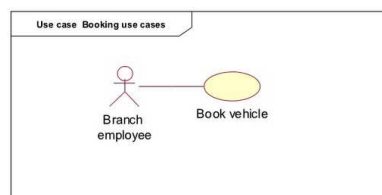


## 2. Allgemeine Grundlagen

## Basis Notation von Diagrammen



Beispiel:



## 2. Allgemeine Grundlagen

## Zusammenfassung

**Checkliste allgemeine Grundlagen:**

- Welche Datentypen unterscheidet die UML und wie heißen sie? ✓
- Ist die Kennzeichnung der Datentypen (z.B. «primitive») ein Stereotyp? ✓
- Was unterscheidet ein Datentyp von einer Klasse? ✓
- Was ist ein Stereotyp? Definiert es ein neues Metamodellelement? ✓
- Welche Standardstereotypen kennt die UML und was bedeuten sie? ✓
- Gibt es Schlüsselwörter, die keine Stereotypen sind? ✓
- Wie sieht die grundsätzliche grafische Darstellung eines Diagramms mit Diagrammkopf aus? ✓
- Welche Informationen stehen im Diagrammkopf? ✓

## Agenda

- 1 Methoden der SW Technik
- 2 Allgemeine Grundlagen UML
- 3 Anwendungsfall-/ Use Case Diagramme
- 4 Aktivitätsdiagramme
- 5 Objekt- und Paketdiagramme
- 6 Komponentendiagramme
- 7 Klassendiagramme
- 8 Sequenzdiagramme
- 9 Objektorientierte Methode

## 3. Use Case Diagramme

### Beispielsystem Bibliothek: Identifikation Anwendungsfälle

- Durch Beobachten der Abläufe werden die **Geschäftsprozesse** identifiziert.
- Für die spätere Programmierung relevant sind die so genannten **Anwendungsfälle**.



Die Anwendungsfälle werden gefunden, indem man die Geschäftsprozesse genau analysiert und festlegt, welche Teile durch das System automatisiert werden sollen.

- Der Geschäftsprozess „Buch ausleihen“, lässt sich wie folgt zerlegen:
    - Buchrecherche durchführen, ←
    - Buchverfügbarkeit prüfen, ←
    - Buch aus Regal holen, ←
    - Ausleiher identifizieren, ←
    - Buch für Ausleiher als entliehen buchen. ←
- Software-Unterstützung naheliegend  
→ Kandidaten für Anwendungsfälle
- Für die Anwendungsfälle ist zu prüfen, welche Objekte mit welchen Methoden hier unterstützen können.



### 3. Use Case Diagramme

#### Definition

- Ein Anwendungsfall (UseCase) spezifiziert eine Menge von Aktionen, die von einem System (subject) ausgeführt werden und zu einem Ergebnis führen, das üblicherweise von Bedeutung für einen Akteur oder Stakeholder ist.
- Der Akteur (Actor) ist definiert als eine Rolle, die sich außerhalb des Systems (subject) des zugehörigen Anwendungsfalles befindet und mit dem System im Kontext des Anwendungsfalles interagiert.
- Ein Anwendungsfalldiagramm beschreibt die Zusammenhänge zwischen einer Menge von Anwendungsfällen und den daran beteiligten Akteuren.

### 3. Use Case Diagramme

#### Beispielsystem Bibliothek: Methoden für die Klasse Buch



Wie kann ein Buch-Objekt die Anwendungsfälle unterstützen?

- Anwendungsfall „Buch-Recherche durchführen“
  - Ein einzelnes Buch-Objekt kann keine „Recherche durchführen“.
  - Ein einzelnes Buch-Objekt kann aber seine Daten für die Recherche bereitstellen:
    - `getTitel()`
    - `getISBN()`
- Anwendungsfall „Buchverfügbarkeit prüfen“
  - Ein Buch-Objekt muss wissen, wie viele Exemplare in der Bibliothek sind.  
→ neues Datenfeld `anzahlExemplare`
  - Ein Buch-Objekt ermöglicht die Abfrage der vorhandenen Exemplare über eine Methode `getAnzahlExemplare()`
  - ...

## 3. Use Case Diagramme

## Notation und Semantik

## ▪ Akteure:

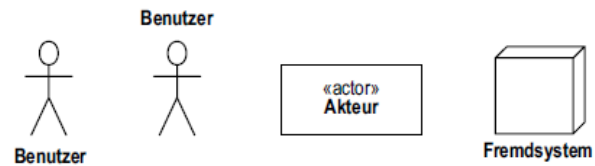


Abb. 2-118: Notation von Akteuren

## ▪ Use Case (Anwendungsfall):

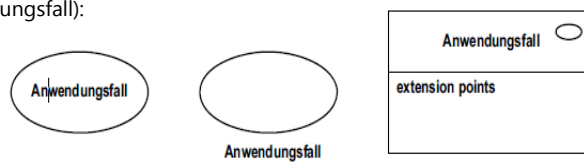


Abb. 2-119: Notation von Anwendungsfällen

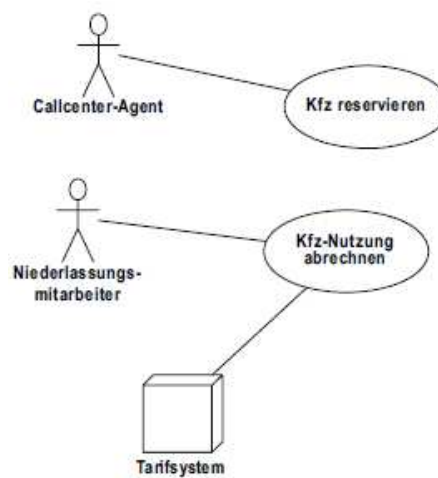
© 2018 MHP Management- und IT-Beratung GmbH

51

## 3. Use Case Diagramme

## Notation und Semantik

## ▪ Beispiel einer Anwendung:



© 2018 MHP Management- und IT-Beratung GmbH

52

## 3. Use Case Diagramme

## Notation und Semantik – Classifier

- Ein Subsystem:

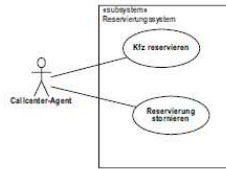


Abb. 2-120: Anwendungskontext

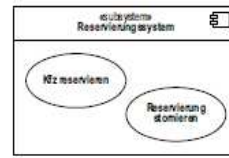


Abb. 2-121: Komponente mit zugehörigen Anwendungsfällen

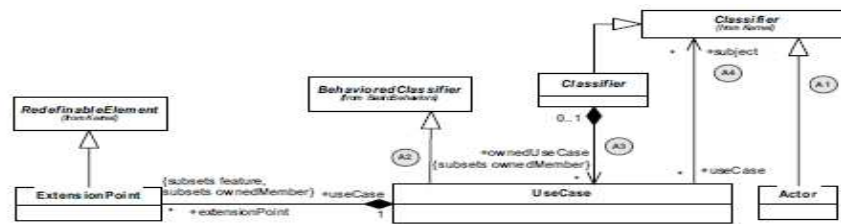
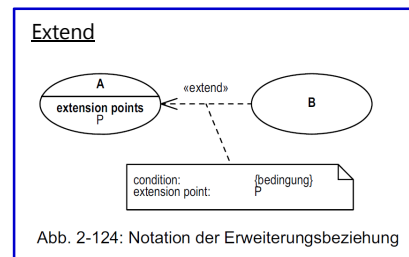
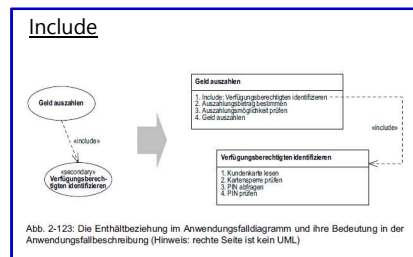


Abb. 2-122: Metamodell Anwendungsfall und Akteur

## 3. Use Case Diagramme

## Anwendungsfallbeziehungen (Include, Extend)

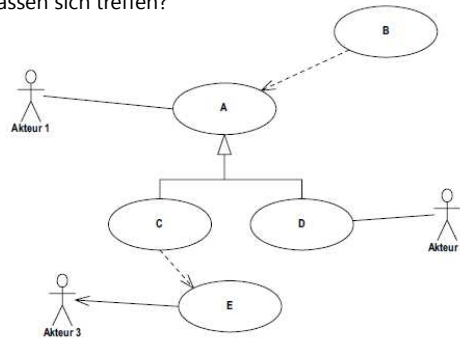
- Mit einer Enthältbeziehung (Include) wird ein Anwendungsfall in einen anderen Anwendungsfall eingebunden und logischer Teil von diesem.
- Mit einer Erweiterungsbeziehung (Extend) hingegen lässt sich ausdrücken, dass ein Anwendungsfall unter bestimmten Umständen und an einer bestimmten Stelle (dem sog. Erweiterungspunkt, engl. ExtensionPoint) durch einen anderen erweitert wird.



## 3. Use Case Diagramme

## Übung 1

- **Gegeben:** Anwendungsfalldiagramm mit Anwendungsfällen, Akteuren, Generalisierungsbeziehungen und Enthält- und/oder Erweiterungsbeziehungen gezeigt, die ohne Schlüsselwort notiert sind
- Welche Aussagen lassen sich treffen?



## 3. Use Case Diagramme

## Übung 2

**Kurzbeschreibung**

Der Anwendungsfall „Benutzer verwalten“ (Akteur: WiBe-Beauftragter/Admin) beschreibt die Verwaltung von Benutzern des Systems. Er dient dabei der Gruppierung der speziellen Anwendungsfälle und beinhaltet die Anwendungsfälle:

- Kennwort ändern
- Benutzer anlegen,
- Benutzer löschen sowie
- Benutzer bearbeiten.



Das Kennwort ändern können des Weiteren folgende Akteure:

- Mitarbeiter
- Projektleiter
- Controller
- Katalog-Autor

>> **Bitte erstellt das dazugehörige Use-Case-Diagramm**

### 3. Use Case Diagramme

#### Zusammenfassung

##### Checkliste Akteure und Anwendungsfälle:

- Was ist die Oberklasse eines Akteurs?
- Was ist die Oberklasse eines Anwendungsfalles?
- Wer kann der Besitzer eines Anwendungsfalles sein?
- Was ist das Subjekt eines Anwendungsfalles?
- Welche Notationsformen gibt es für einen Anwendungsfall?
- Welche Notationsformen gibt es für einen Akteur?
- Kann nur der menschliche Benutzer ein Akteur sein?

##### Checkliste Anwendungsfallbeziehungen

- In welche Richtung zeigt die Exclude-Beziehung?
- In welche Richtung zeigt die Include-Beziehung?
- Wie viele Erweiterungspunkte kann ein Anwendungsfall definieren?
- Muss eine Erweiterungsbeziehung eine Bedingung angeben?
- Muss der enthaltene Anwendungsfall einen Akteur haben?
- Muss der erweiternde Anwendungsfall einen Akteur haben?

#### Agenda

- 1 Methoden der SW Technik
- 2 Allgemeine Grundlagen UML
- 3 Anwendungsfall-/ Use Case Diagramme
- 4 Aktivitätsdiagramme
- 5 Objekt- und Paketdiagramme
- 6 Komponentendiagramme
- 7 Klassendiagramme
- 8 Sequenzdiagramme
- 9 Objektorientierte Methode

## 4. Aktivitätsdiagramme

## Definition

- Die Aktivität (Activity) beschreibt auf Basis von Kontroll- und Objektflussmodellen die Ablaufreihenfolge von Aktionen. Eine Aktivität enthält Kanten und Aktivitätsknoten, speziell Aktionen
- Eine Aktion (Action) ist im Metamodell eine abstrakte Klasse. Die konkreten Unterklassen werden auch in der UML spezifiziert. Sie sind in den so genannten Action Semantics beschrieben

- Mit Aktivitätsdiagrammen können Abläufe, wie sie beispielsweise in Anwendungsfällen (Use Cases) beschrieben sind, grafisch dargestellt werden. Mit natürlicher Sprache werden gewöhnlich nur sehr einfache Abläufe verständlich beschrieben, mit Aktivitätsdiagrammen hingegen ist es möglich, auch sehr komplexe Abläufe mit vielen Ausnahmen, Varianten, Sprüngen und Wiederholungen noch übersichtlich und verständlich darzustellen.

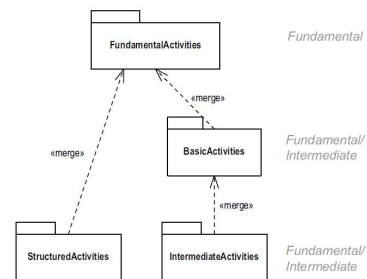
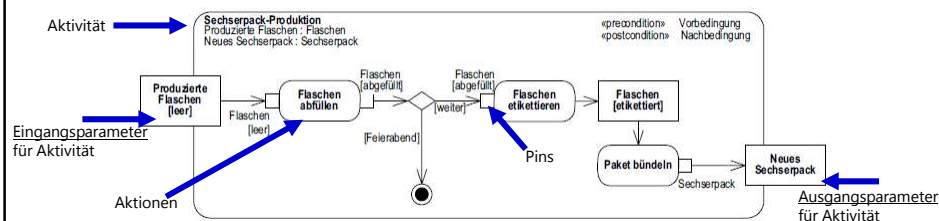


Abb. 3-107: Paketstruktur Aktivitätsdiagramme (activity diagram)

## 4. Aktivitätsdiagramme

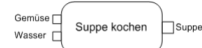
## Notation und Semantik (1/6)

- Eine Aktivität wird durch ein Rechteck mit abgerundeten Ecken dargestellt. In dem Rechteck befinden sich die Knoten und Kanten der Aktivität.
- Links oben im Rechteck steht der Name der Aktivität.



In der Aktivität befinden sich verschiedene Arten von Knoten und Kanten:

- Die Rechtecke mit den abgerundeten Ecken sind Aktionen.
- Die kleinen Rechtecke an den Aktionen sind so genannte Pins. Sie stellen die Eingangs bzw. Ausgangsparameterwerte für die Aktionen bereit.

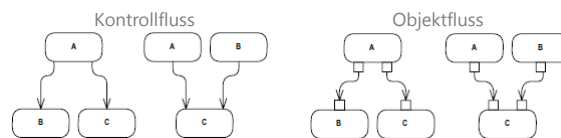


## 4. Aktivitätsdiagramme

## Notation und Semantik (2/6)

- Basiert auf dem Token-Fluss. >> Zwischen den Knoten fließen über die Kanten Kontroll- oder Objekt-Token. Dabei gelten folgende elementare Regeln:
  - Eine Aktion kann erst starten, wenn an allen eingehenden Kanten Token zur Verfügung stehen (implizite Synchronisation; Und-Semantik).
  - Wenn eine Aktion terminiert, wird an allen ausgehenden Kanten ein Token zur Verfügung gestellt (implizites Splitting; Und-Semantik).
  - Ein Token fließt von einer Aktion zur nächsten,
    - wenn an der ausgehenden Aktion ein Token zur Verfügung gestellt wird,
    - die hinführende Aktion bereit ist, das Token aufzunehmen, und
    - wenn die Regeln der Kante den Token-Fluss zulassen.
- Es gibt zwei Arten von Kanten: Kontrollfluss- und Objektflusskanten

Und-Semantik bei:



© 2018 MHP Management- und IT-Beratung GmbH

61

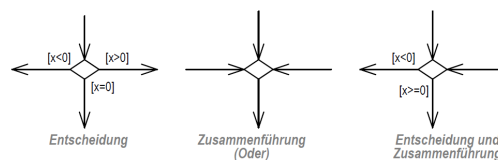
## 4. Aktivitätsdiagramme

## Notation und Semantik (3/6)

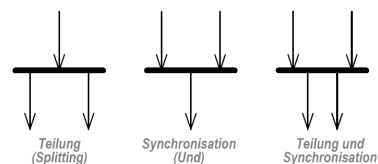
- Start-(Initial Node) und Endknoten (ActivityFinalNode):

 Startknoten      Endknoten      Ablaufende
 

- Entscheidungen (MergeNode):



- Synchronisation/Teilung:



© 2018 MHP Management- und IT-Beratung GmbH

62

## 4. Aktivitätsdiagramme

## Notation und Semantik (4/6)

- Signal-Notation:
  - Signal senden: Sender erstellt aus Eingabedaten ein Signal für den Ereignisempfänger
  - Ereignis empfangen: Gegenstück zum Signal senden; Ablauf läuft erst weiter wenn Ereignis eintrifft
  
- Partition-Notation:
  - Sind mehrere Personen/Systeme an Aktionen beteiligt, gibt es die sog. Zuständigkeitsbahnen (Partitions)
  - Hierbei wird ersichtlich, in welchen Verantwortungsbereich die Aktionen fallen.
  - Auch bekannt als Swimlane-Darstellung

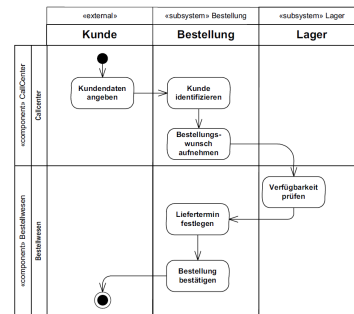
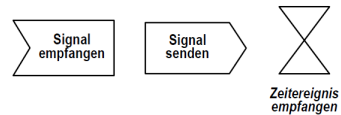
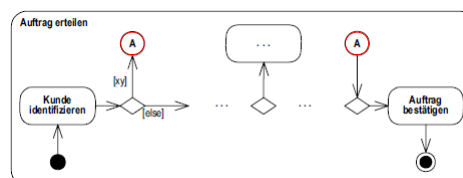


Abb. 3-120: Beispiel Partition

## 4. Aktivitätsdiagramme

## Notation und Semantik (5/6)

- Connector Edges
  - In Aktivitätsdiagrammen kann es vorkommen, dass eine Kante quer durch das ganze Diagramm gezeichnet werden muss. Damit das Diagramm übersichtlich bleibt, können so genannte Konnektoren eingeführt werden. Sie splitten eine Kante auf und haben nur Auswirkungen auf das Diagramm und nicht auf das zugrunde liegende Modell.





## 4. Aktivitätsdiagramme

## Übung

**Kurzbeschreibung**

Um den prinzipiellen Ablauf der Erstellung einer WiBe zu verstehen und eine Gesprächsbasis zu haben, soll dieser in Form eines Aktivitätsdiagramms dargestellt werden.

Ein Kollege der Fachabteilung versucht den Prozess zu erläutern:

*„Bevor wir richtig loslegen können, muss überprüft werden, ob für den vorliegenden Fall bereits ein Kriterienkatalog existiert. Wenn nicht, muss einer vom Mitarbeiter erstellt werden. Wenn dieser erstellt wurde, kann zum einen ein WiBe-Projekt vom Projektleiter im System angelegt werden und vom Mitarbeiter die sogenannten Richtwerttabellen befüllt werden. Der Projektleiter muss außerdem nach dem Anlegen Controlling&Reporting durchführen und gleichzeitig die Kriterien auswählen und diese anschließend belegen. Während des Belegens gibt es weiterhin die Möglichkeit weitere Kriterien im System auszuwählen, um sie zu belegen. Das wäre es im Groben und Ganzen.“*

>> Bitte erstellt das dazugehörige Aktivitätsdiagramm



## 4. Aktivitätsdiagramme

## Zusammenfassung:

**Checkliste Aktivitätsdiagramme:**

- Was stellt eine Aktivität dar?
- Was stellt eine Aktion dar?
- Welche Arten von Aktivitätsknoten gibt es?
- Was sind Pins?
- Welche Arten von Kanten gibt es?
  
- Welche Voraussetzung muss erfüllt sein, damit eine Aktion ausgeführt werden kann?
- An welchen ausgehenden Kanten werden Token zur Verfügung gestellt, wenn eine Aktion terminiert?
  
- Was bedeuten mehrere ausgehende Kanten aus einem Startknoten?
- Wie viele eingehende Kanten darf ein Startknoten haben?
- Was bedeuten mehrere eingehende Kanten in einen Endknoten?
- Was bedeutet das Verhalten, das eine Entscheidung spezifizieren kann?
  
- Wie viele Pins kann eine Aktion besitzen?
- Was geschieht an den Eingangsparametern einer Aktivität, wenn diese aufgerufen wird?

## Agenda




- 1 Methoden der SW Technik
- 2 Allgemeine Grundlagen UML
- 3 Anwendungsfall-/ Use Case Diagramme
- 4 Aktivitätsdiagramme
- 5 Objekt- und Paketdiagramme
- 6 Komponentendiagramme
- 7 Klassendiagramme
- 8 Sequenzdiagramme
- 9 Objektorientierte Methode

## 5. Objekt- und Paketdiagramme

### Das Denken in Objekten und Klassen

- ist von zentraler Bedeutung für das objektorientierte Konzept
- begleitet alle Phasen der SW-Entwicklung
- beginnt mit dem ersten Nachdenken über ein mögliches SW-System

Möchte man Aufgaben, die bisher manuell erledigt wurden, durch den Einsatz eines EDV-Systems unterstützen, so stellt man sich folgende Fragen:

-  Mit welchen Objekten hat man es beim Erledigen der Aufgaben zu tun?
-  Welche Eigenschaften dieser Objekte sind im Rahmen der zu erledigenden Aufgaben von Bedeutung?
-  Was wird mit den Objekten gemacht?

## 5. Objekt- und Paketdiagramme

## Beispielsystem Bibliothek: Objekte finden durch Beobachten



- viele „**Buch-Objekte**“
- werden ausgeliehen,
- werden zurückgegeben,
- ...



- Personen, die Bücher mitnehmen und zurückbringen
- besitzen einen Bibliotheksausweis
- sind in der Bibliothek registriert
- → „**Ausleiher-Objekte**“



Neben existenten Gegenständen in der realen Welt (wie z. B. Bücher) können auch Wesen (wie z. B. Ausleiher) oder Konzepte (wie z. B. Versicherungsverträge) relevante Objekte darstellen.

## 5. Objekt- und Paketdiagramme

## Beispielsystem Bibliothek: Klassen finden

**Klassifikation:** Gleichartige Objekte werden zu einer Klasse zusammengefasst!



- viele „**Buch-Objekte**“
- → Abbildung durch die Klasse `Buch`



- viele „**Ausleiher-Objekte**“
- → Abbildung durch die Klasse `Ausleiher`



Klassen stellen die Baupläne für Objekte dar. Die **Klassen** sind die **Datentypen**, die **Objekte** die **Variablen** (Instanzen) dieser Datentypen. Ein Objekt wird gemäß dem Bauplan einer Klasse erzeugt.

## 5. Objekt- und Paketdiagramme

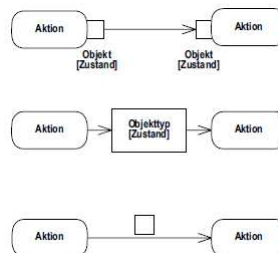
## Definition Objektdiagramme

- Ein Objektknoten (*ObjectNode*) gibt an, dass ein Objekt oder eine Menge von Objekten zu einem bestimmten Zeitpunkt in einem Ablauf vorhanden ist.
- Objektknoten können als ein- oder ausgehende Parameter in Aktivitäten (Aktivitätsparameter (*ActivityParameterNode*)) oder als ein- oder ausgehende Parameter von Aktionen (Pin) verwendet werden.
- Ein *Objektfluss* (*ObjectFlow*) ist wie ein Kontrollfluss eine spezielle Kante, bei der jedoch Objekte (*Objekt-Token*) transportiert werden.

## 5. Objekt- und Paketdiagramme

## Notation und Semantik - Objektdiagramme

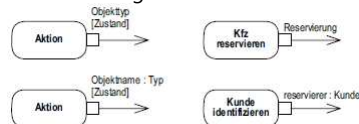
- Mit dem Objektfluss wird ausgedrückt, dass die entsprechenden Objekte von den Aktionsknoten vorausgesetzt bzw. erzeugt oder verändert werden. Eine Aktion startet erst dann, wenn die benötigten Objekte vorliegen. Am Ende der Aktion werden die neuen oder geänderten Objekte bereitgestellt.
- Ein Pin ist immer einer Aktion zugeordnet. Es wird zwischen Eingangs- und Ausgabepin unterschieden (*InputPin*, *Output-Pin*). Eine Aktion kann beliebig viele Pins haben
- Verschiedene Notationen:



## 5. Objekt- und Paketdiagramme

## Notation und Semantik – Beschriftung der Objektknoten

- Ein Objektknoten kann sowohl mit dem Namen des Objekttyps als auch mit dem Objektnamen und der zusätzlichen Angabe des Typs versehen werden.
- Zustände von Objekten müssen nicht modelliert werden; die Notation von Objektzuständen ist lediglich eine Möglichkeit, solche Sachverhalte hervorzuheben, soweit dies von besonderer Bedeutung ist.



- Es ist nicht zwingend erforderlich, dass ein Eingabepin eingehende Kanten und ein Ausgabepin ausgehende Kanten hat. Ein Ausgabepin ohne weiterführende Kanten drückt aus, dass die zugehörige Aktion einen Ausgabeparameter hat, der aber für den weiteren Ablauf der Aktivität keine Rolle spielt.



## 5. Objekt- und Paketdiagramme

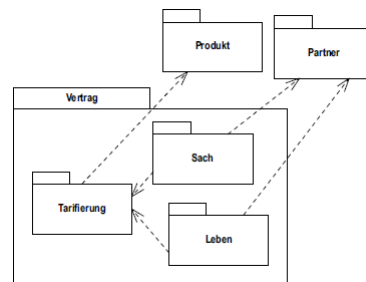
## Definition Paketdiagramme

- Ein Paket (*Package*) ist eine Ansammlung von Modellelementen beliebigen Typs, mit denen das Gesamtmodell in kleinere überschaubare Einheiten gegliedert wird.
- Ein Paket definiert einen Namensraum, d.h., innerhalb eines Paketes müssen die Namen der enthaltenen Elemente eindeutig sein. Jedes Modellelement kann in anderen Paketen referenziert werden, gehört aber nur zu höchstens einem (Heimat-)Paket. Pakete können wiederum Pakete beinhalten.
- Pakete können verschiedene Modellelemente enthalten, beispielsweise Klassen und Anwendungsfälle. Sie können hierarchisch gegliedert werden, d.h. ihrerseits wieder Pakete enthalten.

## 5. Objekt- und Paketdiagramme

### Notation und Semantik - Paketdiagramme

- Ein Paket wird in Form eines Aktenregisters dargestellt. Innerhalb dieses Symbols steht der Name des Paketes. Werden innerhalb des Symbols Modellelemente angezeigt, steht der Name auf der Registerlasche, anderenfalls innerhalb des großen Rechteckes.
- Oberhalb des Paketnamens können Stereotypen notiert werden.



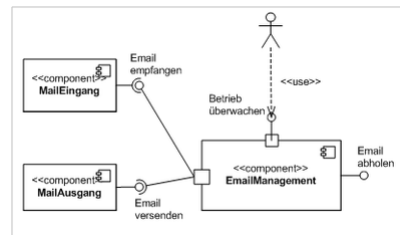
## Agenda

- 1 Methoden der SW Technik
- 2 Allgemeine Grundlagen UML
- 3 Anwendungsfall-/ Use Case Diagramme
- 4 Aktivitätsdiagramme
- 5 Objekt- und Paketdiagramme
- 6 Komponentendiagramme
- 7 Klassendiagramme
- 8 Sequenzdiagramme
- 9 Objektorientierte Methode

## 6. Komponentendiagramme

## Definition

- Es zeigt eine bestimmte Sicht auf die Struktur des modellierten Systems.
- Die Darstellung umfasst dabei typischerweise Komponenten mit deren Schnittstellen bzw. Ports. Es zeigt auch, wie Komponenten über Abhängigkeitsbeziehungen und Konnektoren miteinander verbunden sind.
- Um das Innere einer Komponente darzustellen, zeigt ein Komponentendiagramm oft Notationselemente, die sonst vor allem in Klassen- oder Kompositionsstrukturdiagrammen angezeigt werden, zum Beispiel Klassen oder Parts.
- Beispiel:



© 2018 MHP Management- und IT-Beratung GmbH

77

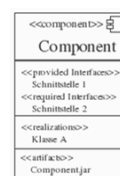
## 6. Komponentendiagramme

## Notation und Semantik

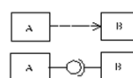
- Eine Komponente ist eine spezielle strukturierte Klasse (Class). Sie kann also auch Attribute und Operationen besitzen.
- Über dem Namen der Komponente steht das Schlüsselwort *component*.
- Notation einer Komponente (externe Sicht):



- Notation einer Komponente (interne Sicht):



- Notation von Beziehungen:

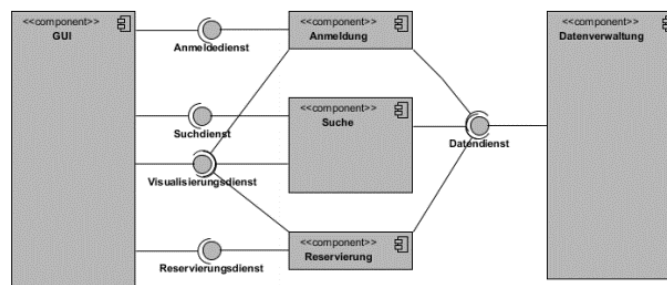


© 2018 MHP Management- und IT-Beratung GmbH

78

## 6. Komponentendiagramme

## Beispiel - Seminarverwaltung



© 2018 MHP Management- und IT-Beratung GmbH

79

## 6. Komponentendiagramme

## Übung

**Kurzbeschreibung**

Folgende Komponenten sollen in einem Komponentendiagramm dargestellt werden:

- Sicherheit
  - Diese Komponente umfasst alle sicherheitstechnischen Aspekte des Systems. Das System verfügt über eine eigene Benutzerverwaltung und ein einfaches Rollenkonzept.
- Projektverwaltung
  - Die Erstellung einer Wirtschaftlichkeitsbetrachtung erfolgt in Form von Projekten. Hier wird auf einen Kriterienkatalog sowie auf die Benutzerverwaltung zugegriffen.
- Kriterienkatalog
  - Für ähnliche Projekte wird in der Regel derselbe Kriterienkatalog verwendet. Diese Komponente enthält alle notwendigen Funktionen zur Verwaltung der Kriterienkataloge.
- Controlling & Reporting
  - Im Controlling&Reporting können verschiedene Projekte die auf demselben Kriterienkatalog basieren miteinander verglichen und ausgewertet werden.
- Drucken
  - Die Komponente Drucken kapselt alle Funktionen des Druckens. Jede Ansicht des Reportings und Controllings muss dabei druckbar sein.

>> **Bitte erstellt das dazugehörige Komponentendiagramm (externe Sicht)**

© 2018 MHP Management- und IT-Beratung GmbH

80



## 6. Komponentendiagramme



### Zusammenfassung:

- Was ist eine Komponente? ✓
- Welche Arten von Schnittstellen können Komponenten haben? ✓

## Agenda



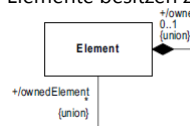
- 1 Methoden der SW Technik
- 2 Allgemeine Grundlagen UML
- 3 Anwendungsfall-/ Use Case Diagramme
- 4 Aktivitätsdiagramme
- 5 Objekt- und Paketdiagramme
- 6 Komponentendiagramme
- 7 Klassendiagramme
- 8 Sequenzdiagramme
- 9 Objektorientierte Methode

## 7. Klassendiagramme

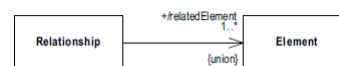
## Basiskonzepte

**Basisklasse der UML**

Die Oberklasse heit *Element* und hat die Eigenschaft, andere Elemente besitzen zu knnen.

**Basisklasse Beziehung**

Die Beziehung (Relationship) ist ein abstraktes Konzept, um Elemente in Beziehungen zueinander zu bringen.

**Gerichtete Beziehung**

Oft gibt es ein Element, das etwas anbietet, und ein anderes Element, das etwas haben mchte

**Notation Kommentar**

Neben den elementaren abstrakten Konzepten gibt es hier auch ein ganz konkretes Element: den Kommentar (*Comment*). Ein Synonym ist Notiz.



## 7. Klassendiagramme

## Beispielsystem Bibliothek: Methoden fr die Klasse Buch



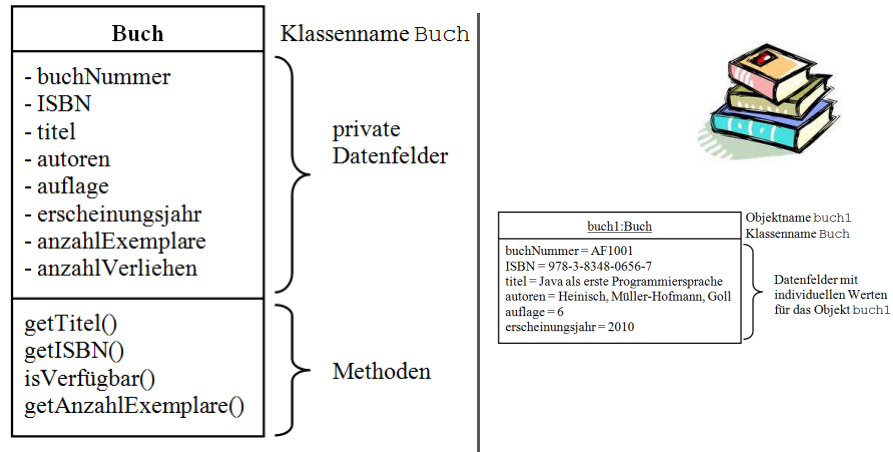
Wie kann ein Buch-Objekt die Anwendungsflle untersttzen?

- Anwendungsfall „Buch-Recherche durchfhren“
  - Ein einzelnes Buch-Objekt kann keine „Recherche durchfhren“.
  - Ein einzelnes Buch-Objekt kann aber seine Daten fr die Recherche bereitstellen:
    - `getTitel()`
    - `getISBN()`
- Anwendungsfall „Buchverfgbarkeit prfen“
  - Ein Buch-Objekt muss wissen, wie viele Exemplare in der Bibliothek sind.  
→ neues Datenfeld `anzahlExemplare`
  - Ein Buch-Objekt ermglicht die Abfrage der vorhandenen Exemplare ber eine Methode `getAnzahlExemplare()`
  - ...

## 7. Klassendiagramme

## Beispielsystem Bibliothek: UML-Notation der Klasse mit Methoden

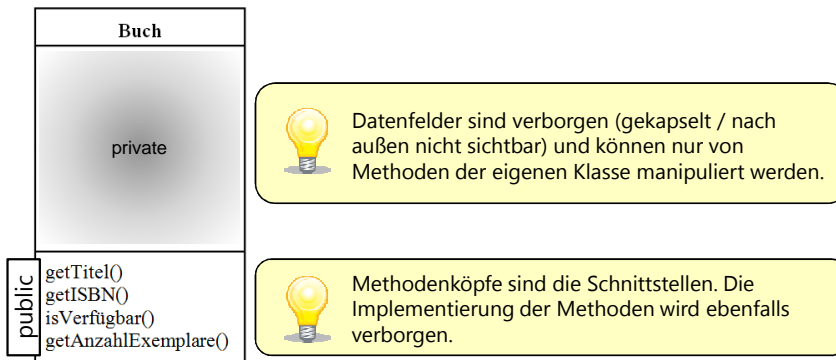
UML-Notation der Klasse Buch nach einigen Überlegungen:



## 7. Klassendiagramme

## Kapselung / Information Hiding / Geheimnisprinzip

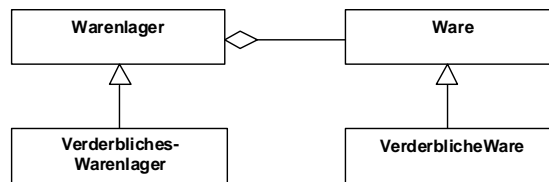
- Abschotten der internen Implementierung vor direktem externen Zugriff.
- Zugriff auf Daten und private Methoden nur über explizit definierte Schnittstelle.
- Aufrufer ist unabhängig von Implementierungsdetails.



## 7. Klassendiagramme

## Wiederverwendung

- Durch Vererbung:  
Die spezialisierten Klassen erben den Programmcode der Vaterklassen.
- Durch Aggregation:  
Verwendung von fertigen Komponenten, Modulen und Bibliotheken.
- Durch Polymorphie:  
Wiederverwendung gesamter Programmsysteme → Prinzip von Frameworks.  
Spezifische Eigenschaften werden durch Ableiten und Überschreiben von Methoden hinzugefügt.



## 7. Klassendiagramme

## Objektorientierung und die Prinzipien der SW-Technik



Wie werden die Prinzipien der SW-Technik in der Objektorientierung abgedeckt?

- Prinzip der Abstraktion:  
Abstrakte Datentypen, Schnittstellen, Klassen und Pakete.
- Prinzip der Kopplung und Bindung (Low Coupling, High Cohesion):  
Anwendung auf Klassen und Pakete.
- Prinzip der Hierarchisierung:  
Vererbungshierarchien und Zerlegungshierarchien.
- Prinzip der Modularisierung:  
Klassen und Pakete.
- Geheimnisprinzip:  
Klassen (Objekte) und Pakete.



Die Prinzipien der SW-Technik lassen sich optimal auf die Objektorientierung anwenden, bzw. sind Bestandteil der Objektorientierung.

## 7. Klassendiagramme



## Namensräume (Namespaces) (1/2)

- Ein benennbares Element (**NamedElement**) ist ein Element, das einen Namen und eine definierte Sichtbarkeit (*public*, *private*, *protected*, *package*) haben kann. Der Name des Elementes und die Sichtbarkeit sind optional.
- Ein Namensraum (**Namespace**) ist ein benennbares Element, das benennbare Elemente enthält, die eindeutig über ihre Namen identifizierbar sind.
- Das Importieren eines Elementes (**ElementImport**) ist eine Beziehung zwischen einem Namensraum und einem paketierbaren Element in einem anderen Namensraum. Das referenzierte Element kann dann direkt über seinen (unqualifizierten) Namen angesprochen werden.
- Das Importieren eines Paketes (**PackagelImport**) ist semantisch äquivalent mit dem Importieren jedes einzelnen Elementes des Paketes. Ein Aliasname ist hier natürlich nicht möglich.

## 7. Klassendiagramme



## Namensräume (Namespaces) (2/2)

Sichtbarkeit:

„+“ *public*„#“ *protected*„-“ *private*„~“ *package*

Folgende Importmöglichkeiten bestehen:

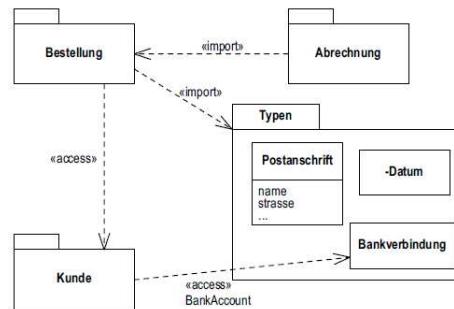
- «import»  
Sichtbarkeit ist öffentlich (**public**). In Abb. 2-14 ist beispielsweise Postanschrift in Bestellung sichtbar. Der öffentliche Import ist eine transitive Beziehung.
- «access»  
Nichtöffentliche Sichtbarkeit (**private**). In Abb. 2-14 ist beispielsweise Kunde nur in Bestellung sichtbar, aber nicht in Abrechnung. Der private Import ist nicht transitiv.

## 7. Klassendiagramme

## Namensräume (Namespaces) - Beispiel

Folgende Aussagen können zu diesem Diagramm getroffen werden:

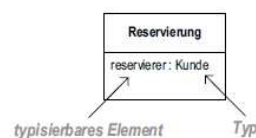
- *Datum* ist ein privates Element im Paket *Typen*.
- *Postanschrift* und *Bankverbindung* sind im Paket *Bestellung* sichtbar (Paketimport). *Datum* ist in *Bestellung* nicht sichtbar, da Elemente mit privater Sichtbarkeit nicht importiert werden.
- Der öffentliche Import ist transitiv. Daher sind *Postanschrift* und *Bankverbindung* auch im Paket *Abrechnung* sichtbar.
- Die Elemente des Paketes *Kunde* sind im Paket *Bestellung* sichtbar (privater Paketimport), aber nicht im Paket *Abrechnung*.
- Im Paket *Kunde* ist die *Bankverbindung* unter dem Namen *BankAccount* sichtbar (Alias).



## 7. Klassendiagramme

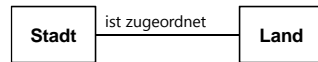
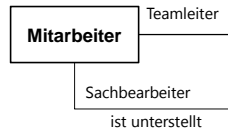
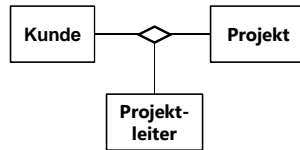
## Typisierbare Elemente (TypedElements)

- Ein typisierbares Element (**TypedElement**) ist ein benennbares Element (*NamedElement*), das einen Typ haben kann. Zum Beispiel sind Attribute oder Parameter typisierbare Elemente.
- Ein Typ (**Type**) spezifiziert eine Menge von Werten eines typisierbaren Elementes. Zum Beispiel sind einfache Datentypen und Klassen Typen.



## 7. Klassendiagramme

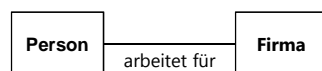
## Assoziationen

**Binäre Assoziation:****Reflexive Assoziation:****Ternäre Assoziation:****Wichtige Eigenschaften einer Assoziation:**

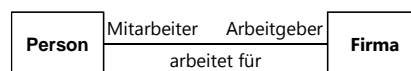
- Name der Assoziation
- Rollen (Endnamen) am Ende der Assoziation
- Multiplizitäten an beiden Enden
- Navigation
- Assoziationsklasse

## 7. Klassendiagramme

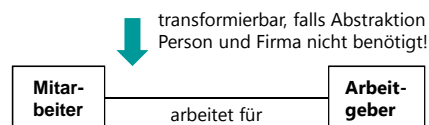
## Assoziationen

**Name einer Assoziation:****Leserichtung:**

- von oben nach unten
- von links nach rechts
- mit Lese Pfeil

**Endnamen einer Assoziation:**

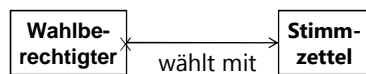
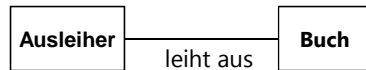
- Klasse ist als Rolle an der Assoziationsbeziehung beteiligt.
- Rollennamen kann am Endpunkt der Assoziation (als Endnamen) notiert werden.
- Bei Verwendung von Rollennamen kann der Name der Assoziation entfallen.



## 7. Klassendiagramme

## Assoziationen

## Navigation:



## Erläuterung:

- Eine Assoziation (Strich zwischen zwei Klassen) ist bidirektional oder unspezifiziert.

**Beispiel:** Ein Ausleiher leiht ein Buch aus und das Buch wird durch den Ausleiher ausgeliehen.

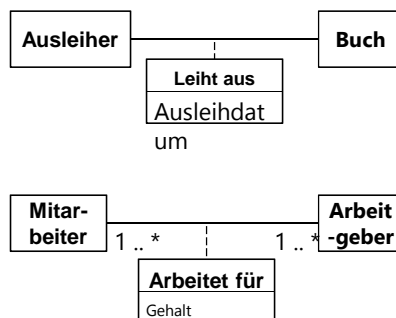
- Die Navigierbarkeit kann auf eine Richtung eingeschränkt werden → unidirektionale Navigierbarkeit.

**Beispiel:** Eine Person als Wahlberechtigter kann mit Hilfe eines Stimmzettels wählen, aber vom Stimmzettel aus soll es nicht möglich sein, zurück zur Person zu gelangen.

## 7. Klassendiagramme

## Assoziationen

## Assoziationsklasse:



## Erläuterung:

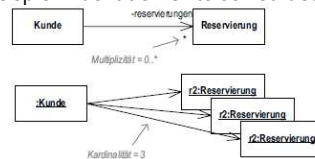
- Die Assoziationsklasse fällt weg, wenn die Beziehung gestrichen wird.
- Eine Assoziationsklasse gehört also zur Assoziation und beschreibt diese näher.
- Enthält die Assoziationsklasse keine Operationen, so spricht man auch von einem Assoziationsattribut.
- Der Name der Assoziationsklasse sollte mit dem Namen der Assoziation übereinstimmen bzw. sich daraus direkt ableiten lassen.



## 7. Klassendiagramme

## Multiplizitäten (Multiplicities)

- Eine Multiplizität (**MultiplicityElement**) ist die Definition eines Intervalls positiver ganzer Zahlen der erlaubten Kardinalitäten (**Cardinality**). Eine Kardinalität ist die Anzahl der Elemente in einer Menge.
- Die Begriffe Multiplizität und Kardinalität werden häufig synonym verwendet. Das ist allerdings verkehrt. Das Beispiel macht den Unterschied deutlich:



## Beispiele für Multiplizitäten:

0..1	null oder eins
1	genau eins (Kurzschreibweise für 1..1)
*	null bis beliebig viele (Kurzschreibweise für 0..*)
1..*	eins bis beliebig viele
5..3	nicht erlaubt; der untere Wert muss kleiner gleich dem oberen Wert sein
-1..0	nicht erlaubt; die Werte müssen alle positiv sein

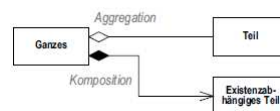
© 2018 MHP Management- und IT-Beratung GmbH

97

## 7. Klassendiagramme

## Aggregation und Komposition

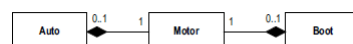
- Die Komposition wird wie die Aggregation als Linie zwischen zwei Klassen gezeichnet und mit einer kleinen Raute auf der Seite des Ganzen versehen.
- Im Gegensatz zur Aggregation wird die Raute jedoch ausgefüllt.



- Aggregation



- Komposition

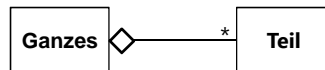


© 2018 MHP Management- und IT-Beratung GmbH

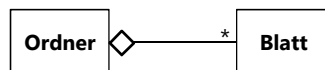
98

## 7. Klassendiagramme

## UML-Grundlagen – Aggregation



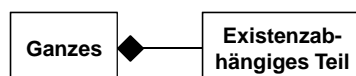
**Aggregation**  
(universell wiederverwendbare Teile):

**Erläuterung:**

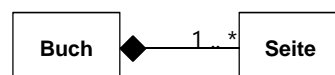
- In der Implementierung wird die Aggregation mit Zeigern bzw. Referenzen umgesetzt.
- Das „Groß“-Objekt (das Ganze) zeigt auf eine beliebige Anzahl von „Klein“-Objekten (das Teil).
- Die Aggregation ist eine Zusammensetzung eines „Groß“-Objektes aus einer Menge von „Klein“-Objekten. Das „Groß“-Objekt nimmt stellvertretend für seine „Klein“-Objekte alle Aufgaben wahr.
- Die Aggregation ist eine stärkere Ausprägung der Assoziation!

## 7. Klassendiagramme

## UML-Grundlagen – Komposition



**Komposition**  
(„feste Verbindung“ der Teile):

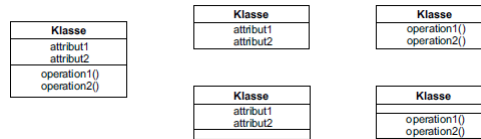
**Erläuterung:**

- Die Komposition drückt den **exklusiven Besitz** eines „Klein“-Objektes durch ein „Groß“-Objekt aus. Ein „Klein“-Objekt kann nur genau einem „Groß“-Objekt gehören.
- Die Komposition ist eine stärkere Beziehungsform als die Aggregation, bei der die Teile vom Ganzen existenzabhängig sind.
- Wird das „Groß“-Objekt zerstört, so wird auch automatisch das „Klein“-Objekt (existenzabhängiges Teil) zerstört.

## 7. Klassendiagramme

## Klasse (Class)

- Eine Klasse (**Class**) beschreibt eine Menge von Instanzen, die dieselben Merkmale, Zusicherungen und Semantik haben.
- Klassen werden durch Rechtecke dargestellt, die entweder nur den Namen der Klasse tragen oder zusätzlich auch Attribute und Operationen.
- Verschiedene Notationsvarianten für Attribute und Operationen:



## 7. Klassendiagramme

## Beispielsystem Bibliothek: Eigenschaften finden



Welche Eigenschaften dieser Objekte sind im Rahmen der zu erledigenden Aufgaben von Bedeutung?



Finden der relevanten Datenfelder (Eigenschaften, Attribute) der Klassen!

„Buch-Objekte“



Klasse Buch

- Buchnummer
- ISBN
- Titel
- Autor
- Auflage

„Ausleiher-Objekte“



Klasse Ausleiher

- Ausleihernummer
- Name
- Vorname
- Anschrift

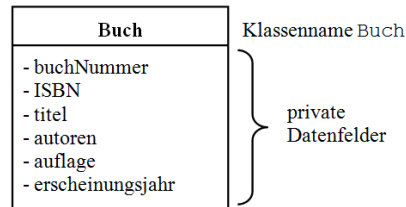
## 7. Klassendiagramme

## Beispielsystem Bibliothek: UML-Notation

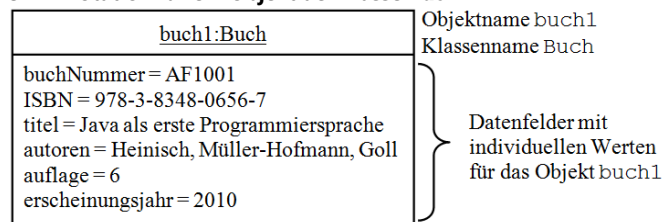
„Buch-Objekte“



## UML-Notation der Klasse Buch



## UML-Notation für ein Objekt der Klasse Buch



## 7. Klassendiagramme

## Beispielsystem Bibliothek: Methoden finden



Was kann man mit den Buch-Objekten in der Bibliothek machen?  
 Was machen die Ausleiher-Objekte in der Bibliothek?



Finden der relevanten Methoden für die Klassen!



Beobachtbare Vorgänge in der Bibliothek:

- Ein Buch wird von einem Ausleiher ausgeliehen.
- Ein Buch wird von einem Ausleiher zurückgegeben.
- Ein Buch wird in die Bibliothek aufgenommen.
- Ein Buch wird durch den Bibliothekar in ein Regal gestellt.
- Ein Buch wird durch den Ausleiher aus einem Regal geholt.
- ...



Welche der beobachtbaren Vorgänge bzw. welche Anteile sollen automatisiert werden?

## 7. Klassendiagramme

## Beispielsystem Bibliothek: Identifikation Anwendungsfälle

- Durch Beobachten der Abläufe werden die **Geschäftsprozesse** identifiziert.
- Für die spätere Programmierung relevant sind die so genannten **Anwendungsfälle**.



Die Anwendungsfälle werden gefunden, indem man die Geschäftsprozesse genau analysiert und festlegt, welche Teile durch das System automatisiert werden sollen.

- Der Geschäftsprozess „Buch ausleihen“, lässt sich wie folgt zerlegen:
    - Buchrecherche durchführen, ←
    - Buchverfügbarkeit prüfen, ←
    - Buch aus Regal holen, →
    - Ausleiher identifizieren, ←
    - Buch für Ausleiher als entliehen buchen. →
- Software-Unterstützung naheliegend  
→ Kandidaten für Anwendungsfälle
- Für die Anwendungsfälle ist zu prüfen, welche Objekte mit welchen Methoden hier unterstützen können.

## 7. Klassendiagramme

## Beispielsystem Bibliothek: Methoden für die Klasse Buch



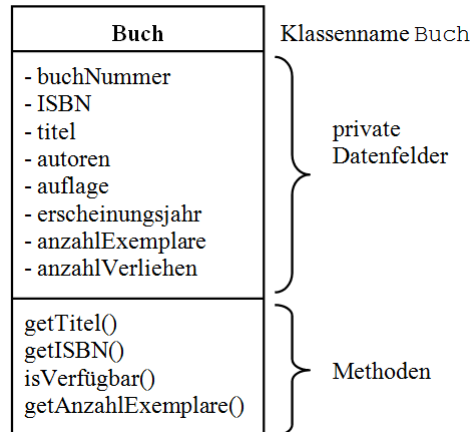
Wie kann ein Buch-Objekt die Anwendungsfälle unterstützen?

- Anwendungsfall „Buch-Recherche durchführen“
  - Ein einzelnes Buch-Objekt kann keine „Recherche durchführen“.
  - Ein einzelnes Buch-Objekt kann aber seine Daten für die Recherche bereitstellen:
    - `getTitel()`
    - `getISBN()`
- Anwendungsfall „Buchverfügbarkeit prüfen“
  - Ein Buch-Objekt muss wissen, wie viele Exemplare in der Bibliothek sind.  
→ neues Datenfeld `anzahlExemplare`
  - Ein Buch-Objekt ermöglicht die Abfrage der vorhandenen Exemplare über eine Methode `getAnzahlExemplare()`
  - ...

## 7. Klassendiagramme

## Beispielsystem Bibliothek: UML-Notation der Klasse mit Methoden

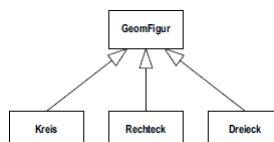
UML-Notation der Klasse Buch nach einigen Überlegungen:



## 7. Klassendiagramme

## Generalisierung (Generalization)

- Die Generalisierung (**Generalization**) ist ein Abstraktionsprinzip zur hierarchischen Strukturierung der Semantik eines Modells.
- Eine Generalisierung ist eine Beziehung zwischen einem allgemeinen und einem speziellen Classifier, wobei der speziellere weitere Merkmale hinzufügt und sich kompatibel zum allgemeinen verhält.
- Die Generalisierung wird durch einen speziellen Pfeil notiert, der vom speziellen zum allgemeinen Element zeigt
- Notation:



## 7. Klassendiagramme

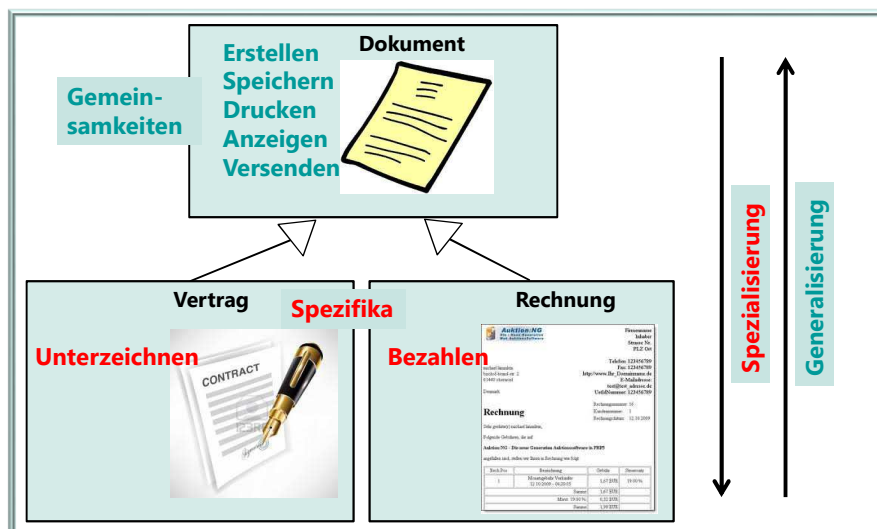
## Vererbung – Ausgangslage: mehrere Klassen mit Gemeinsamkeiten



Problem: Bei separater Implementierung der Gemeinsamkeiten in unterschiedlichen Klassen kommt es zur Code-Duplizierung --> Änderungsaufwand an mehreren Stellen --> erschwerte Wartbarkeit.

## 7. Klassendiagramme

## Vererbung – Lösung: Vererbung mit Variantenbildung



## 7. Klassendiagramme

### Übung: Finden von Klassen und geeigneten Abstraktionen

Betrachten Sie die folgenden Entitäten der realen Welt. Finden Sie geeignete Klassen zur Abbildung der Entitäten in einem Klassendiagramm und identifizieren Sie für die gefundenen Klassen weitere Abstraktionen im Sinne einer Generalisierung. Hinweis: Zum Finden der Abstraktionen können Sie auch im Internet recherchieren.



Vorbereitungszeit: 10 Minuten  
Ergebnisdiskussion: 10 Minuten

## 7. Klassendiagramme

### Übung

#### Kurzbeschreibung

- Zu einem Projekt existiert immer mindestens eine Version, welche wiederum Bezug zu mindestens einer oder mehreren Projektalternativen hat. Des Weiteren hat ein Projekt genau einen Projektkopf, welcher mit mindestens einem Projektattribut verknüpft ist. Die Attribute können in einem Projekt mit konkreten Werten belegt sein.
- WiBe unterscheidet verschiedene Kriterien (Monetäre Kriterien, Dringlichkeitskriterien, Qualitätskriterien, Externes Kriterium), wobei ein Monetäres Kriterium mit einem oder mehreren Richtwerttabellen verknüpft ist.
- Verschiedene Kriterien werden in einem Kriterienkatalog miteinander verbunden. Solch ein Kriterienkatalog ist mit einem Projekt über das Projektattribut verbunden.
- Ebenfalls zu beachten ist das Element Notiz, welches die in vielen Teilen der Anwendung verfügbare Notizfunktion abbilden soll.

**>> Bitte erstellt für dieses Datenmodell ein Klassendiagramm**



## 7. Klassendiagramme

### Zusammenfassung:

#### Checkliste Klassendiagramm:



- Wie sind gerichtete Beziehungen definiert?
- Können gerichtete Beziehungen mehr als ein Quell- bzw. Zielement haben?
  
- Kann für ein Paketimport ein Alias definiert werden?
- Was ist der Unterschied zwischen privatem und öffentlichem Import?
- Welche Art von Elementen können mit dem Elementimport importiert werden?
  
- Nennen Sie ein Beispiel für einen Typ.
- Nennen Sie ein Beispiel für ein typisierbares Element.
  
- Welcher Wertebereich wird von einer Multiplizität beschrieben?
- Was ist der Unterschied zwischen Multiplizität und Kardinalität?

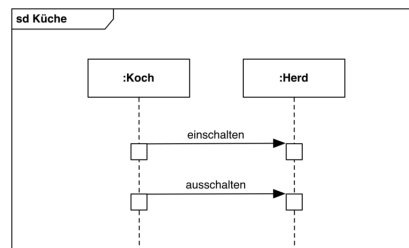
## Agenda

- 1 Methoden der SW Technik
- 2 Allgemeine Grundlagen UML
- 3 Anwendungsfall-/ Use Case Diagramme
- 4 Aktivitätsdiagramme
- 5 Objekt- und Paketdiagramme
- 6 Komponentendiagramme
- 7 Klassendiagramme
- 8 Sequenzdiagramme
- 9 Objektorientierte Methode

## 8. Sequenzdiagramme

## Definition

- Sequenzdiagramme beschreiben die Kommunikation zwischen Objekten in einer bestimmten Szene.
- Es wird beschrieben welche Objekte an der Szene beteiligt sind, welche Informationen (Nachrichten) sie austauschen und in welcher zeitlichen Reihenfolge der Informationsaustausch stattfindet.
- Sequenzdiagramme enthalten eine implizite Zeitachse. Die Zeit schreitet in einem Diagramm von oben nach unten fort. Die Reihenfolge der Pfeile in einem Sequenzdiagramm gibt die zeitliche Reihenfolge der Nachrichten an.



© 2018 MHP Management- und IT-Beratung GmbH

116

## 8. Sequenzdiagramme

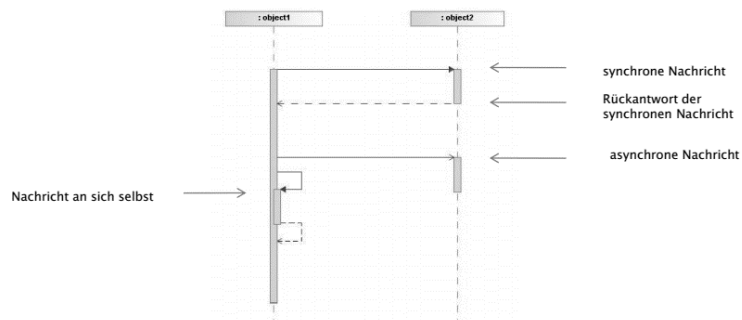
## Notation und Semantik

**Synchrone Nachricht**

- Sender wartet, bis Empfänger die Nachricht abgearbeitet hat
- Gestrichelter Pfeil für Rücksprung zum Sender

**Asynchrone Nachricht**

- Sender wartet nicht auf Empfänger und arbeitet unmittelbar weiter
- Sender und Empfänger befinden sich in unterschiedlichen Ausführungsprozessen
- Kein gestrichelter Rückgabepfeil !

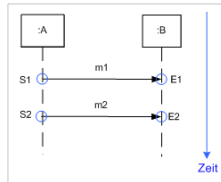


© 2018 MHP Management- und IT-Beratung GmbH

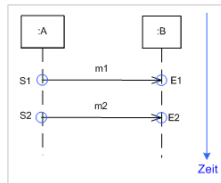
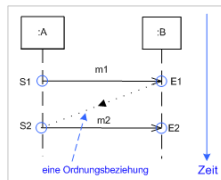
117

## 8. Sequenzdiagramme

## Notation und Semantik – Zeitliche Ordnung

**Synchrone Aufrufe**

Reihenfolge: &lt;S1, E1, S2, E2&gt;

**Asynchrone Aufrufe**Reihenfolge: <S1, E1, S2, E2> oder  
<S1, S2, E1, E2> oder <S1, S2, E2, E1>**Asynchrone Aufrufe + Ordnungsbeziehung**

Reihenfolge: &lt;S1, E1, S2, E2&gt;

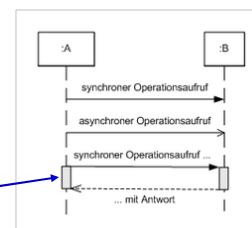
img GmbH

118

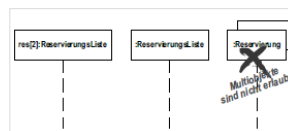
## 8. Sequenzdiagramme

## Notation und Semantik

- In der Abbildung: 2 „Lebenslinien“
- Wenn zwischen Lebenslinien Nachrichten ausgetauscht werden, muss auch ein Verhalten in den zugehörigen Elementen ausgeführt werden. Das wird durch die länglichen Rechtecke auf der Lebenslinie dargestellt. Die Rechtecke repräsentieren den so genannten Ausführungsfokus.



- Multiobjekte sind nicht erlaubt

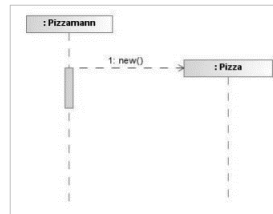


- Im Kopf einer Lebenslinie kann auch das Schlüsselwort *self* stehen. Dem Lebenslinie repräsentiert dann ein Exemplar des Classifiers, zu dem die Interaktion gehört.

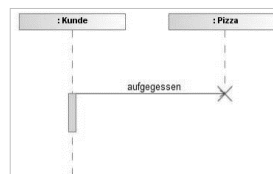
## 8. Sequenzdiagramme

## Notation und Semantik - Nachrichten

- Erzeugen von neuen Objekten



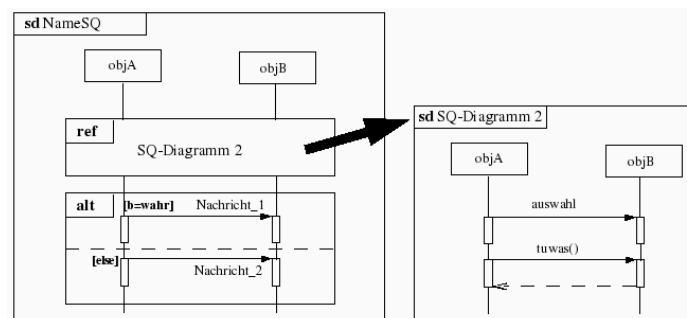
- Zerstören/Löschen von Objekten



## 8. Sequenzdiagramme

## Notation und Semantik – Verweis &amp; Alternative darstellen

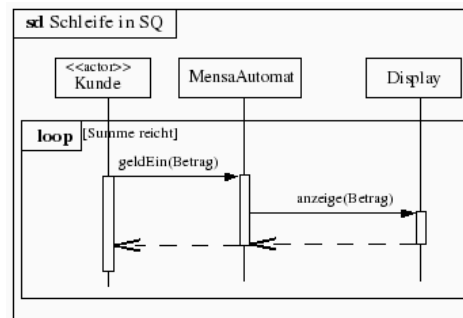
- If/Else-Darstellung sowie Referenz abbilden:



## 8. Sequenzdiagramme

## Notation und Semantik – Schleife darstellen

- Loop/Schleife abbilden:



## 8. Sequenzdiagramme

## Übung

**Kurzbeschreibung**

Case: Benutzer anlegen

1. Der WiBe-Beauftragte wählt die Operation „Benutzer, Neu ...“ aus einem durch die Benutzerschnittstelle zur Verfügung gestellten Menü.
2. Das System präsentiert einen Dialog, in dem die Benutzerdaten (z. B. Name, Email-Adresse, Rolle, Passwort, Passwortwiederholung, etc.) eingegeben werden können.
3. Das System prüft die korrekte Eingabe aller Daten. Dabei wird auf Vorhandensein und Korrektheit bzgl. des Formats der Eingabe und der Verträglichkeit mit den Zieldatentypen getestet.
4. Die erfassten Benutzerdaten werden durch das System dem WiBe-Beauftragten in einer Übersicht zur Kontrolle dargestellt. Findet er einen Fehler, springen wir zu Punkt 2 zurück.
5. Ansonsten legt das System daraufhin entsprechende Datenobjekte und Einträge in der Datenbank ab.

**>> Bitte erstellt für diesen Ablauf ein Sequenzdiagramm**

## 8. Sequenzdiagramme

## Zusammenfassung:

## Checkliste Sequenzdiagramme:



- Wie sind gerichtete Beziehungen definiert?
- Können gerichtete Beziehungen mehr als ein Quell- bzw. Zielelement haben?
  
- Kann für ein Paketimport ein Alias definiert werden?
- Was ist der Unterschied zwischen privatem und öffentlichem Import?
- Welche Art von Elementen können mit dem Elementimport importiert werden?
  
- Nennen Sie ein Beispiel für einen Typ.
- Nennen Sie ein Beispiel für ein typisierbares Element.

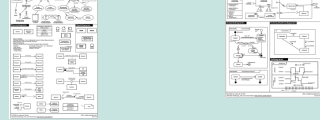
## Agenda

- 1 Methoden der SW Technik
- 2 Allgemeine Grundlagen UML
- 3 Anwendungsfall-/ Use Case Diagramme
- 4 Aktivitätsdiagramme
- 5 Objekt- und Paketdiagramme
- 6 Komponentendiagramme
- 7 Klassendiagramme
- 8 Sequenzdiagramme
- 9 Objektorientierte Methode

## 9. Objektorientierte Methode

## Notation, Prozess und Methode

## Notation - UML



## Prozess (Vorgehensmodell):

- zu erfüllende Aktivitäten,
- deren zeitliche Abhängigkeiten,
- deren Resultate (Dokumente, Code)



Fokus: "Was und Wann"

## Objektorientierte Methode

Fokus: "Wie"

bestimmt zusätzlich für die Aktivitäten Analyse und Entwurf eines Prozesses:

- die einzelnen Schritte, um die Resultate zu erhalten
- die Verwendung der Sprachelemente aus der Notation für jeden Schritt.

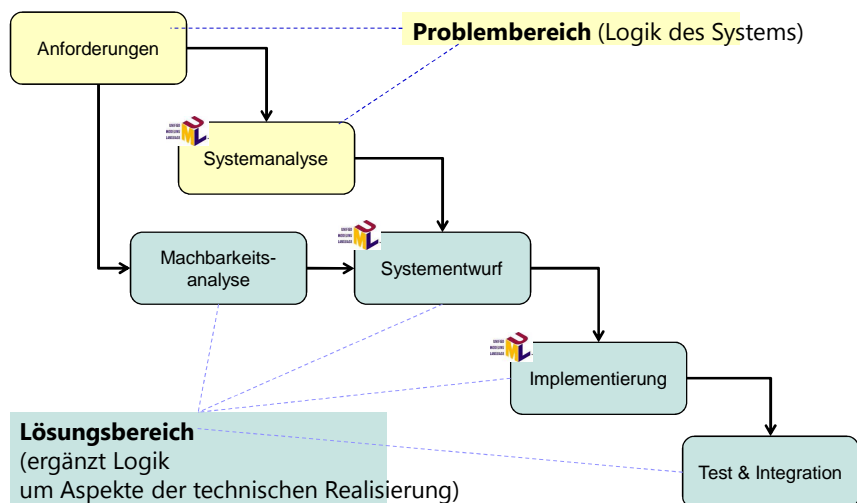


Je nach

- Projekt (klein, groß, Informationssystem, Realzeitsystem, ...),
  - Anwendungsdomäne (Automotiv, Banken, ...),
  - Vorkenntnissen der Projektmitarbeiter
- entscheidet der Systemanalytiker/-architekt, welche **Methode** zum Einsatz kommt.

## 9. Objektorientierte Methode

## Problembereich und Lösungsbereich



## 9. Objektorientierte Methode

## Systemanalyse und Systementwurf

**Systemanalyse**

- befasst sich mit dem Problembereich
- d.h. der fachlichen Logik der Anwendung
- analysiert werden Geschäftsprozesse
- Anwendungsfälle werden identifiziert und ausgearbeitet.

**Systementwurf**








- befasst sich mit dem **Lösungsbereich**
- d.h. mit der **technischen Realisierung**
- die Logik aus der Systemanalyse wird in ein **lauffähiges System** gegossen



Werden in der Systemanalyse nicht sorgfältig die Geschäftsprozesse analysiert und die **richtigen Anwendungsfälle** identifiziert, läuft man Gefahr, ein System zu bauen, das der Anwender nicht haben wollte, oder nicht gebrauchen kann.

## 9. Objektorientierte Methode

## Schritte in der Objektorientierten Systemanalyse







1. Überprüfen der Anforderungen
2. Spezifizieren der Geschäftsprozesse
3. Priorisieren der Anforderungen
4. Erstellen des Kontextdiagramms (Systemgrenzen festlegen) 
5. Anforderungen neu definieren (Pflichtenheft)
6. Erstellen des Anwendungsfalldiagramms 
7. Kurzbeschreibung der Anwendungsfälle 
8. Finden von Klassen und Erstellen des Klassendiagramms der konzeptionellen Sicht
9. Langbeschreibung der Anwendungsfälle
10. Erstellen der Kommunikationsdiagramme für jeden Anwendungsfall 
11. Erstellen des Klassendiagramms der Verarbeitungssicht 
12. Festlegen der Abhängigkeitsbeziehungen 
13. Erstellen des Klassendiagramms der finalen Sicht der Systemanalyse 



## 9. Objektorientierte Methode



### Schritte im Objektorientierten Systementwurf

1. Ableiten des Schichtenmodells (inkl. Entwurfsspezifische Schichten) 
2. Vervollständigen des dynamischen Verhaltens aus der Systemanalyse 
3. Entwurfsspezifische Kollaborationen ergänzen 
4. Kollaborationen optimieren und Entwurfsmuster identifizieren 
5. Abhängigkeitsbeziehungen ermitteln 
6. Klassendiagramm des Entwurfs zeichnen 

Alle Schritte aus Systemanalyse und Systementwurf werden anhand der Fallstudie „Bücherverwaltung“ in den nächsten Vorlesungen durchlaufen.



Herzlichen Dank  
für Ihre Aufmerksamkeit!

Hugo Colceag

**Mieschke Hofmann und Partner**  
Gesellschaft für Management- und IT-Beratung mbH

Film- und Medienzentrum | Königsallee 49 | D-71638 Ludwigsburg  
Telefon +49 (0)7141 7856-0 | Fax +49 (0)7141 7856-199  
eMail [info@mhp.com](mailto:info@mhp.com) | Internet [www.mhp.com](http://www.mhp.com)