

## Requirements & Modelling

# Inhalt

## **1. USDP Der Software - Entwicklungsprozess**

Wie ein SW Projekt ablaufen sollte oder könnte

## **2. UML Anwendungsfall - Modell**

Akteure und ihre Aufgaben

## **3. UML Verhaltens - Modell 1**

Aktivitäts- und Zustandsbeschreibung

## **4. UML Klassen - Modell**

Strukturbeschreibung des Systems

## **5. UML Verhaltens - Modell 2**

Sequenzdiagramme

## **6. UML Design und Implementation**

Ausblick

## **7. Testen von eBusiness Applikationen**

Allgemeine Prinzipien, Spezialitäten für eBusiness

# Unterlagen

- "The Unified Software Development Process"; Ivar Jacobson, Grady Booch James Rumbaugh; Addison-Wesley; 1999 .
- "Objektorientierte Softwareentwicklung Analyse und Design mit der UML"; Bernd Oestereich; Oldenbourg; 2001.
- "Die UML-Kurzreferenz für die Praxis"; Bernd Oestereich; Oldenbourg; 2001.
- "Building Web Applications with UML"; Jim Conallen; Addison-Wesley; 1999.
- <http://www.oose.de/uml>, [http:// www.oose.de/oep](http://www.oose.de/oep)
- <http://www.omg.org/uml>

# Kursziel

- Einführung und Motivation für UML (Unified Modelling Language), einer gemeinsamen Sprache für Fachgebiets-Spezialisten und Software-Entwickler, damit Sie
  - beim Erstellen von Requirement & Analyse mitarbeiten können
  - von Entwicklern erstellte Dokumente interpretieren können
- Dauer dieses Kurses: 1.5 bis 2 Tage.

# Software-Entwicklung

Ablauf des ganzen Projektes:

**Unified Software Development Process USDP**

Beschreibung der einzelnen Teile:

**Unified Modelling Language UML**

# USDP

USDP hat drei Hauptaxiome:

1. Iteratives und inkrementelles Vorgehen
  - Projekte in Teilprojekte (Phasen) zerlegen.
  - Jede Phase beinhaltet alle Schritte der Software ~~Ent~~wicklung
  - Aus jeder Phase lernen und damit Risiken minimieren.
  - Kein Wasserfall ~~V~~orgehen.
2. Anwendungsfallorientiert
  - Software auf Basis von geschäftlichen Anwendungsfällen entwickeln.
3. Architektur-orientiert
  - System in logische Teile gruppieren, sei es bezüglich den Anwendungsfällen, sei es bezüglich Software - Komponenten.

Übersicht zum Entwicklungsprozess  
von Software mit Hilfe des Unified  
Software Development Process  
USDP.

USDP ermöglicht einen nahtlosen  
Übergang zur Sprache für die  
Softwaremodellierung UML.

USDP wird von verschiedenen Firmen  
mit Werkzeugen unterstützt, z.B.  
Rational (RUP) oder methodisch  
ausgearbeitet (OEP).

# USDP Phasen

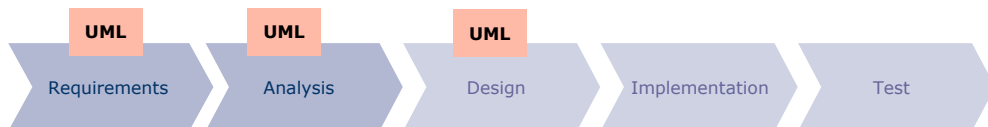
Es werden 4 Entwicklungsphasen unterschieden:



Jede Phase ist ein vollständiges Software-Projekt, allerdings mit unterschiedlichen Zielen.

Jede Phase kann ev. mehrmals durchlaufen (iteriert) werden.

*Jede Phase* beinhaltet wiederum folgende (klassischen) Abschnitte:



# R A D I T



- **R**equirements (im eBusiness): Was verlangt der Benutzer vom System? Welche Anwendungsfälle resp. Abläufe soll das System unterstützen oder ausführen? Arbeitsbeschreibung. Nicht in Software-Methodik denken.
- **A**nalyse: Abstraktion und innere Strukturierung. Methodik des Software-Engineerings anwenden (Objektorientierung). Datenmodell und Funktionsprimitiven erarbeiten, Geschäftsregeln formal definieren. Noch *mit Begriffen des Anwenders* arbeiten.
- **D**esign/**I**mplementation: *Programmier-Werkzeuge und -Techniken festlegen*. Beispielsweise: Soll mit Java oder C# gearbeitet werden? Soll XML als Datentransfer-Protokoll verwendet werden? *Begriffe des Software-Engineerings* stehen im Vordergrund. Entschluss für eine Systemarchitektur, zum Beispiel 4-Tier-Architektur.
- **T**esten: Modultest (I), Integrationstest (D), Systemtest(A), Anforderungstest(R).

# Requirements



- **Functional Requirements**  
Definition aller Aufgaben, welche das System aus Sicht des *Anwenders* lösen soll. Jede Aufgabe ist als Anwendungsfall definiert, Anwendungsfälle können (sollten) in Geschäftsbereich gruppiert werden. Arbeitsbeschreibung! Nicht in Software-Methodik denken.
- **Non-Functional Requirements**  
Rahmenbedingungen an das System, beispielsweise Angaben zur Performance, Verwendung von bestimmten Software-Werkzeugen, Schnittstellen zu Legacy-Systems, Verwendung von Standards und Richtlinien, Sicherheitsanforderungen, Performance, Datenmengen, Transaktionssicherheit usw.

Anwendungsfall 'Offerte erstellen' im Paket Sachversicherung:

1. Kundendaten aufnehmen
2. Mindestleistung definieren
3. Maximale Prämie definieren
4. Vorschläge berechnen
5. Maximal drei Vorschläge auswählen
6. Vorschläge an Kunde weiterleiten
7. Offerte als pendent eintragen, Erinnerungsdatum bestimmen

Anwendungsfälle sind das Gerüst für die Requirement-Spezifikation.

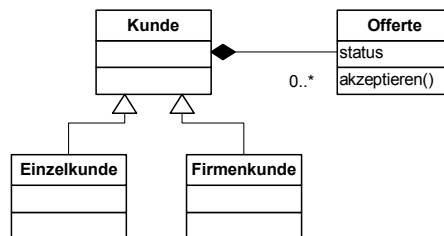
Selbstverständlich können und müssen meistens noch weitere Dokumente, Datenbeschreibungen, Geschäftsregeln vorhanden sein, damit eine Software-Applikation entstehen kann.

Festzuhalten ist bereits hier, dass Anwendungsfälle einen normalen "Schönwetter"-Ablauf beschreiben. Exzessive Verzweigungen, Fehlerbehandlung- und Ausnahmebeschreibungen sind zu unbedingt zu vermeiden.

# Analyse



- Ziel ist es, Datenmodell und Funktionsprimitiven zu erarbeiten und Geschäftsregeln formal zu definieren. Noch wird *mit Begriffen des Anwenders* gearbeitet.
- Für das Offertbeispiel kann in der Analyse beispielsweise entschieden werden, folgendes, globales Datenmodell zu erstellen:



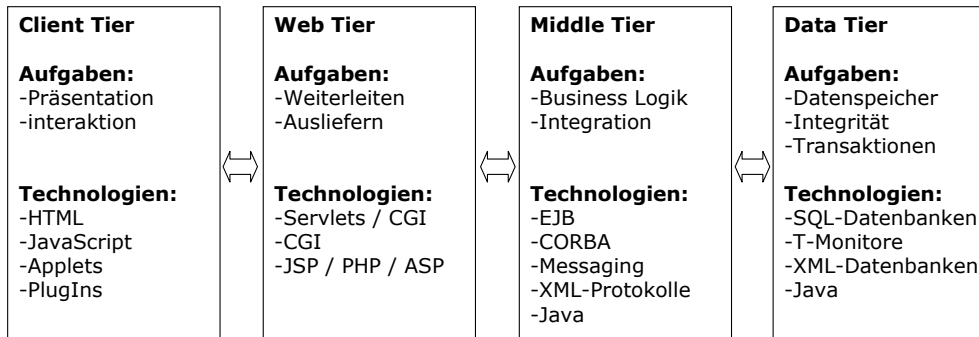
Obiges Beispiel ist ein so genanntes UML-Klassendiagramm. Es zeigt die Annahmen der Software-Entwickler über das System. Die Notwendigkeit von Annahmen taucht eventuell erst in der Analyse auf, und ist dann mit den Fach-Experten zu diskutieren und zu verifizieren. In diesem grafischen Beispiel kommen folgende Annahmen zum Tragen:

Eine Offerte gehört immer zu einem und nur zu einem Kunden. Es gibt keine Standardofferten. Eine Kunde ist entweder ein Einzelkunde oder ein Firmenkunde. Die Art des Kunden ändert sich zu seiner Lebenszeit (in der Software) nie. Einzelkunden und Firmenkunden unterscheiden sich in wesentlichen Aspekten. Die Applikation kann für gewisse Funktionen, muss aber nicht, zwischen Firmen- und Einzelkunden unterscheiden. Beide Arten von Kunden sind Offertfähig. Es gibt keine strukturellen Unterschiede zwischen Offerten für Einzel- und für Firmenkunden.

# Design



- Eine Kernaufgabe des Designs ist das Festlegen der Software-Werkzeuge und einer *Systemarchitektur*. im eBusiness wird meist eine *Multitier*-Architektur eingesetzt:



Die Unterscheidung zwischen Analyse und Design ist fließend.

Technologische Zwänge können zu grundsätzlichen Änderungen in der Analyse führen. Besteht beispielsweise die Anforderung, dass sämtliche Versicherungsdaten für statistische On Line Auswertungen (Data Mining) zur Verfügung stehen, ist das System - im Rahmen der Budgetmöglichkeiten - eventuell überfordert. In diesem Fall sind in der Analyse bereits zusätzliche Datenelemente für "vorgekochte" Statistik-Informationen vorzusehen.

# Implementation



- Design und Implementation gehen Hand in Hand
- Entwicklungsplattformen erhöhen die Produktivität, aber nicht um Faktoren
  - A fool with a tool is still a fool !
  - Aufwand für Erfahrungen berücksichtigen
  - Aufwand für Schulung berücksichtigen
  - Wie wirken sich Toolfehler aus
- Implementation und Test ca. 40% des Projektes (Construction Phase).

# Test



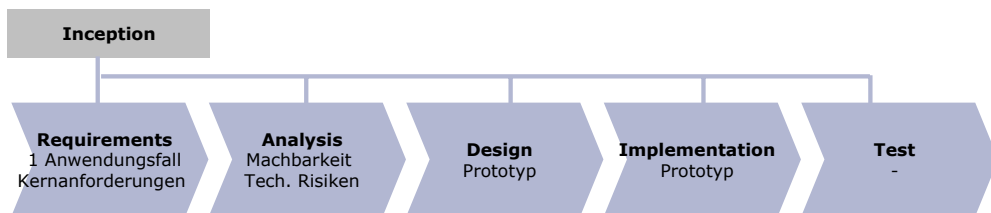
- Testen heisst nicht nur debuggen. Getestet wird auf allen Abstraktionsebenen:

|           |             |                 |                     |
|-----------|-------------|-----------------|---------------------|
| Requirmts | Abnahmetest | Kunde,PL, Entw. | Use Cases           |
| Analyse   | Systemtest  | PL + Entwickler | Review (Architekt.) |
| Design    | Integr.test | PL + Entw.      | Review, Code        |
| Impl      | Modultest   | Entwickler      | Code                |

- Es kann nicht alles getestet werden
  - Was *muss* funktionieren
  - Auswirkungen bei Funktionsfehlern
  - Zeitaufwand für die Tests
  - ⇒ Prioritäten festlegen
- Tests vorbereiten und dokumentieren

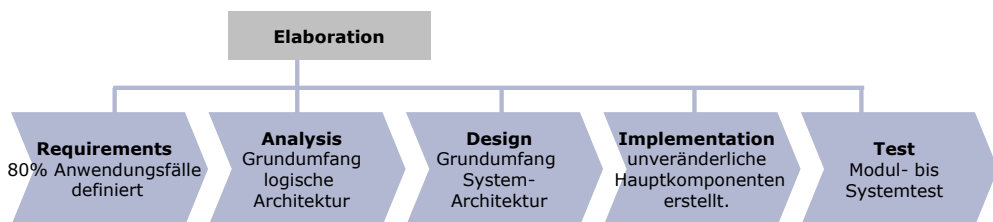
# Inception

- Ziele
  - Machbarkeit feststellen.
  - *Einen* Anwendungsfall realisieren.
  - Feststellen, ob das Projekt einen Nutzen bringt.
  - Risiken identifizieren.



# Elaboration

- Ziele
  - 80% der identifizierten Anwendungsfälle und 80% ihrer Funktionalität realisieren. Stabiles Software Gerüst bauen.
  - Rest Umfang des Projektes abschätzen bezüglich Zeit, Ausrüstung, Human Resources, Kosten.
  - Qualitätskriterien definieren (Verfügbarkeit, verkraftbare Last, Integrität).
  - Weitere Risiken abschätzen (technische Knacknüsse, Systemverhalten in Produktion?).
  - Über definitive Durchführung entscheiden.
  - Glossar erstellen.
  - Ev. Entwicklungs Prioritäten für Construction Phase festlegen.



Elaboration = Pilot

Noch keine Datengarantie

Seltene Situationen in der Abarbeitung eines Anwendungsfalles können noch weggelassen werden.

# Glossar

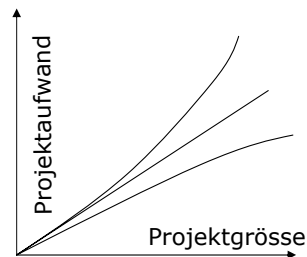
- Ein Glossar mit den verwendeten Begriffen vermeidet Irrtümer und Kommunikationsstörungen. Beispiele aus einem med. Projekt:
- Was ist ein Benutzer?
  - Entwickler: Jeder, der vor dem Bildschirm sitzt und mit dem System arbeitet.
  - Auftraggeber: Ein externer Arzt, der lesend auf anonymisierte, statistisch ausgewertete Patientendaten zugreift.
- Was ist ein Formular?
  - Entwickler: *Ein* Stück Papier mit vorgedruckten Feldern zum Ausfüllen. Wird *in einem Zug* ausgefüllt.
  - Auftraggeber: Mehrere Stücke Papier mit Sektionen, die zeitlich versetzt ausgefüllt werden. Im Prinzip ein ganzer medizinischer Fall.
- Was heisst 'submit'?
  - Entwickler: OK ~~Kopf~~ drücken.
  - Auftraggeber: (elektronisches) Unterschreiben eines Formulars, meist zeitlich versetzt zum Ausfüllen des Formulars.

# Aufwandschätzung

- Beispiel Methode CoCoMo (Construction Cost Model).
- Basiert auf der zu erwartenden Projektgrösse gemäss der Formel :

$$\text{Aufwand} = K_1 * \text{Grösse}^{K_2}$$

- $K_1$  Projektparameter
- $K_2$  Komplexitätsmass
- Metriken für die Grössenschätzung
  - Function Points: Anzahl Dialoge, Schnittstellen, Reports, Tabellen, Transaktionen.
  - GUI: Anzahl Dialoge, Menus, Reports, Windows, Tabellen.
  - Klassen: Anzahl Klassen.
  - Anwendungsfälle: Anzahl.
  - SLOC: Anzahl Zeilen Source Code.
- Aufwandsresultat in Zeit, Geld, Personen



 Demo CostXpert

# Risiko-Abschätzung

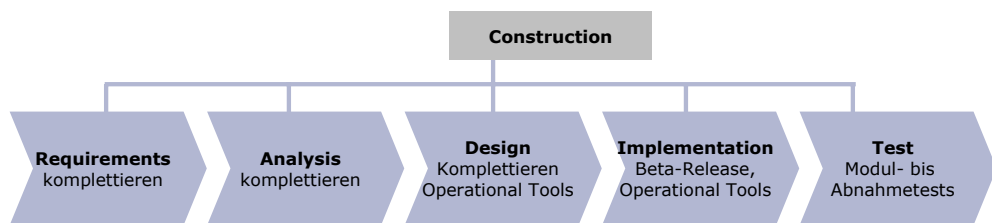
- Risikos bezügl. SW-Entwicklung:
  1. Functional Risks
    1. Ist man sich über den Zweck und die Aufgaben im Klaren?
    2. Tun die AWF's das was man von ihnen erwartet, sind sie sinnvoll?
  2. Non-functional Risks
    1. Kommt das System technisch zum Funktionieren?
    2. Wird es jemals schnell genug sein?
    3. Fehler in eingekauften Komponenten?

Beispiel zu 1.2 aus med. Projekt: Im Pilotprojekt stellt sich heraus, dass viele Patienten nicht in der Lage sind, auch nur ansatzweise mit einem Computer umzugehen. Der AWF 'Patient erstellt Eintrittsblatt' auf einem PDA oder Tablet-PC ist damit in Frage gestellt.

Beispiel zu 1.3: Java-Komponenten für PDA's sind extrem fehlerbehaftet. Projektzeit verlängert sich um 50 bis 100%.

# Construction

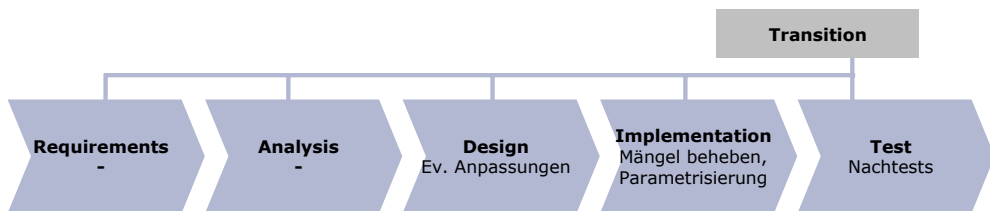
- Ziele
  - Gesamtsystem bauen, alle Anwendungsfälle komplett implementieren.
  - System Architektur nicht korrumpieren.
  - Operational Tools (Anwenderseitig) für die System- Überwachung,- Pflege.
  - Tests so, dass Datenintegrität (technisch, nicht juristisch) gewährleistet.



Viele Aspekte des Bediener-Comforts können hier noch zum Zug kommen. Beispielsweise Blätter-Mechanismen, Default-Selektionen oder feinere Auswahl-Mechanismen für grosse Datenmengen.

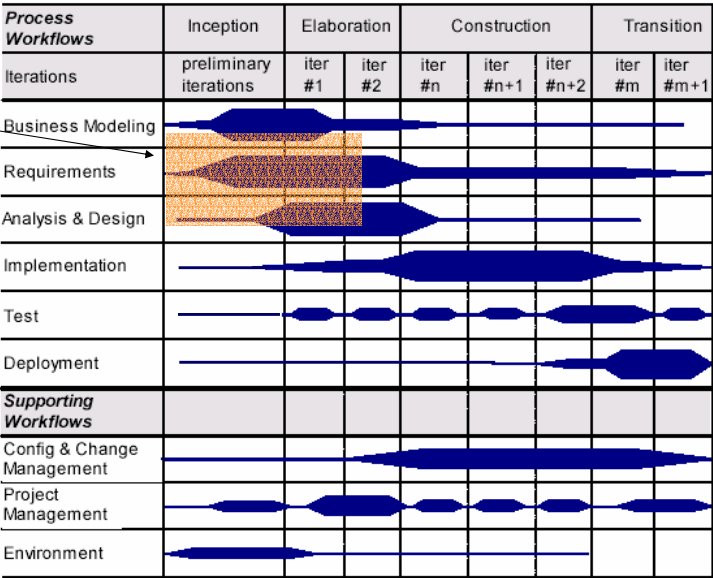
# Transition

- Ziele
  - Mängel beheben
  - Installations Tools und Parametrisierung ausfeilen
  - Benutzer-Handbücher erstellen
  - Dokumentation fertigstellen
  - Benutzer - Support einrichten



# USDP-Zusammenfassung

Gegenstand  
dieses Kurses



# UML

UML = Unified Modelling Language. UML wird gepflegt von einer offenen Gruppe. Mitglieder sind viele namhafte Software-Firmen.

Die Publikations-Plattform des Gremiums ist [www.omg.org](http://www.omg.org)

# Modellbildung

- Die Entwicklung von Software setzt ein *Modell* voraus.
- Ein Modell ist die Abstraktion eines realen Systems im Hinblick auf bestimmte *Ziele*.
- Das Modell beschreibt die *Struktur* und das *Verhalten* des Systems in einem angemessenen Detaillierungsgrad.
- An der Entwicklung des Modells sind Fachleute aus unterschiedlichen Gebieten beteiligt, welche miteinander *kommunizieren* müssen.
- Kommunikation setzt eine *gemeinsame Sprache* voraus.
- Die Unified Modelling Language UML *ist* eine gemeinsame Sprache.
- UML eignet sich für die Fachleute aus dem Anwendungsbereich *und* für die Software-Entwickler.

# Modellbildung mit UML

- UML ist stark durch grafische Darstellungen abgestützt, Kernaussagen können übersichtlich und prägnant dargestellt werden.
- UML beinhaltet auch textuelle und formale Beschreibungen. In vielen Fällen wird dafür eine bestimmte Gliederung empfohlen.
- UML-Beschreibungen werden iterativ von grob nach fein "stufenlos" präzisiert.
- UML ermöglicht einen bruchfreien Übergang von der Requirement-Spezifikation in die Software-Entwicklung.
- In der obersten Detaillierungsstufe wird nur mit Anwenderbegriffen (*Fachbegriffen*) gearbeitet.
- In der untersten Detaillierungsstufe wird nur noch die Software-entwicklung angesprochen und die Code-Generierung unterstützt.

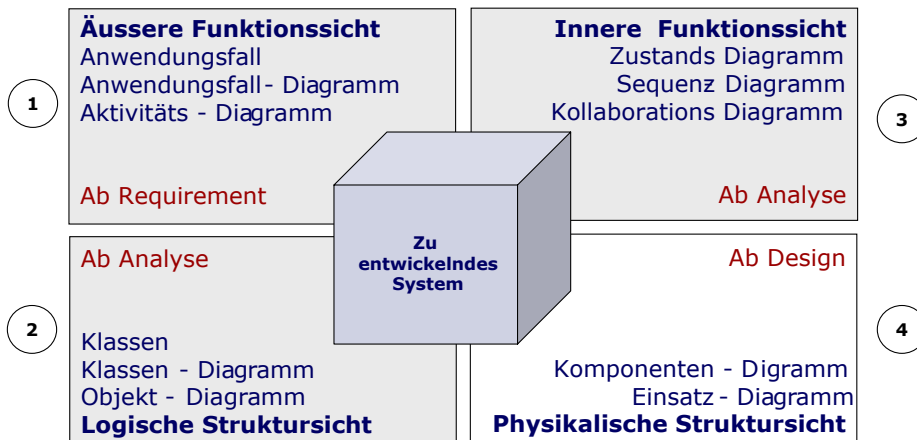
# Modellbildung mit UML ff

- UML ist an sich keine Beschreibung des Entwicklungs-*prozesses*, sondern eine Sprache für die Beschreibung von fachlichen Abläufen, Strukturen und Inhalten (im Hinblick auf die Software-Entwicklung).
- Eine Beschreibung des Entwicklungs*prozesses* ist zum Beispiel der RUP (Rational Unified Process) oder etwas verallgemeinert und nicht firmengebunden der USDP (Unified Software Development Process).
- Leute und Firmen von UML, RUP, USDP sind jedoch stark vernetzt, dass die Grenzen verschwinden.

- |                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>• UML ist eine mächtige und sehr allgemeine Sprache:<br/>Soviele Elemente wie nötig, sowenig wie möglich verwenden !</li></ul> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|

# UML-Sichtweisen

- UML arbeitet mit verschiedenen Sichtweisen auf ein System. Jede Sichtweise hebt andere Aspekte hervor.



Beispiel Mensch:

Äussere Funktionssicht: Spricht einen Satz.

Innere Funktionssicht: bewegt Zunge und Mund beim Sprechen.

Logische Struktursicht: Nase zwischen zwei Augen, darunter Mund.

Physikalische Struktursicht: Nase besteht aus Knorpel, Bindegewebe und Haut.

# Äussere Funktionssicht

Nimmt die Anforderungen an das System aus Anwendersicht, in Form typischer Abläufe, auf.  
Im Zentrum stehen Anwendungsfälle und Aktivitätsdiagramme.

# Anwendungsfall

- Ein Anwendungsfall ist eine zeitlich ununterbrochener Folge von Arbeitsschritten (Aktivitäten), die ein Ergebnis von geschäftlichem Wert produzieren.
- Ein Anwendungsfall wird von einem Akteur ausgelöst.
- Die Arbeitsschritte werden von den Akteuren oder einem "System" durchgeführt.
- Ein Anwendungsfall ist eine testbare Einheit.
- Beispiel für Anwendungsfälle: Kunde aufnehmen, Offerte erstellen, Vertrag abschliessen.

## Anwendungsfall, Beispiel

- Name des AWF: 'Buch ausleihen'
- Akteure: Benützer, Bibliotheksmitarbeiter
- Ergebnis: Ausgeliehenes Buch
- Ablauf: Benützer liest im Katalog ein Buch aus, ein freies Exemplar wird in der Bibliothek gesucht, die Entleihung wird auf der Entleihkarte eingetragen und dem Benützer wird das Buch ausgehändigt.
- *Beachte:*
  - *Der AWF wird zeitlich ununterbrochen durchgeführt.*
  - *Am Schluss des AWF steht ein Ergebnis von geschäftlichem Nutzen.*

# Anwendungsfall-Arten

- Essentieller Anwendungsfall
  - Existenzbegründend für den Geschäftsprozess.
  - abstrahiert, vereinfacht, verallgemeinert
  - fachbezogen, technologieneutral
  - Beispiel: 'Buch ausleihen'
- Gewöhnlicher Anwendungsfall
  - Notwendig, aber nicht existenzbegründend.
- Systemanwendungsfall
  - Nur für die Software, nicht für das eigentliche Geschäft relevanter Anwendungsfall. Beispiel: 'In das System einloggen'.
  - Technologische Ausprägung eines essentiellen oder gewöhnlichen AWFs. Beispiel: 'Buch über Internet ausleihen', 'Buch per Telefon ausleihen' usw.

Es ist natürlich nicht notwendig in jedem Projekt alle Arten von Anwendungsfällen zu unterscheiden. Es ist durchaus möglich, lediglich von Anwendungsfällen zu sprechen und diese nicht weiter zu klassifizieren. Bei umfangreichen Projekten lohnt sich aber mindestens die Unterscheidung zwischen essentiellen und gewöhnlichen Anwendungsfällen. Im Rahmen des Bibliothekssystems ist ein essentieller Anwendungsfall beispielsweise 'Buch ausleihen'. Dieser repräsentiert den Hauptzweck eines Bibliothekssystems. Daneben ist sicherlich 'Benutzerkarte ausfüllen' auch ein Anwendungsfall, aber nicht ein existenzbegründender. Die Klassifizierung der essentiellen Anwendungsfälle ist insofern nützlich, als diese im Rahmen einer Dokumentation oder Produktinformation als erstes aufgeführt werden.

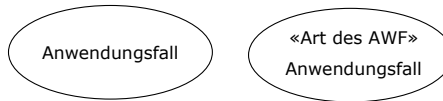
Trennung essentiell/gewöhnlich nach Häufigkeit der Anwendung?

## Anwendungsfall-Arten ff

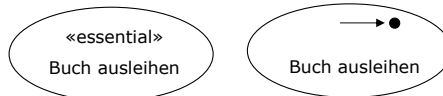
- Sekundärer Anwendungsfall
  - entstehen im Rahmen einer Zerlegung eines grösseren AWF im Rahmen der Software-Entwicklung.
  - keinen direkten Akteur, aber häufige und *erwähnenswerte* Aufgabe.
  - Beispiel: Session-Timeout prüfen.
- Externer Anwendungsfall
  - nicht zu implementieren, aber wichtig für das Verständnis des Systems.
  - Beispiel: 'Bücherbestellung per Post an Entleiher schicken'.
- Die Gesamtzahl aller Anwendungsfälle kann sich ohne weiteres im Rahmen von 100-200 bewegen, betrachtet man beispielsweise jede Mutation (Ändern, Löschen, Einfügen, Sichten) eines Business-Objektes als Anwendungsfall.

# Anwendungsfall, Darstellung

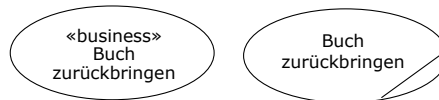
- Allgemeine Darstellung



- Essentieller Anwendungsfall



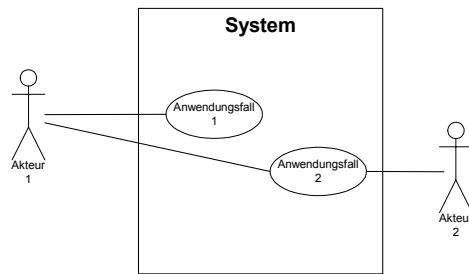
- Explizite Kennzeichnung der geschäftsorientierten Anwendungsfällen (gegenüber den Systemanwendungsfällen)



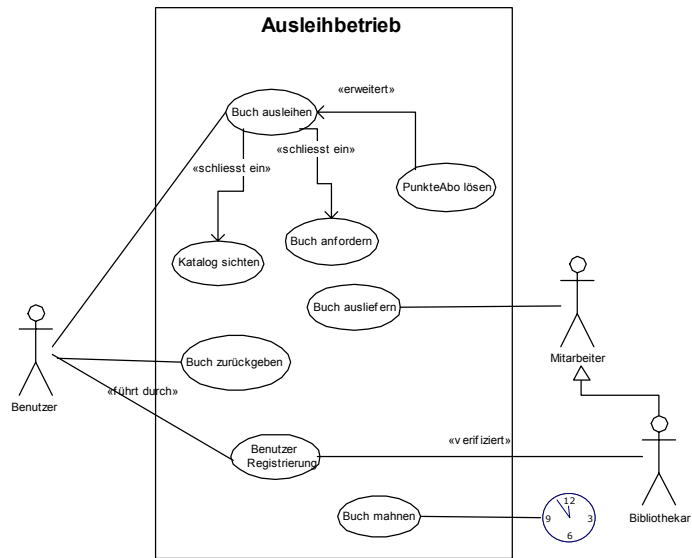
Die spezielle Charakterisierung von UML-Elementen findet häufig mit der «Stereotyp»-Notation oder mit zusätzlichen Grafik-Symbolen statt. Die «Stereotyp»-Notation hat den Vorteil, dass die Bedeutung unmittelbar ersichtlich ist.

# Anwendungsfall-Diagramm

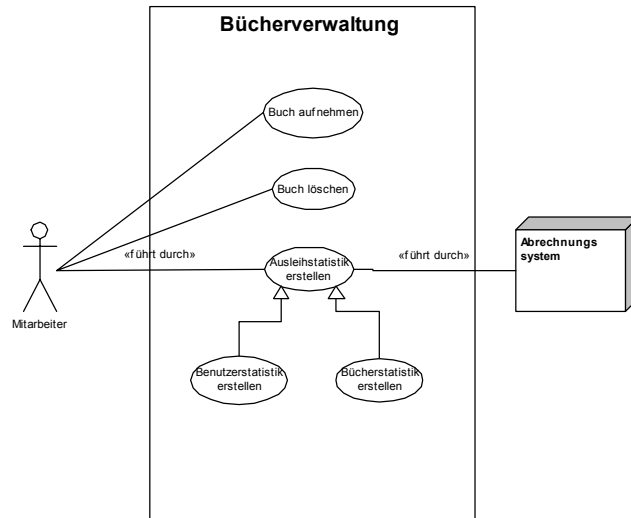
- Ein Anwendungsfall-Diagramm zeigt die Anwendungsfälle eines Systems oder Teilsystems (Geschäftsprozess) mit ihren Akteuren auf.



# Beispiel Bibliothekssystem



# Beispiel Bibliothekssystem ff



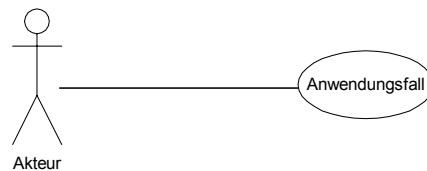
# Akteur

- Ein Akteur ist ein Mensch oder ein System, das einen Anwendungsfall auslöst oder an seiner Durchführung beteiligt ist.
- Der Akteur befindet sich *ausserhalb* des zu realisierenden Systems.
- Im Anwendungsfall-Diagramm wird ein menschlicher Akteur *nicht als Person* sondern als *Rolle* modelliert. Derselbe physische Mensch kann mehrere Rollen haben.
- Akteure haben, wie der Name sagt, eine aktive Rolle bei der Durchführung eines Anwendungsfalles: Sie geben Daten ein, treffen die notwendigen Entscheide und führen die einzelnen Schritte des Anwendungsfalles durch. Passive technische Systeme (z.B. Drucker, Datenbank) sind keine Akteure.

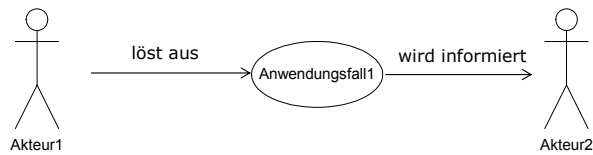
In der Regel werden in Anwendungsfällen menschliche Akteure modelliert. Bei rein technisch begründeten Anwendungen, wo verschiedene Komponenten in einem Netzwerk aktiv zusammenarbeiten, können natürlich auch Geräte Akteure sein. Beispielsweise können in einer Produktionsumgebung eines Industriebetriebes Sensoren, Roboter, Überwachungsgeräte etc. sehr wohl Akteure sein.

# Akteur, Aktivitätsart

- Die Beziehung eines Akteurs zum Anwendungsfall ist oft 'löst aus und führt durch'. Sie wird deshalb oft auch nicht näher gekennzeichnet:



- Soll die Art der Beziehung genauer gekennzeichnet werden, wird das wie folgt dargestellt:

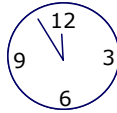


# Akteur-Typen



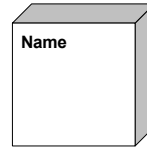
## Person

Löst in einer bestimmten *Rolle* einen Anwendungsfall aus.



## Zeitgeber

Löst stellvertretend für einen Personen Akteur einen Anwendungsfall aus.



## Fremdsystem

Löst als selbständiger Akteur einen Anwendungsfall aus.

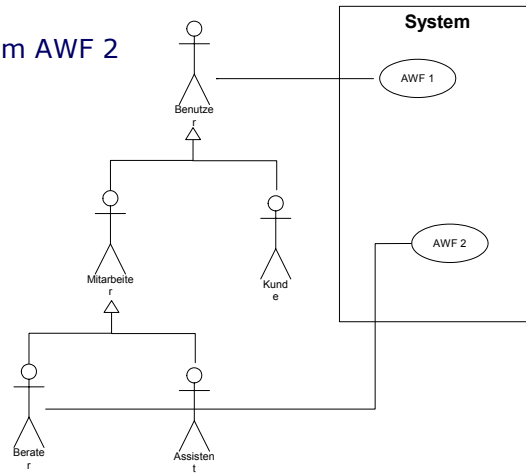
👉 Übung Anwendungsfalldiagramm

## Beziehung zwischen den Akteuren

- Verschiedene Akteur-Rollen können häufig zu einer generellen (abstrakten) Rolle zusammengefasst werden oder es können aus einer generellen (abstrakten) Rolle verschiedene spezialisierte Rollen abgeleitet werden.
- Der generelle Akteur repräsentiert immer alle darunterliegenden speziellen Akteure.
- Eine Beziehung zwischen einem generellen Akteur und einem Anwendungsfall gilt immer auch für die von der generellen Akteur-Rolle abgeleiteten (spezialisierten) Rollen.

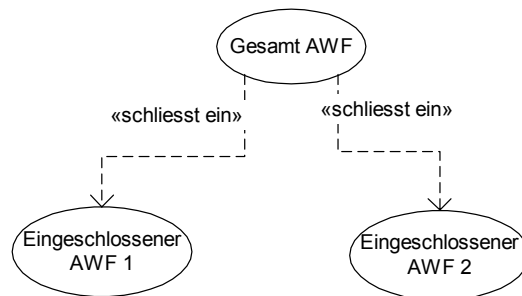
## Beziehung zwischen den Akteuren ff

- Alle Arten Benutzer können am AWF 1 beteiligt sein.
- Nur Berater können am AWF 2 beteiligt sein.



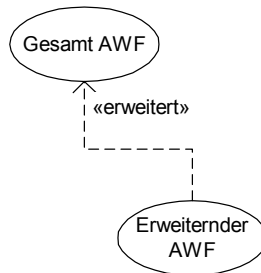
# Eingeschlossener Anwendungsfall

- Ein übergeordneter Anwendungsfall schliesst weitere, bestehende Anwendungsfälle mit ein.
- Die eingeschlossenen Anwendungsfälle sollten prinzipiell auch als eigenständige AWF in Frage kommen (oder es bereits sein)



# Erweiterter Anwendungsfall

- Ein Anwendungsfall kann unter bestimmten Bedingungen einen anderen erweitern (ergänzen).
- Im Gegensatz zum eingeschlossenen AWF kommt ein erweiternder AWF nur manchmal zur Ausführung.

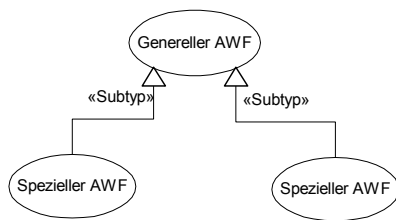


schliesst ein = include

erweitert = extends

# Spezialisierter/Generalisierter Anwendungsfall

- Von einem Anwendungsfall existieren eventuell mehrere spezielle Varianten (Unteranwendungsfälle, Subtypen).



- Beispiel 1: Bestellung aufnehmen -> Bestellung über Internet aufnehmen, Bestellung über Telefon aufnehmen.
- Beispiel 2: Kunde aufnehmen -> Privatkunde aufnehmen, Firmenkunde aufnehmen.

## Spezialisierter/Generalisierter Anwendungsfall ff

- Der Vorteil einer Hierarchisierung liegt darin, dass in vielen Situationen nur mit dem generellen AWF gearbeitet werden kann.
- Zeigt vom Akteur eine Beziehung zu einem generellen AWF, bedeutet dies, dass der Akteur auch die speziellen AWF's auslösen kann oder daran beteiligt ist.
- Der generalisierte AWF repräsentiert immer seine verschiedenen speziellen (abgeleiteten) AWF.
- Ist der generelle AWF soweit abstrahiert, dass er selbst nicht mehr zur Anwendung kommt (nur noch eine Hülle bildet), sondern nur noch seine speziellen AWFs, heisst er *abstrakter* Anwendungsfall.

# Anwendungsfall-Beschreibung

- Die grafische Darstellung kann auch auf konzeptioneller Stufe nicht alle Information zu einem AWF aufnehmen, sie dient lediglich der Übersicht.
  - Mit einer gegliederten, vorstrukturierten Beschreibung kann ein Anwendungsfall in einer vernünftigen Detaillierung definiert werden.
  - Bei einer AWF- Beschreibung ist Gewicht zu legen auf:
    - Beschreibung der *Tätigkeiten*
    - Beschreibung *eines* durgehenden Ablaufes (roter Faden)
    - Zustand des Systems *vor* dem Start, *Auslöser oder* Auslöse- Bedingung
    - Zustand des Systems *nach* dem Anwendungsfall
  - Vermeiden:
    - Komplexe Verzweigungen (if - else, do - while) beschreiben
    - Beschreibung von zuvielen Ausnahmen
- 
- Aktivitäts und Zustands - Diagramm verwenden

# Anwendungsfall-Beschreibung

- Name des Awf
  - Akteure
  - Vorbedingungen
  - Auslöser
  - Nachbedingungen
  - Ergebnisse
  - Invarianten
  - Nicht-Funktionale Anforderungen
  - Ablaufbeschreibung
  - Ausnahmen
  - Variationen
  - Geschäftsregeln
  - Allgemeine Systemdienste
  - Fremdspezifikationen
  - Ansprechpartner
  - Änderungshistorie
  - Offene Fragen
  - Dialogbeispiele, Muster
  - Verweise auf Diagramme
  - Weiterentwicklungen [Erg. Schmidhauser]
- 1-5 Seiten

# AWF-Beschreibung, Beispiel 1

- **Name des AWF**  
Buch aufnehmen
- **Akteure**  
Ein Mitarbeiter der Bibliothek
- **Vorbedingungen**  
Mindestens ein Exemplar des Buches liegt vor.
- **Auslöser**  
Ein oder mehrere Exemplare sind per Post eingetroffen.
- **Ergebnis**  
Das Buch wurde im Katalog erfasst, alle Exemplare etikettiert, am korrekten Ort abgelegt und zur Ausleihe freigegeben.
- **Nicht-funktionale Anforderungen**  
Für die Etikettierung sind Plastiketiketten DIN 123 zu verwenden.

# AWF-Beschreibung, Beispiel 1 ff

## ■ **Ablauf**

1. Akteur: Dewey-Klassifikation festlegen.
2. Akteur: Buch im Katalog eintragen mit Titel, ISBN, Autoren, Jahr, Verlag
3. Akteur: Exemplarnummer(n) zuteilen.
4. System: Etikette erstellen
5. Akteur: Etikette am Buchrücken aufkleben.
6. Akteur: Buch im Katalog freigeben zur Ausleihe.
7. Akteur: Exemplare gemäss Klassifikation in Bibliothek einordnen.

## ■ **Variationen**

1. Buch ist bereits vorhanden, es sind neue Exemplare eingetroffen: Nur Schritte 4-6 durchführen.

## ■ **Regeln**

Exemplarnummerierung: Beginnen bei grösster vorhandener Nummer + 1

## AWF-Beschreibung, Beispiel 2

- **Name des AWF**  
Buch ausleihen über Internet
- **Akteure**  
Registrierter Benutzer der Bibliothek, Mitarbeiter der Bibliothek.
- **Vorbedingungen**  
Der Benutzer muss eingeloggt sein.
- **Auslöser**  
Durch Auswahl eines entsprechenden Menüpunktes vom Bibliotheksbenutzer.
- **Nachbedingung**  
Das angeforderte Buch ist als ausgeliehen markiert. Der Benutzer bleibt eingeloggt.
- **Ergebnis**  
Der Benutzer hat ein Buch entliehen, das ihm vom Bibliotheks-Mitarbeiter manuell mit der Post zugeschickt wird. Die Auslieferung wird ihm mit einer Meldung auf dem Bildschirm bestätigt.
- **Invarianten**  
Der Benutzer muss sich für den ganzen Ablauf in einer gültigen Benutzer-Session befinden.

# AWF-Beschreibung, Beispiel 2 ff

## ■ Nicht-funktionale Anforderungen

1. Die Verwendung von JavaScript für die Verifikation von Benutzer-Eingaben ist erlaubt. Bei nicht eingeschaltetem JavaScript muss das System jedoch mit vermindertem Benutzerkomfort trotzdem funktionsfähig sein.

## ■ Ablauf

1. Benutzer wählt im Browser den Menüpunkt 'Buch ausleihen'.
2. Eingeschlossener Anwendungsfall 'Katalog sichten' durchführen:
  1. System liefert Suchmaske aus.
  2. Benutzer gibt Suchbedingungen und gewünschte Sortierfolge ein.
  3. System liefert Titelauswahl zurück.
  4. [optionale wiederholung] Benutzer verlangt Detailansicht eines Titels oder wechselt von der kompakten zur ausführlichen Listenansicht.
  5. [optionale wiederholung] Schritte 2 – 4.
3. Benutzer wählt für einen Titel 'anfordern'.
4. Eingeschlossener Anwendungsfall 'Buch anfordern' durchführen:
  1. System sucht freies Exemplar und bestätigt die Entleiherung mit Meldung der Entleiherungsnummer und Stand des PunkteAbo und dirigiert Benutzer auf Hauptmenu zurück.
  2. System benachrichtigt Bibliotheks-Mitarbeiter für die Auslieferung.

# AWF-Beschreibung, Beispiel 2 ff

## ■ **Variationen**

1. Es ist kein ausleihbares Exemplar verfügbar: Der Benutzer erhält eine Mitteilung und muss zu einem späteren Zeitpunkt eine Ausleihe versuchen.
2. Der Benutzer hat keine Punkte mehr in seinem PunkteAbo: Das System verzweigt in den Erweiterungs-Anwendungsfall 'Punkte-Abo lösen' und kehrt anschliessend zurück.

## ■ **Ausnahmen, Fehler**

1. Ungültige Benutzer-Session wegen zu langer Inaktivität zwischen zwei Eingabeschritten. Anwendungsfall muss neu ausgelöst werden.
2. Bei fehlenden Eingaben wird eine Meldung ausgegeben und das System kehrt auf die letzte Eingabe-Maske zurück unter Beibehaltung der bisherigen Angaben.
3. Der Ablauf kann vom Benutzer jederzeit abgebrochen werden.

## ■ **Regeln**

Für die Entleihe eines Buches wird 1 Punkt abgebucht (Diese Grösse muss nur in einer Konfigurationsdatei veränderbar sein).

## ■ **Allgemeine Systemdienste**

Suchmaske für Bücher, kompakte Bücherliste, ausführliche Bücherliste.

Für Erweiterungs-Anwendungsfall 'Punkte-Abo' lösen ist eine zertifizierte SSL Kommunikation notwendig.

## AWF-Beschreibung, Beispiel 2 ff

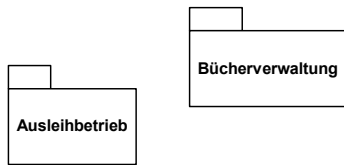
- **Fremdspezifikationen**  
Internationale Dewey-Klassifikation für Bücher (als Suchkriterium in Suchmaske)
- **Ansprechpartner**  
Bibliothekar X, Administrationsleiterin Y, Bibliotheksmitarbeiter Z
- **Änderungshistorie**  
Erstellt 15. 2. 2002 A. Schmidhauser
- **Offene Fragen**  
Soll die Zeitspanne zwischen 'Buch ausleihen' und tatsächlicher Auslieferung des Buches vom System überwacht werden? z.B. durch Einführen eines Anwendungsfalles 'Auslieferungen prüfen', der periodisch vom System ausgelöst wird.
- **Dialogbeispiele, Muster**  
HTML-Muster für Spezifikation der Suchbedingungen und Sortiermöglichkeiten.
- **Verweise auf Diagramme**  
Aktivitätsdiagramm 'Buch Ausleihen', Zustandsdiagramm 'Buch'.
- **Mögliche Weiterentwicklungen**  
Mehrere Bücher können in einem Warenkorb gelegt und gesammelt ausgeliehen werden. Einzel- oder Sammelrückgabe soll möglich sein.  
Verzweigung in eine Erweiterungs-Anwendungsfall Reservation, wenn ein Buch nicht ausleihbar ist.

# Pakete

- Anwendungsfälle in *Pakete* gruppieren.
  - Die Gruppierung erfolgt nach *fachlichen* Kriterien.
  - Die Gruppierungskriterien sind in den jeweiligen Paketen zu *dokumentieren*.
  - Ein Paket umfasst i.a. *ein* Anwendungsfall-Diagramm
- 
- Im Anwendungsfall-Modell sind die Beziehungen zwischen den Paketen kaum dargestellt, im Gegensatz etwa zu Paketen für Klassen-Modelle.
  - Die Definition von Schnittstellen bei Anwendungsfall-Paketen ist von untergeordneter Bedeutung.

☞ Übung Anwendungsfall-Beschreibung

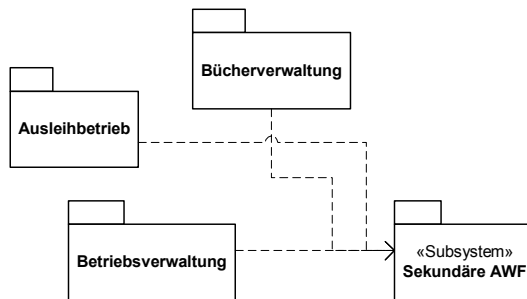
# Pakete ff



Pakete ohne Beziehungen



Pakete mit  
Abhängigkeitsbeziehung



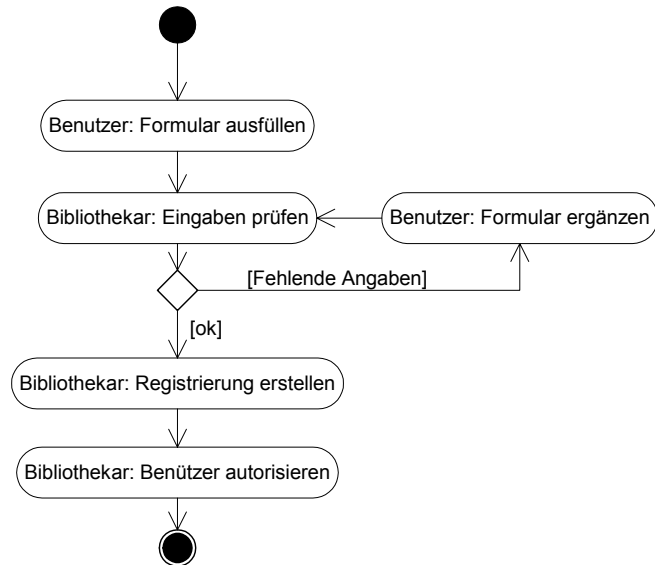
# Aktivitätsdiagramme

Aktivitätsdiagramme unterstützen das Anwendungsfall-Modell mit der Spezifikation von *Abläufen* auf der Ebene des Anwenders.

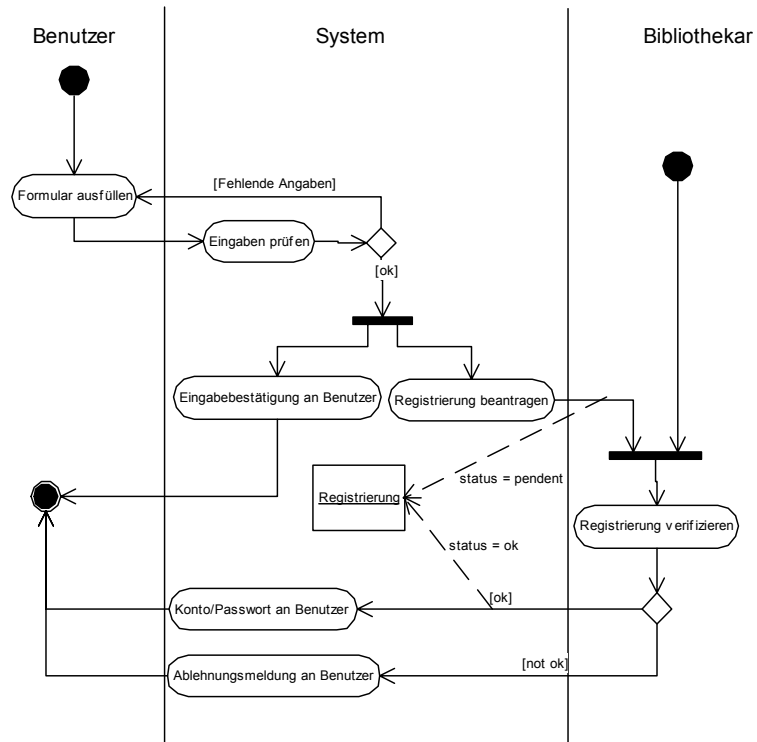
# Aktivitätsdiagramm

- Ein Aktivitätsdiagramm zeigt den zeitlichen Ablauf verschiedener Arbeitsschritte (Aktivitäten).
- Eine Aktivität ist ein Arbeitsschritt, der auf der betrachteten Ebene nicht mehr weiter hinterfragt wird und "in einem Zug" abläuft.
- Aktivitäten sind untereinander verknüpft, indem das Ende der einen den Beginn von anderen darstellt.
- Welche Aktivitäten welche anderen auslösen, ist durch Transitionen (Pfeile) gekennzeichnet.
- Aktivitäten können sequentiell, parallel, alternativ, wiederholt ablaufen.
- Die Auswirkungen der Aktivitäten auf Datenobjekte können dargestellt werden.
- Aktivitätsdiagramme beziehen sich auf konzeptioneller Ebene meist auf einen Anwendungsfall.

## Aktivitätsdiagramm für AWF 'Benutzer-Registrierung in der Bibliothek'



Beispiel AWF 'Benutzer' Registrierung über Internet'.

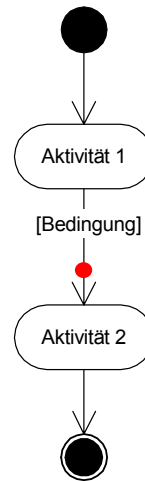


## Kommentare

- Aktivitätsdiagramm für den Anwendungsfall 'Benutzer-Registrierung'.
- Der Akteur 'Benutzer' und der Akteur 'Bibliothekar' werden gemäss diesem AD unabhängig voneinander aktiv.
- Ein Registrierungsobjekt enthält die während des Anwendungsfalles erzeugten Daten und den Status der Registrierung.
- Die Aktionen 'Eingabebestätigung' und 'Registrierung beantragen' laufen parallel ab (sind unabhängig voneinander).
- Die Bearbeitung der pendenten Registrierung ist blockiert bis der Bibliothekar aktiv wird und die Registrierung verifiziert.

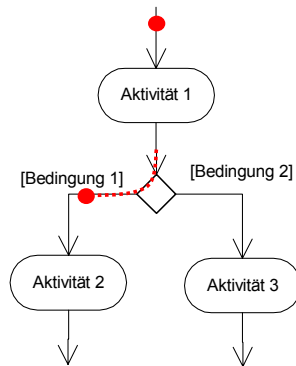
# Sequenz von Aktivitäten

- Initialisiert wird der Durchlauf durch ein Aktivitätsdiagramm durch den Benutzer oder das System. Der Startpunkt ist schwarz dargestellt.
- Wo sich die Ausführung gerade befindet kann durch einen Cursor (roter Punkt) dargestellt werden.
- Manchmal wird an das Weiterspringen zur nächsten Aktivität eine Bedingung geknüpft. Die Ausführung bleibt dann dort stehen, bis die Bedingung erfüllt ist. Beispiel: [Zeit > 22:00]
- Die Ausführung endet, wenn der Cursor in Endpunkt verschwindet.

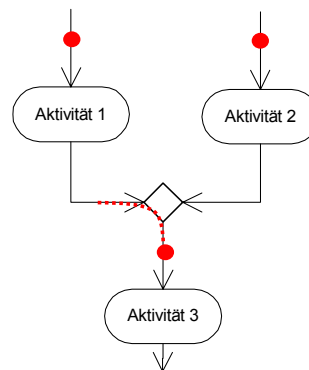


## Verzweigung von Aktivitäten

Nach Aktivität 1 wird entweder Aktivität 2 *oder* 3 durchgeführt. Die Bedingungen müssen sich gegenseitig ausschliessen.

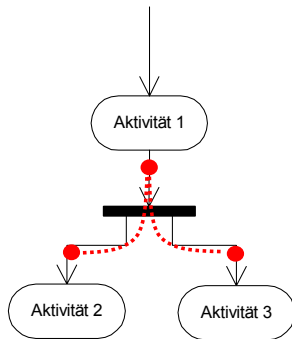


Aktivität 3 kann ausgeführt werden, *sobald* Aktivität 1 *oder* 2 beendet ist. Es wird nur *ein* Cursor weitergeführt.

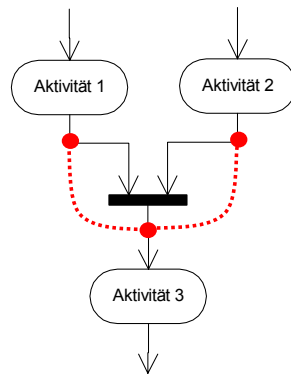


## Parallele Aktivitäten

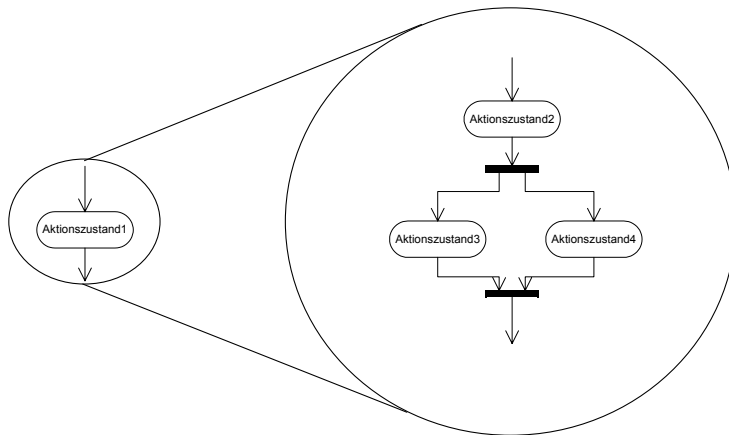
Splitting: Nach Aktivität 1 werden Aktivität 2 und 3 *gleichzeitig* ausgeführt. Es wird ein *neuer* Cursor erzeugt.



Synchronisation: Aktivität 3 wird erst in Angriff genommen wenn 1 und 2 erledigt sind. Es wird nur *ein* Cursor weitergeführt.



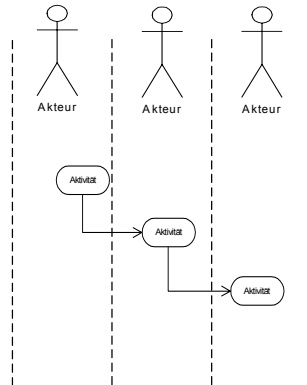
## Verfeinerung



Ein- und Ausgänge in der Verfeinerung müssen identisch zu ein und Ausgang der groben Struktur sein.

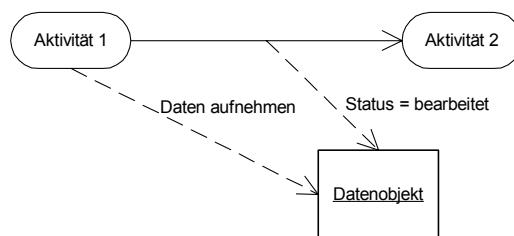
# Schwimmbahnen

- Aktivitätsdiagramme können in Schwimmbahnen (Swim Lanes) eingeteilt werden. Jede Schwimmbahn enthält dabei die Aufgaben eines verantwortlichen Akteurs. Die Zeitachse wird *tendenziell* von oben nach unten geführt. Im vorhergehenden Beispiel wurden drei Bahnen für Benutzer, System und Bibliothekar geführt.



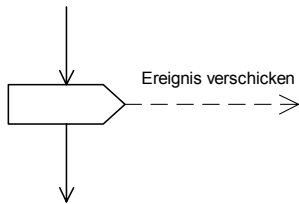
# Datenobjekte

- Aktivitäten mutieren fast immer bestimmte Datenobjekte.
- Es kann nützlich sein, diese Mutationen im Aktivitätsdiagramm darzustellen.
- Die Mutationsangabe kann von der Aktivität ausgehen oder von der Transition. Ersteres wenn die Mutation der Daten *während* der Durchführung ausgedrückt werden soll, letzteres wenn der Zustand der Daten *nach* der Durchführung ausgedrückt werden soll.

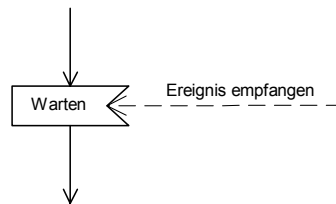


# Ereignisse

Ereignis verschicken ohne auf Antwort zu warten (z.B. Email)



Auf Eintreffen eines Ereignisses warten (z.B. Zeitereignis, OK Signal)



👉 Übung Aktivitätsdiagramm

# Logische Struktursicht

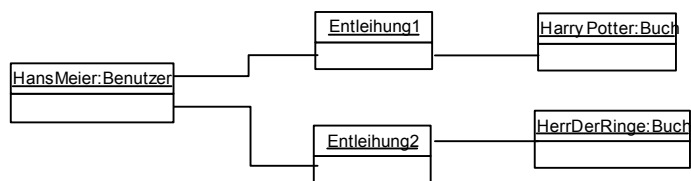
Sie Beschäftigt sich mit den eigentlichen *Dingen* (Objekten) eines Software-Systems. Dinge, welche in den Anwendungsfällen erzeugt, modifiziert und herumgereicht werden. Jedes Ding gehört zu einer Klasse, hat Eigenschaften und Beziehungen zu anderen Dingen.

# Was ist ein Objekt

- Objekte sind physisch greifbare Dinge (Buch, Person) oder konzeptionelle Dinge (Entleihung, Reservation).
- Objekte haben statische Eigenschaften (Attribute) und für jede dieser Eigenschaften einen bestimmten Wert. Beispielsweise Attribut = geburtsdatum, Wert = 13.8.1997.
- Objekte stellen Operationen (Funktionen, Methoden) zur Verfügung, die bestimmte Aufgaben lösen oder Berechnungen durchführen, meist aufgrund der eigenen Attributwerte oder solcher "verwandter" Objekte. Beispielsweise die Methode berechneAlter().
- Ein Objekt ist klar adressierbar. Auf konzeptioneller Ebene hat es einen Namen (*derKunde*), auf Ebene der Programmiersprache einen Variablennamen oder einen Identifier (Speicheradresse, Nummer).
- Da ein Objekt sowohl ein Gedächtnis (Attributwerte) wie auch ausführende Fähigkeiten hat, ist es quasi eine autonom funktionierende Einheit. Das Arbeiten mit solchen Einheiten hat die Software - Entwicklung revolutioniert.

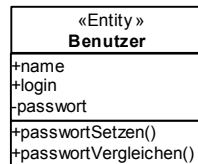
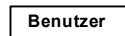
# Was ist ein Objekt ff

- Attribute und Operationen heissen Eigenschaften eines Objektes.
- Eigenschaften eines Objektes können öffentlich oder privat sein.
- Die öffentlichen Eigenschaften können von aussen direkt benutzt/aufgerufen werden. In der konzeptionellen Entwicklungsphase interessieren nur diese.
- Private Eigenschaften tragen Hilfsinformation oder Hilfsfunktionen, die von aussen nicht benutzt oder aufgerufen werden können. Sie sind nur anderen Operationen des Objektes selbst zugänglich.
- Grafische Darstellung von Objekten werden gebraucht, um *exemplarische* Situationen darzustellen.



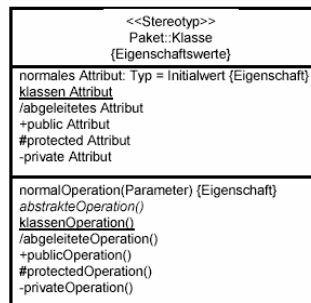
# Was ist eine Klasse

- Eine Klasse repräsentiert eine Menge gleichartiger Objekte.
- Eine Klasse ist eine *Strukturvorgabe* (Schablone, Template) für diese Objekte. Sie besitzt einen Namen, Attribute (Datenfelder) und Operationen (Funktionen, Methoden) der Objekte, welche sie repräsentiert.
- Aufgrund der Klassendefinition können neue Objekte dieser Klasse erzeugt werden.
- Alle Objekte einer Klasse haben dieselben Attribute und Operationen, aber unterschiedliche Werte ihrer Attribute.
- Eine Klasse wird manchmal auch ObjektTyp genannt.
- Grafische Darstellung einer Klasse:



# UML Notation für Klasse

- UML ermöglicht sehr konzeptionelle bis in die letzten Details gehende Angaben. Für eine Klasse sind im Extremfall alle folgenden Angaben möglich:



# Was ist ein Klassen-Modell

- Ein Klassen-Modell beinhaltet folgende Teile:
  1. Eines oder mehrere Diagramme mit den Klassen und ihren Beziehungen untereinander.
  2. Textuelle/formale Beschreibung der Aufgaben jeder Klasse, sowie ihrer Kooperationspartner.
- Der Detaillierungsgrad reicht von simpel bis sehr detailliert, je nach Phase und Entwicklungsabschnitt.
- Beispiel eines einfachen Klassen-Diagrammes:



# Stellenwert des Klassen-Modells

- Das Klassenmodell definiert in allgemeiner Art die *Struktur* aller Objekte, die in einem Software-System vorkommen, und Beziehungen untereinander.  
→ Business-Klassen (Business-Objekte)
- Das Klassenmodell definiert *Operationen* zur Erzeugung und Verarbeitung von Objekten.  
→ Business-Logik
- Auf jeder Stufe des Software-Entwurfs wird mit Klassenmodellen gearbeitet. In der obersten konzeptionellen Stufe spricht man vom *Business-*, *Fachklassen-* oder *Analyseklassen-*Modell. In diesem existieren nur Begriffe der Anwenderwelt, z.B. 'Vertrag'. Auf der untersten Implementationstufe kommen viele weitere, software-technische Klassen dazu, beispielsweise 'ListOfObject' oder 'DBConnection'.

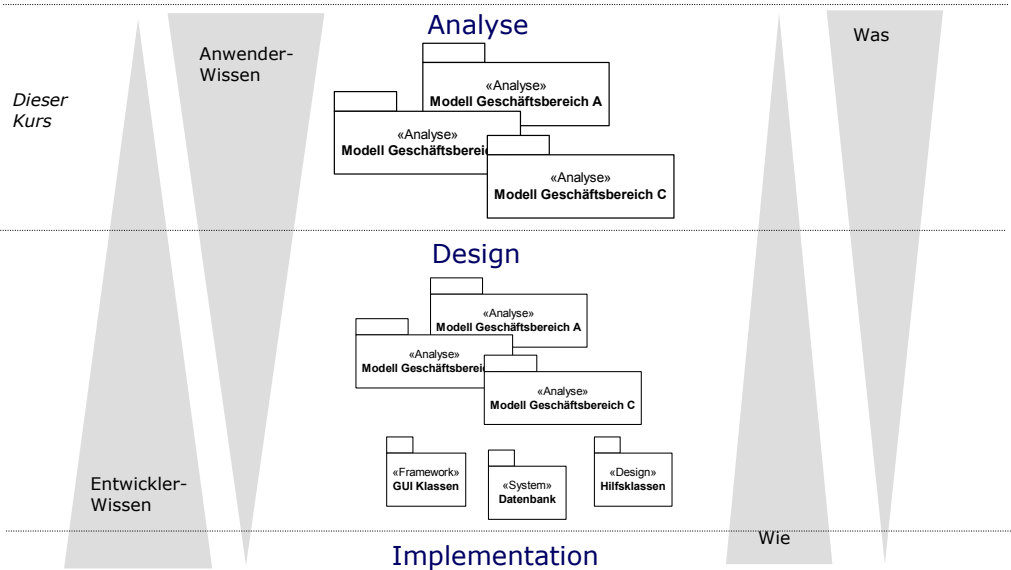
# Stellenwert des Klassen-Modells ff

- Klassen-Modelle sind ein Kernstück des Software-Entwurfs. Die Entwicklung guter Klassenmodelle ist die hohe Kunst des Software-Engineerings.

- Klassen sind nicht nur konzeptionelle Konstrukte, sondern auch *programmierbar*. In objektorientierten Programmiersprachen (Java, C++, C#) können Klassen definiert und Objekte von ihnen verarbeitet werden.

- Einige verbreitete Programmiersprachen (Visual Basic, PHP) sind nur teilweise objektorientiert. Klassenmodelle sind trotzdem ein sehr gutes Mittel für den Software-Entwurf. Es ist Sache der Entwickler, eine geeignete Umsetzung des Klassenmodells in die Programmiersprache zu finden.

# Entwicklung des Klassen-Modelles



# Business-Klassen

- Business - Klassen repräsentieren die konzeptionellste Sicht auf das System.
- Oft sind einige Business - Klassen offensichtlich, andere müssen erarbeitet werden. Es gilt auch hier: so wenig wie möglich, so viel wie nötig.
- Quellen zum Auffinden von Business - Klassen
  - Anwendungsfälle
  - Aktivitäts - und Zustands - Diagramme
  - Business - Dokumente, Formulare, Eingabemasken aus Prototypen
  - CRC Karte erstellbar
- Methoden
  - Textanalyse, Wörterbuch erstellen
  - Substantiv - Begriff -> Klasse oder Attribut,
  - Verb - Begriff -> Operation in einer Klasse,
  - identifizierbares (Seriennummer) Objekt -> Klasse

# Business-Klassen

- Einige Checks zum Auffinden von Klassen
  1. Beschreibt der Name der Klasse ein einzelnes Objekt?
  2. Beschreibt der Name wirklich einen Begriff aus der Welt des Anwenders?
  3. Beschreibt die Klasse einen Hauptaspekt oder nur ein untergeordnetes Detail?
  4. Besitzt die Klasse mehr als ein Attribut, rsp. mehr als eine Operation?
  5. Kann ein prägnanter Name gefunden werden?
- Es gibt keine Kochrezepte für das Auffinden von Klassen!
- Detaillierungsgrad von Business - Klassen beachten: Attribute ev. summarisch benennen z.B. 'PersonenDaten', keine Datentypen angeben, Methodenparameter nur wenn für Verständnis notwendig.

## Businessklasse ↔ Fachklasse

- Einige Autoren unterscheiden Business- und Fachklassen, z.B. Österreicher:

Eine Businessklasse repräsentiert einen Gegenstand, ein Konzept, einen Ort oder eine Person aus dem realen Geschäftsleben in einem *fachlichen geringen Detaillierungsgrad*, beispielsweise 'Rechnung'. Für die praktische Umsetzung ist eine Businessklasse eine Komposition mehrerer *Fachklassen*, welche alle relevanten Geschäftsinformationen tragen. Beispielsweise besteht eine Rechnung aus den Klassen 'Kunde', 'Rechnungsposition', 'Zahlung'.

- Die begriffliche Unterscheidung Businessklasse/Fachklasse soll hier nicht gemacht werden.

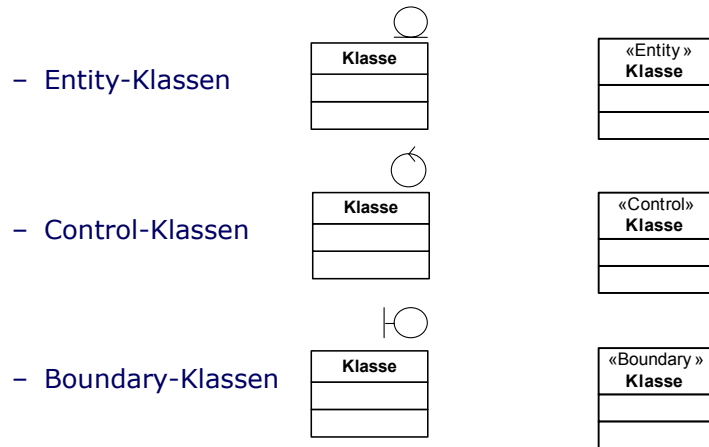
# CRC-Karte

- Für eine sinnvolle Business Klasse kann eine CRC - Karte erstellt werden (Class Responsibility Collaborator - Karte).
- Die CRC - Karte übernimmt eventuell viel aus einer Anwendungsfall beschreibung. Sie ist aber klassenbezogen, während der Anwendungsfall ablaufbezogen ist.

| Klasse                                                                                                     |                                             |
|------------------------------------------------------------------------------------------------------------|---------------------------------------------|
| Entleiher                                                                                                  | Zustandsdiagramm ZD-Entleiher               |
| Beschreibung                                                                                               |                                             |
| Eine Entleiher enthält alle Daten und Methoden für die Entleiher eines Buches durch <i>einen</i> Benutzer. |                                             |
| Verantwortlichkeiten                                                                                       |                                             |
| Was                                                                                                        | mit Wem                                     |
| Prüfen und Übernahme der Entleihparameter (von,bis, Benutzer, Buch)                                        | BuchAusleihe (Auslöser)                     |
| prüfen, ob Buch frei ist                                                                                   | Buch, Status des Buches prüfen              |
| Rücknahme des Buches                                                                                       | BuchAusleihe (Auslöser), Buch Status setzen |
| Punkte im PunkteAbo abbuchen                                                                               | PunkteAbo, Benutzer                         |
| Mahnungs-Daten unterhalten (wann, wie oft)                                                                 | BuchAusleihe (Auslöser), Buch Status setzen |

# Grundarten von Business-Klassen

- UML unterscheidet drei *Grundarten* von Klassen (Model-View-Controller Prinzip)



# Entity-Klassen

- Entity - Klassen bilden das Fundament aller Business - Klassen. Sie besitzen immer Attribute zur Aufnahme von Nutz - Information.
- Entity - Klassen partizipieren in der Regel an vielen Anwendungsfällen.
- Die Daten von Entity - Klassen werden aufgrund ihrer Wichtigkeit und damit langen Lebensdauer meist in einer Datenbank abgelegt. Sie werden daher häufig auch mit dem Stereotyp «persistent» statt «Entity» bezeichnet.
- Entity - Klassen werden nie von sich aus aktiv, sie warten auf Befehle.

# Entity-Klassen ff

- Entity- Klassen sind nicht nur das objektorientierte Gegenstück von relationalen Tabellen:
  - Sie können eine komplexere Struktur haben
  - Sie besitzen operative Verantwortung. Beispielsweise ist die Klasse 'Entleihung' dafür verantwortlich, dass sich eine Entleihung nicht mit anderen zeitlich überschneidet.
- Beispiele von Entity - Klassen im Bibliothekssystem:

| «Entity »<br><b>Benutzer</b>                |
|---------------------------------------------|
| +name<br>+login<br>-passwort                |
| +passwortSetzen()<br>+passwortVergleichen() |

| «Entity »<br><b>PunkteAbo</b>                 |
|-----------------------------------------------|
| -punkte                                       |
| +punkteStand()<br>+abbuchen()<br>+auffüllen() |

| «Entity »<br><b>Buch</b>                                                             |
|--------------------------------------------------------------------------------------|
| isbn {key}<br>titel<br>seiten {optional}<br>jahr<br>status { buchstatus }<br>autoren |
|                                                                                      |

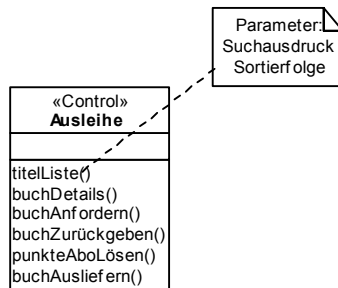
| «Entity »<br><b>Entleihung</b> |
|--------------------------------|
| von<br>bis<br>entleihen()      |

# Control-Klassen

- Control - Klassen beinhalten die Funktionalität zur Abwicklung von Anwendungsfällen. Sie benützen hierzu die Information aus Entity - Objekten, bearbeiten sie und übergeben sie an Boundary - Objekte oder umgekehrt.
- Control - Klassen teilen sich die Business - Logik mit den Entity - Klassen .
- Die Lebensdauer von Objekten einer Control - Klasse ist häufig auf die Dauer des Anwendungsfalles beschränkt. Sie beinhalten nur Daten, die für die unmittelbare Abwicklung von Operationen notwendig sind.
- Control - Klassen enthalten vorallem Methoden und wenig Attribute, im Gegensatz zu den Entity - Klassen.
- Die Control - Klassen werden in der Design - Phase mit Sicherheit zu einer ganzen Gruppe von Klassen aufspalten, welche Aufgaben wie HTML - Session - Handling, Verbindungsaufbau zu einer DB, Transaktions - Kontrolle usw. wahrnehmen.

# Control-Klassen

- Beispiel: Die Klasse 'Ausleihe' beinhaltet Operationen, welche für die Abwicklung der Anwendungsfälle 'Buch ausleihen', 'Buch ausliefern' und 'Buch zurückgeben' benötigt werden.



- Natürlich wären auch mehrere Klassen möglich, z.B. eine welche die benutzer - orientierten Operationen beinhaltet und eine welche die bibliothekar - orientierten Operationen beinhaltet.

# Boundary-Klassen

- Boundary - Klassen stehen an der Grenze des betrachteten Software Systems. Sie bedienen den Benutzer oder nehmen von ihm Aufträge entgegen. In Web - Applikationen liefern die Boundary - Klassen beispielsweise eine Webseite oder ein Eingabeformular aus. Sie nehmen auch Eingaben des Benutzers entgegen, bereiten sie auf und leiten sie an die Control - Objekte weiter.
- Auf Konzept - Ebene (Analyse) sollen Boundary - Klassen vorallem aufzeigen, was man dem Benutzer auf dem GUI (beispielsweise im Browser) präsentieren will.
- In der Designphase der Software kann es durchaus sein, dass Boundary Klassen nicht als Klassen im Sinne einer Programmiersprache implementiert werden, sondern Progrämmchen/Module einer Skriptsprache (Java Server Page, Perl, PHP, JavaScript, ASP - Seite) sind.
- Die Boundary - Klassen werden in der Design - Phase, wie die Control - Klassen mit Sicherheit zu einer ganzen Gruppe von Klassen aufspalten oder sie werden mit einer generischen Eingangs/Ausgangskomponente zusammenarbeiten, z.B. einem Webserver mit einer Servlet - Engine.

# Boundary-Klassen ff

- Je nachdem, wie allgemein die Boundary - Klassen gehalten sind, können sie für mehrere Anwendungsfälle benutzt werden. Beispielsweise kann eine Suchmaske für Bücher verschiedentlich eingesetzt werden.
- Beispiel:

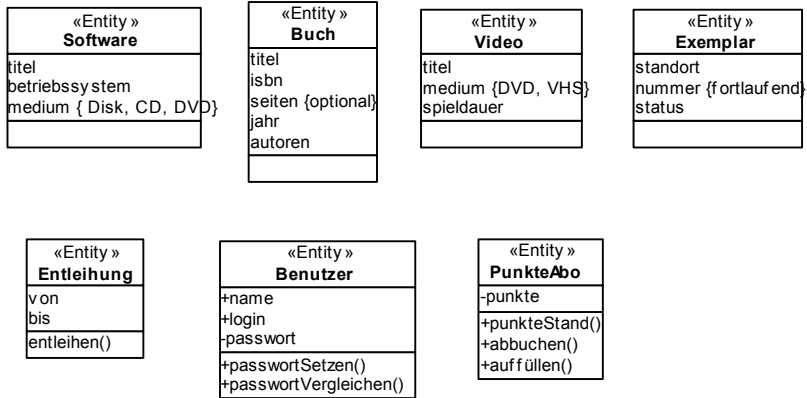


- Sehr nützlich: Mit Anwender GUI Prototypen erstellen !

# Beziehungen zwischen Klassen

- Anhand des Bibliothekssystems sollen die Möglichkeiten von UML aufgezeigt werden, Beziehungen zwischen Klassen zu formalisieren.
- Das Bibliothekssystem wird dazu noch etwas ergänzt mit folgenden Anforderungen:
  - Es können nicht nur Bücher, sondern auch Videos und Software ausgeliehen werden.
  - Zu jedem Buch gibt es mehrere Exemplare, welche an unterschiedlichen Standorten beheimatet sind.

# Klassen-Modell, lose Klassen

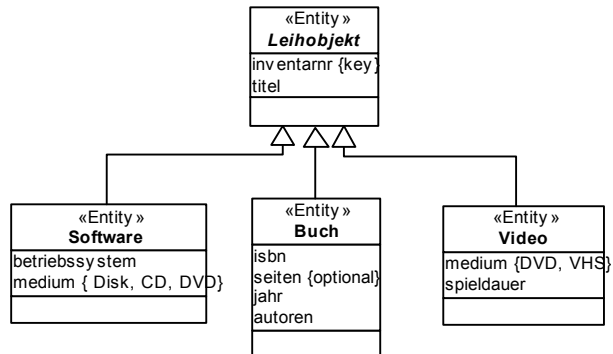


# Klassen-Modell, lose Klassen

- Erklärungen
  - Es wurde eine zusätzliche Klasse Exemplar geschaffen, da es sich offensichtlich um ein wichtiges Ding handelt in einem Bibliothekssystem.
  - Die Klasse Exemplar übernimmt das Attribut 'status', da jetzt die einzelnen Exemplare unterschiedliche Stati haben können.
- Auffallend
  - Die ausleihbaren Objekte besitzen Gemeinsamkeiten: Alle haben Exemplare, alle haben einen Titel, wahrscheinlich wird man für viele Anwendungsfälle gar nicht zwischen Büchern, Videos und Software unterscheiden müssen.
- Frage
  - Abstraktion möglich?

# Klassen-Modell, Generalisierung

- Es wird eine Abstraktion gebildet in Form einer Klasse, die Leihobjekt heisst.

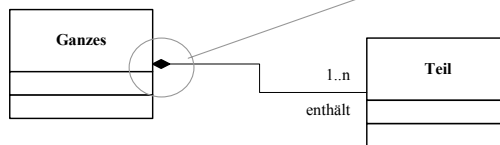


# Klassen-Modell, Generalisierung ff

- Buch, Software, Video sind alles *spezielle Arten* von Leihobjekten. Umgekehrt ist Leihobjekt eine *genereller Begriff* für Buch, Software, Video.
- 'Leihobjekt' ist eine abstrakte Klasse, sie hat selber keine Objekte (kursiv notiert).
- Gemeinsame Attribute müssen nur einmal aufgeführt werden. Die spezielleren Klassen *erben* sie.
- Beziehungen, welche von der Klasse Leihobjekt ausgehen, gelten auch für alle speziellen Klassen.
- Begriffe: Oberklasse , Unterklasse, Unterklasse ist abgeleitet von Oberklasse, Unterklasse erbt von Oberklasse, Generalisierung, Spezialisierung.

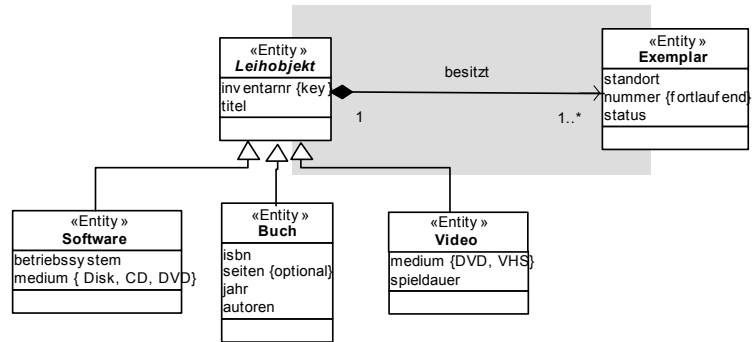
# Klassen-Modell, Komposition

- Ein Buch, resp. alle Leihobjekte, besteht aus der Beschreibung des Buches (Titel, isbn, Autoren etc.), sowie einer Menge von Exemplaren.
- Die Bindung von Buch zu Exemplar ist sehr stark: ohne Exemplare macht das Buch keinen Sinn, ohne Buch haben die Exemplare kein Gesicht mehr.
- Eine solche existentielle Abhängigkeit wird mit einem speziellen Begriff und mit einem speziellen Symbol ausgedrückt, einer *Komposition*.



- Beachte für andere Beispiele: Ein Klasse kann immer nur zu *einer* Eltern Klasse im Sinne einer Komposition gehören. Kompositionen sind häufig anzutreffen, wo physisch existierende Dinge modelliert werden, die mit einer Identität versehen sind (z.B. einer Seriennummer).

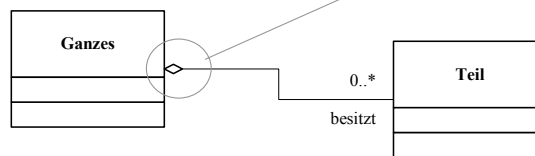
# Klassen-Modell, Komposition ff



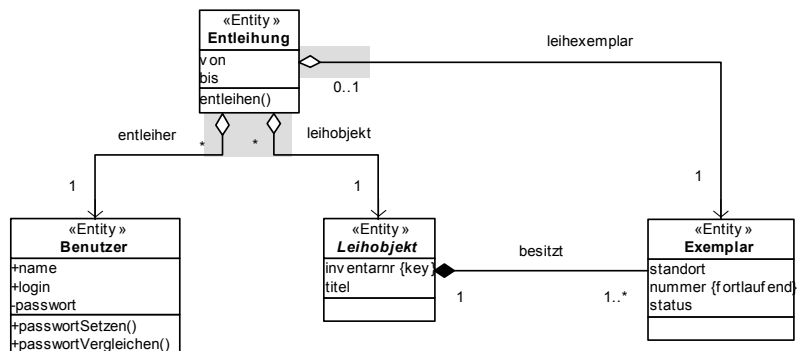
- Die Komposition ist dargestellt zwischen Leihobjekt und Exemplar. Sie gilt damit auch für Software, Buch und Video.

# Klassen-Modell, Aggregation

- Zwischen Entleihung und Leihobjekt (resp. Person) besteht eine Beziehung: Eine Entleihung ist nur sinnvoll, wenn sie sich auf ein Leihobjekt, ein Exemplar und eine Person bezieht.
- Die Beziehung ist jedoch weniger stark als zwischen Leihobjekt und Exemplar: Ein Entleihung kann gelöscht werden ohne dass das Leihobjekt seine Existenzberechtigung verliert.
- Eine nicht-existentielle Beziehung heisst in UML *Aggregation*.



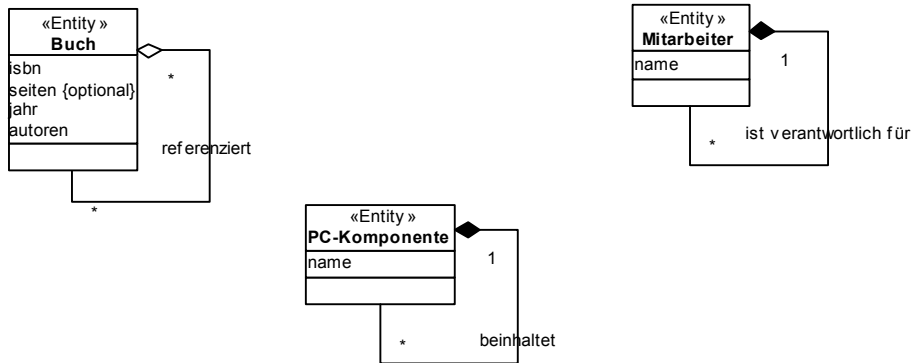
# Klassen-Modell, Aggregation ff



- Zu einer Entleiher gehört genau *ein* Leihobjekt und *ein* Entleiher.
- Ev. mehrere Entleihen für einen Benutzer vorhanden.
- Ev. mehrere Entleihen für ein Leihobjekt (mehrere Exemplare) vorhanden.
- Ein Exemplar gehört immer nur zu einer Entleiher (allenfalls keiner).
- Das Klassendiagramm drückt den *allgemeinst möglichen Fall* aus.

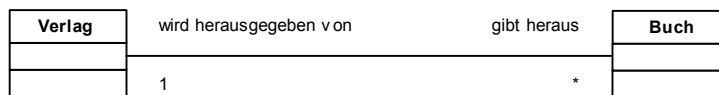
# Klassen-Modell, rekursive Beziehungen

- Es ist erlaubt und kann durchaus vorkommen, dass Klassen Beziehungen auf sich selbst besitzen. Beispiele:



# Allgemeine Assoziationen

- Möchte man keine Hierarchie oder Richtung in einer Beziehung darstellen, so spricht man in UML einfach von einer Assoziation.
- Assoziationen entsprechen am ehesten den Beziehungen (Relationen) aus der Datenbank-Modellierung.
- Beispiel:



# Wechselwirkungs-Matrix

- Wenn sich Objekte von Business-Klassen gegenseitig stark beeinflussen und die Änderung an einem Objekt Änderungen an vielen anderen bewirkt, ist oft eine Wechselwirkungs-Matrix nützlich:

| bewirkt in<br>Ändern von | Dozent | Fach     | Fragebogen  | Klasse   | Kommentar | KriterienGruppe | Kriterium | Legende | Rating   | Student  | Umfrage   |
|--------------------------|--------|----------|-------------|----------|-----------|-----------------|-----------|---------|----------|----------|-----------|
| Organisation             | -      | <b>C</b> | -           | <b>C</b> | -         | -               | -         | -       | -        | <b>C</b> | -         |
| Dozent                   | -      | -        | -           | -        | -         | -               | -         | -       | -        | -        | <b>N</b>  |
| Fach                     | -      | -        | -           | -        | -         | -               | -         | -       | -        | -        | <b>C</b>  |
| Fragebogen               | -      | -        | -           | -        | <b>C</b>  | -               | -         | -       | <b>C</b> | -        | <b>B1</b> |
| Klasse                   | -      | -        | -           | -        | -         | -               | -         | -       | -        | -        | <b>C</b>  |
| KriterienGruppe          | -      | -        | -           | -        | -         | -               | <b>C</b>  | -       | -        | -        | -         |
| Kriterium                | -      | -        | -           | -        | <b>C</b>  | -               | -         | -       | <b>C</b> | -        | -         |
| Legende                  | -      | -        | -           | -        | -         | -               | -         | -       | -        | -        | -         |
| Student                  | -      | -        | <b>N</b>    | -        | -         | -               | -         | -       | -        | -        | -         |
| Umfrage                  | -      | -        | <b>C/B2</b> | -        | -         | -               | -         | -       | -        | -        | -         |
| UmfrageTyp               | -      | -        | -           | -        | -         | <b>C</b>        | -         | -       | -        | -        | <b>R</b>  |

**C** = Beim Löschen ebenfalls Löschen (Cascade)

**R** = Gesamte Löschung abbrechen (Restriction)

**N** = Nach Löschen Referenz auf Null setzen (Nullify)

**B1** = Nach Ausfüllen/Löschen Gesamtwert neu berechnen

**B2** = Bei Abschluss Studenten anonymisieren

## Innere Funktionssicht

Zustandsdiagramme zeigen den Lebenszyklus von Objekten auf.

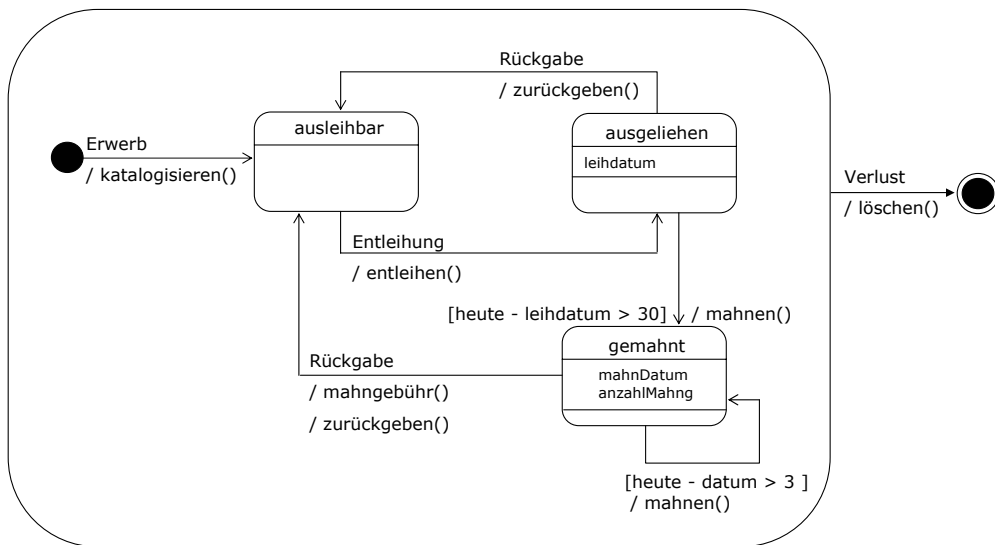
Sequenzdiagramme zeigen den zeitlichen Ablauf von Operationen und die beteiligten Objekte und Klassen für einen Anwendungsfall auf.

# Zustandsdiagramm

- Beschreibung des Lebenszyklus und Zustandes von Objekten.  
Beispiel: Status einer Bestellung.
- Zustandsdiagramme sind sinnvoll für Objekte und Systeme
  - mit einzeln definierbaren Zuständen.
  - wenn nicht jeder Zustand von jedem anderen aus erreichbar ist.
  - wenn der Zustand mit einer (dem Zustandsnamen) oder wenigen Variablen beschreibbar ist.
- Für *jedes* Objekt (Klasse) überlegen, ob ein Zustandsdiagramm benötigt wird.

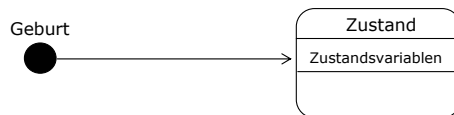
# Beispiel Bibliothekssystem

## Zustandsdiagramm für ein *Buch*



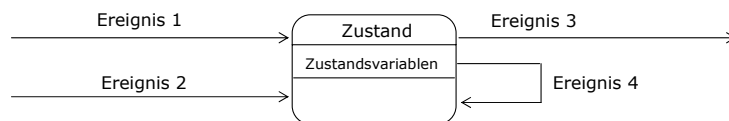
# Elemente eines Zustandsdiagramms

- Geburt
  - Das zustandsbehaftete Objekt wird erzeugt. Es tritt in seinen ersten Zustand ein. Es kann ein spezielles *Erzeugungsereignis* und eine *Erzeugungsbedingung* angegeben werden.
- Zustand
  - Das Objekt verbleibt in einem bestimmten Zustand bis es aufgrund eines Ereignisses und/oder einer Bedingung zu einem Zustandswechsel kommt. Der Zustand kann neben seinem Namen weitere Zustandsvariablen besitzen.
- Zustandsvariablen
  - Eine Zustandsvariable beinhaltet den Namen des Zustandes. Diese wird grafisch nicht aufgeführt. Weitere Zustandsvariablen, zum Beispiel wie oft der Zustand schon durchlaufen wurde, wann er eingetreten ist usw., können aufgeführt werden.



# Elemente eines Zustandsdiagramms ff

- Ereignis
  - Beschreibt den Auslöser für einen Zustandswechsel. Ereignisse können ganz konzeptionell sein, beispielsweise 'Buch wird zurückgebracht'. Ereignisse können software-orientiert sein, beispielsweise 'OK-Knopf wird gedrückt'.
- Übergang (Transition)
  - Aufgrund eines Ereignisses findet ein Übergang in den nächsten Zustand statt. Ein Zustand kann mehrere eingehende und mehrere ausgehende Übergänge besitzen. Der Übergang wird durch eine Pfeil dargestellt.



# Elemente eines Zustandsdiagramms ff

- Bedingung
  - Bedingung welche das Wechseln in den nächsten Zustand ermöglicht. Kann mit einem Ereignis kombiniert werden oder allein stehen. Im ersten Fall muss für einen Zustandsübergang das Ereignis eintreten *und* die Bedingung erfüllt sein. Im zweiten Fall muss nur die Bedingung erfüllt sein.
- Aktion
  - Wird nach Eintritt des Ereignisses und nach Prüfen der Bedingung durchgeführt. Aktionen entsprechen in der Realisierungsphase meist Funktionsaufrufen. Die Exit - Aktion aus dem Vorgänger - Zustand wird nur ausgeführt wenn die Bedingung erfüllt ist.
- Tod
  - Aufgrund eines Ereignisses oder einer Bedingung wird das Objekt im System gelöscht.



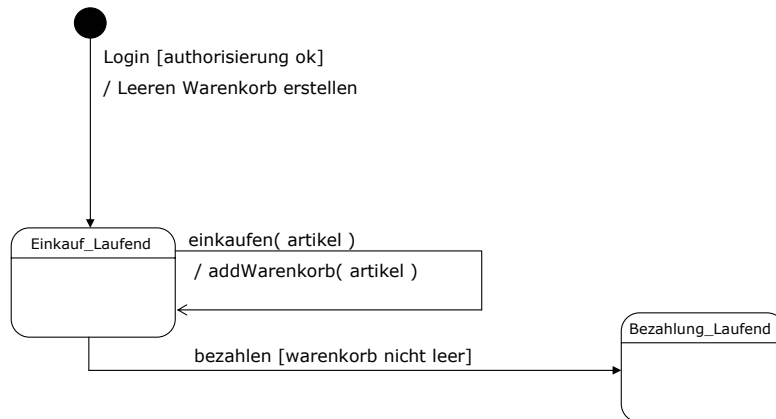
## Kommentare

- Mit einem Zustandsdiagramm lässt sich sehr gut ausdrücken, dass die Verarbeitung *bestimmter Ereignisse* nur sinnvoll oder möglich ist, wenn sich ein Objekt in einem *bestimmten Zustand* befindet.
- Tritt ein Ereignis ein, das für den aktuellen Zustand nicht definiert ist, muss es per Definition ignoriert werden.
- Im Prinzip hat *jedes* Objekt einen bestimmten Zustand. Ist dieser zu jeder Zeit und auf jeden Wert änderbar, sind Zustandsdiagramme nicht sinnvoll. Beispiel: Adressdaten einer Person.
- Entry- und Exit-Ereignis werden bei jedem Eintritt, resp. Austritt aus dem Zustand aufgerufen. Beispiele: Eintritt/Austritt in Protokolldatei speichern.

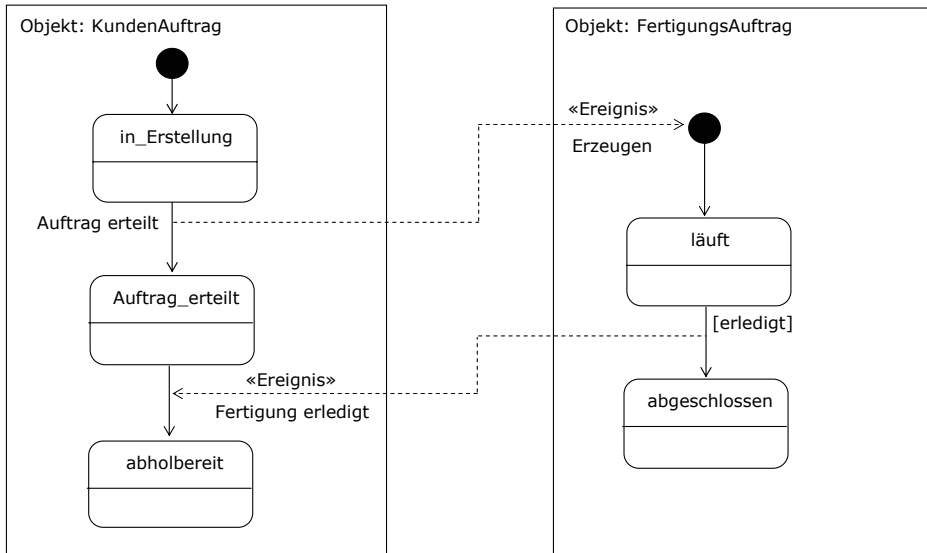
👉 Übung Zustandsdiagramm

# Parametrisierte Ereignisse

- Ereignisse und Aktionen können Parameter haben. Beispielsweise im Diagramm für ein Objekt 'Session', welches den Zustand einer Einkaufs Session in einem Webshop repräsentiert:



# Zusammenwirken von Zustandsdiagrammen



## Kommentare

- Ein Ereignis kann als Aktion weitere Ereignisse für andere Objekte auslösen. Im Diagramm ist dies durch den gestrichelten Pfeil und den Stereotyp «Ereignis» dargestellt. Achtung: der *gestrichelte* Pfeil stellt ein Ereignis, der *ausgezogene* einen Übergang dar.
- Die Art, wie der Aufruf von Ereignissen in der Software implementiert wird, ist durch konzeptionelle Diagramme nicht definiert. Es gibt verschiedenste Programmiermöglichkeiten:
  - Als synchroner Funktionsaufruf `signal( ereignisname )`
  - Als synchroner Funktionsaufruf `ereignisname()`
  - Als asynchrones Signal (`notify()`-Operation)
  - Im Rahmen einer State-Machine
  - Im Rahmen eines Eventmodelles der Programmiersprache

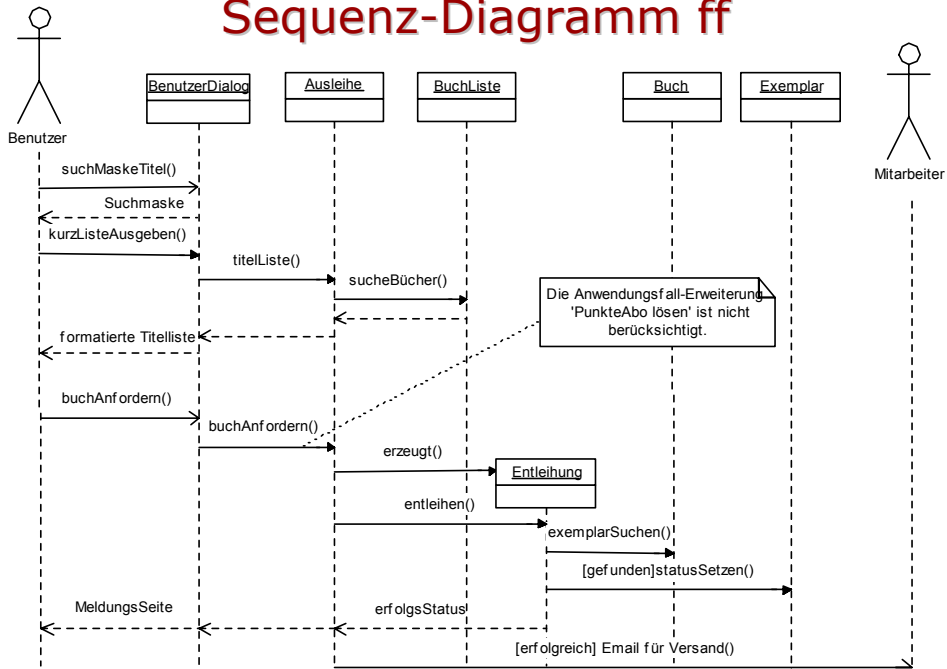
## Weitere Beispiele

- Personen mit Zivilstand
- Anfrage in ein Call-Center  
(aufgenommen, eskaliert, in bearbeitung, beantwortet)
- Hotelzimmer  
(frei, belegt, service)
- Umfrage  
(erstellt, verschickt, abgeschlossen, ausgewertet)
- Warenkorb (leer, belegt, anKasse, bezahlt)
- Session (anonym, eingeloggt, abgelaufen, ausgeloggt)

# Sequenz-Diagramm

- Sequenz-Diagramme eignen sich sehr gut für lückenlose Ablauf-Überlegungen.
- Mit Hilfe eines Sequenz-Diagrammes können sehr oft fehlende oder ungeeignete Operationen und Klassen identifiziert werden.
- Auf der Anwenderseite kann ein Sequenz-Diagramm sehr gut mit Dummy-Eingabemasken illustriert werden.
- Genau wie bei Anwendungsfällen ist mit einem Sequenzdiagramm primär *ein* bestimmter Durgang zu beschreiben, der häufigste Ablauf.
- Die Darstellung von Ausnahmen und Verzweigungen ist zu vermeiden, respektive in einem *separaten* Sequenz-Diagramm darzustellen.
- Links ist häufig der äussere Benutzer dargestellt. Dessen Aufrufe können beispielsweise in Form von URL's auf einen Webserver eingehen.

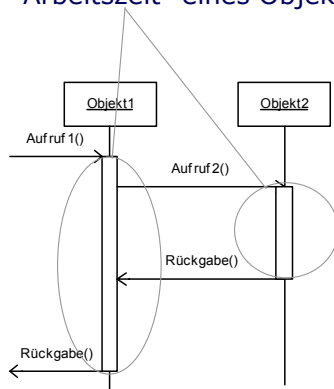
# Sequenz-Diagramm ff



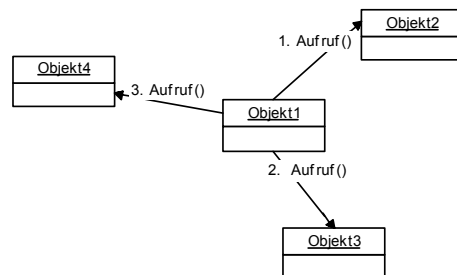
# Sequenz-Diagramm ff

- Weitere Darstellungsmöglichkeiten

"Arbeitszeit" eines Objektes



Kollaborationsdiagramm



## Physikalische Struktursicht

## Von der Analyse zum Design

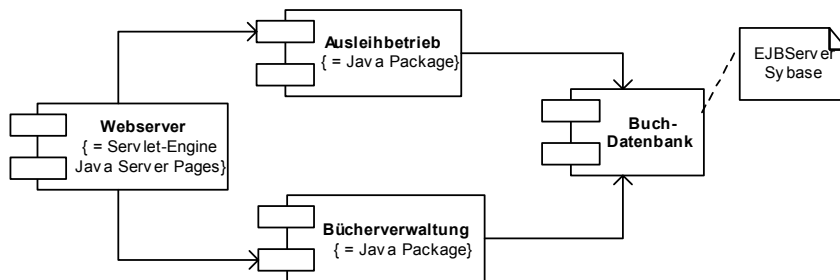
- In diesem Kurs wurden nur Konzepte betrachtet, in die der Anwender/Fachbereichs-Experte wesentlich involviert ist (Requirements- und Analyse-Phase).
- Der grosse Problemkreis der Detailausführung (Design-Phase), bezogen auf bestimmte Werkzeuge (Programmiersprachen, Plattformen, Programmbibliotheken, Datenbanken, Legacy-Systeme, Netzwerk-Gegebenheiten usw.) wurde hier nicht besprochen. Dies ist die Domäne der Software-Entwickler.
- Die Designphase beinhaltet eine Detailierung der Klassen- und Verhaltensdiagramme.
- Die Anzahl benutzter Klassen nimmt in der Design-Phase um einen Faktor 10 bis 100 zu. Damit ist auch eine wesentlich stärkere Bildung von Paketen und allenfalls Subsystemen verbunden.

# Vom Design zur Implementation

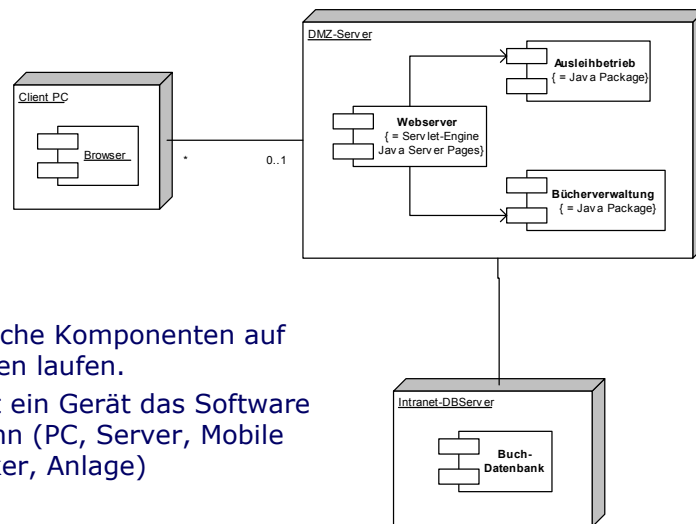
- Die Implementation beinhaltet die tatsächliche Codierung aller Klassen und die Programmierung der Methoden.
- UML beschäftigt sich nicht mit den zahlreichen Fragen und Problemen der Betriebs-Optimierung, der Pflege eines laufenden Systems, des Testens usw.
- Zwei Diagrammtypen zeigen auf, in welchen logischen und physischen Einheiten ein Software-System ausgeliefert und betrieben wird:
  - Komponenten-Diagramm
  - Einsatz-Diagramm

# Komponenten-Diagramm

- Eine Komponente ist ein installierbares und ausführbares Stück Software.
- Komponenten kommunizieren untereinander über definierte Schnittstellen.
- Schnittstellen sind typischerweise weiträumiger Art, z.B. CORBA, RMI, SQL, SOAP (Simple Object Access Protocol) über HTTP usw.



# Einsatz-Diagramm



- Zeigt auf, welche Komponenten auf welchen Knoten laufen.
- Ein Knoten ist ein Gerät das Software ausführen kann (PC, Server, Mobile Device, Drucker, Anlage)

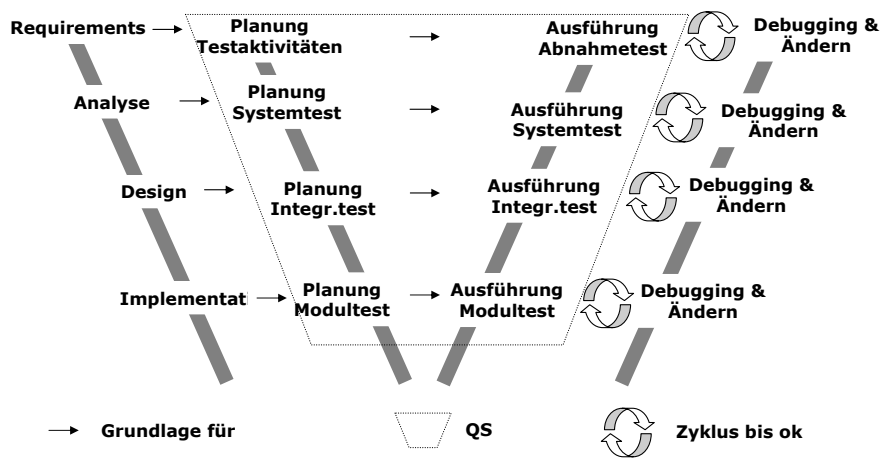
# Testen

Die Wahl der Entwicklungsmethodik beeinflusst natürlich auch das Testen der Software. Dieser Zusammenhang wird kurz aufgenommen.

eBusiness findet, unabhängig von der Entwicklungsmethodik, in einer anderen Umwelt als klassische SW-Applikationen statt. Dieser erfordert auch bezüglich Testing ein teilweise neues Augenmerk.

# Testgrundsätze

- Testen ist nicht die letzte Aktivität im Entwicklungsprozess
- W-Modell gilt generell als guter Ansatz



# Testaspekte von UML-Elementen

- Anwendungsfall
  - Grundlage des Abnahmetests
  - Sollte sowohl funktionale (werden alle fachlichen Aufgaben korrekt gelöst) wie nicht-funktionale Aspekte berücksichtigen.
  - Normalfälle, Grenzfälle und nicht-erlaubte Fälle durchführen.
- Aktivitätsdiagramm
  - alle Aktivitäten können einzeln durchlaufen und geprüft werden.
- Zustandsdiagramm
  - alle Ereignisse und Zustände sollten durchlaufen werden.
  - Robustheit testen (unerlaubte Übergänge/Zustände).

# Testaspekte von UML-Elementen ff

- Klassendiagramm
  - Assoziationen: Sind die angegebene Multiplizitäten korrekt:  
Was passiert bei  $n = -1$ ,  $n = 0$ ,  $n = 100$ ,  $n = 64000$ ?
  - Aktive Klassen: Funktionieren die parallelen Abläufe korrekt?  
Künstliche Warteschlaufen einbauen?
- Sequenzdiagramm
  - korrekte Parameterweitergabe prüfen
  - nicht mehr benötigte Methoden ermitteln
- Komponentendiagramm
  - Kann eine Komponente korrekt angesprochen werden? Wird ein Fehler gemeldet, wenn die Komponente nicht verfügbar ist?

# Testen im eBusiness

Wichtige Aspekte können bei eBusiness-Applikationen nicht durch das Projekt definiert werden oder sind schwierig abzuschätzen.

- Hard- und Software der Benutzer (PC, Browser, Internet-Zugang).  
⇒ Tests unter verschiedenen Benutzerbedingungen!
- Benutzerakzeptanz: Verärgerte Benutzer springen ab.  
⇒ Usability-Tests!
  - Wie schnell findet der Benutzer an den richtigen Ort
  - Wie oft wird er in die Irre geführt
  - Spricht ihn das Look&Feel an

# Testen im eBusiness

Lastverhalten: Wann erfährt das System die maximale Last? Wie gross ist die maximale Last?

⇒ Automatisierte Stress-Tests!

- Benutzer sind häufig anonym und die ganze Welt kann vor die Türe treten.

⇒ Sicherheitstests sind überlebenswichtig

- Manipulation von Formular- Eingaben
- Durchbruch in das umgebende System via eBiz Applikation
- Denial of Service Attacks
- Automatisches "Ausloggen" von Benutzern
- Vertraulichkeit der Übertragung

# Test im eBusiness ff

- Erreichbarkeit der Applikation
  - Ist die Site über verschiedene Provider erreichbar
  - Ist die Site in Search Engines auffindbar
  - Wie lange dauert das Laden von Seiten
- Verfügbarkeit der Site
  - Wie gut soll und wie gut ist die Site verfügbar (7\*24?)
- Nach der Entwicklung ist das Testen nicht vorbei, gerade im eBusiness!
  - Lastmonitoring
  - Zugriffspfade der Benutzer sammeln (Access Log)