

Vorlesung

Software Engineering I

Analysephase

Einführung Modellierung mit UML

Ein Modell stellt die Realität vereinfacht dar, um die wesentlichen Einflussfaktoren zu identifizieren, die für das zu betrachtende System bedeutsam sind.

Modellierung lässt sich in folgende Prozesse differenzieren:

Abgrenzung: Nichtberücksichtigung irrelevanter Objekte

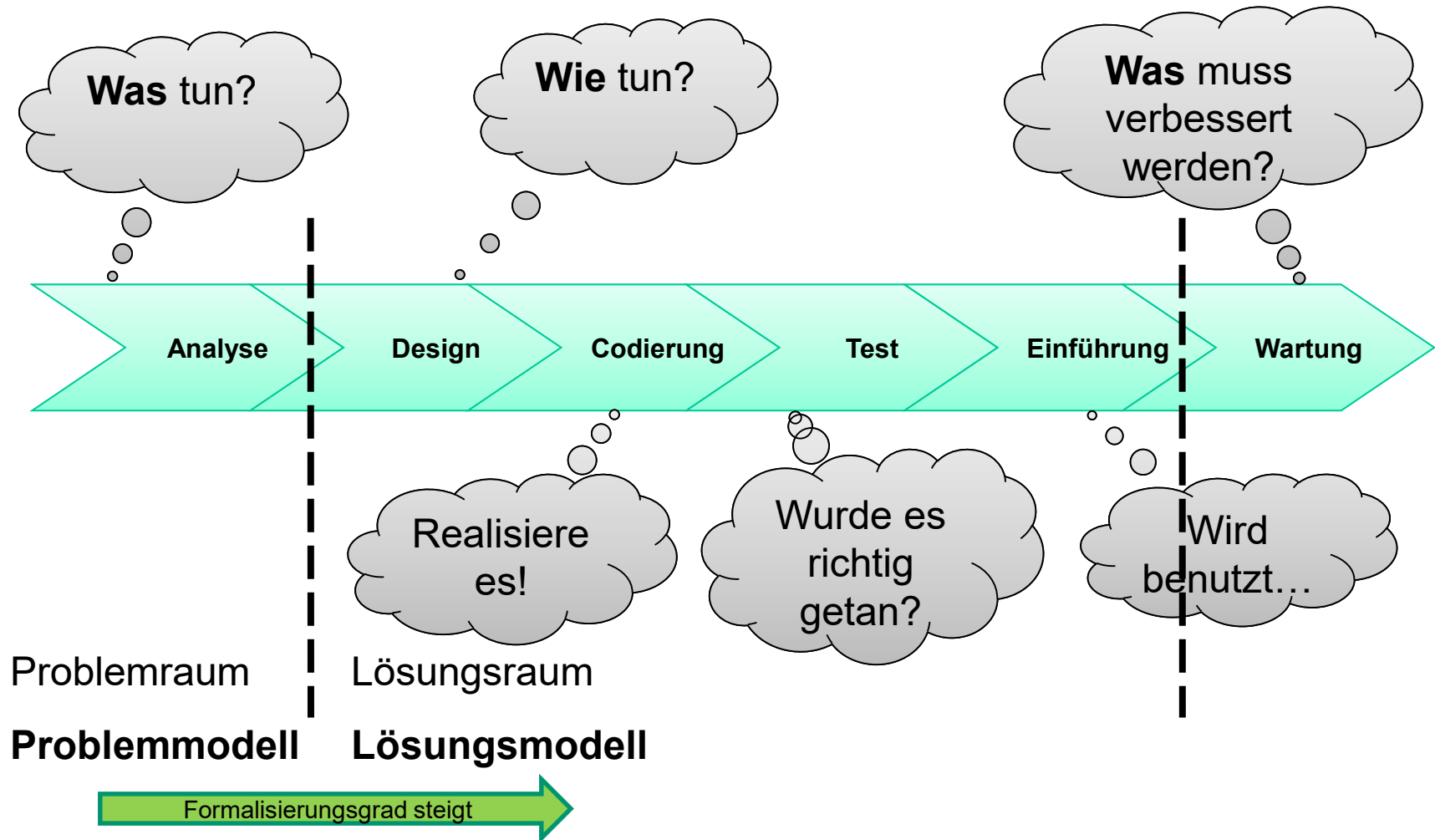
Reduktion: Weglassen von irrelevanten Objektdetails

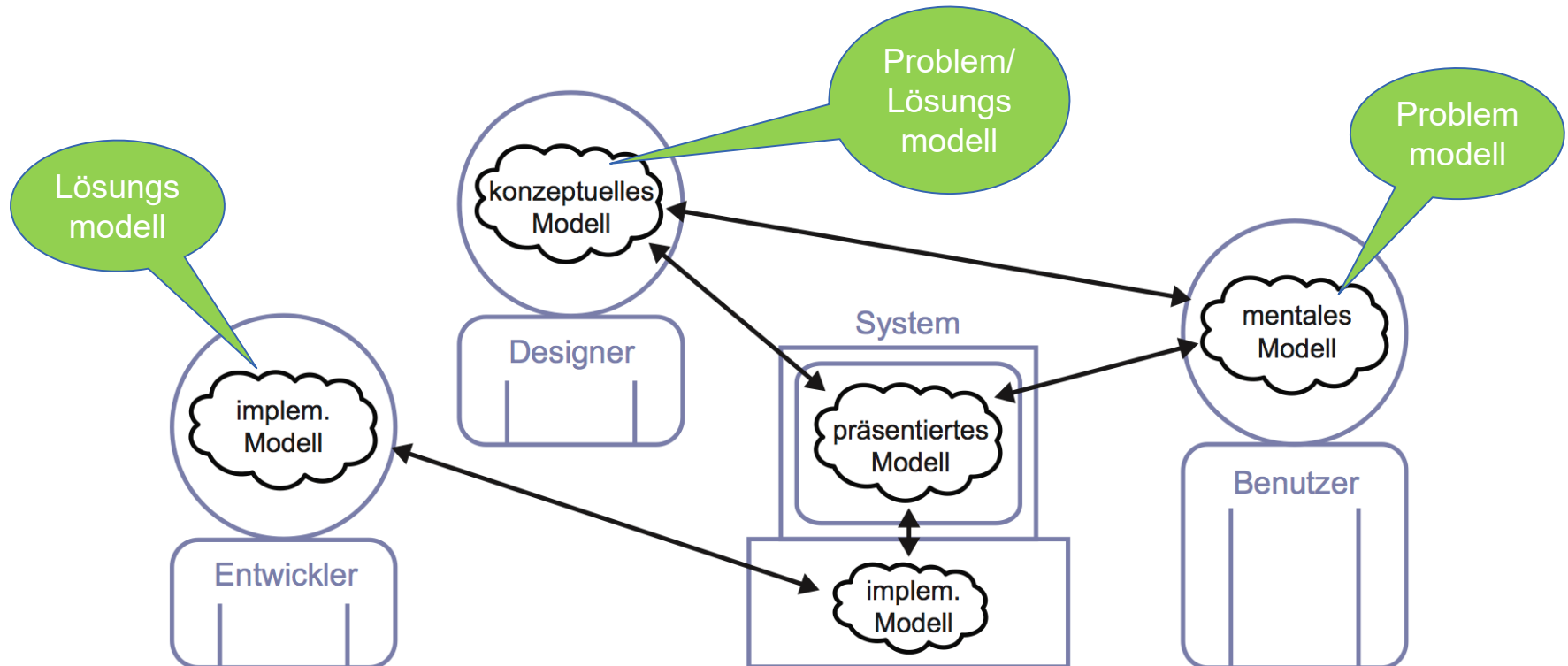
Dekomposition: Zerlegung in handhabbare Segmente

Aggregation: Zusammenfassen von Segmenten

Abstraktion: Klassenbildung bei ähnlichen Objekten

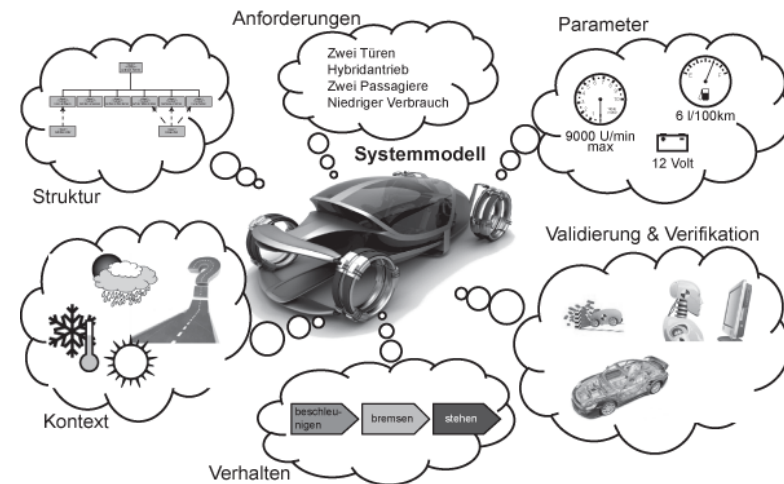
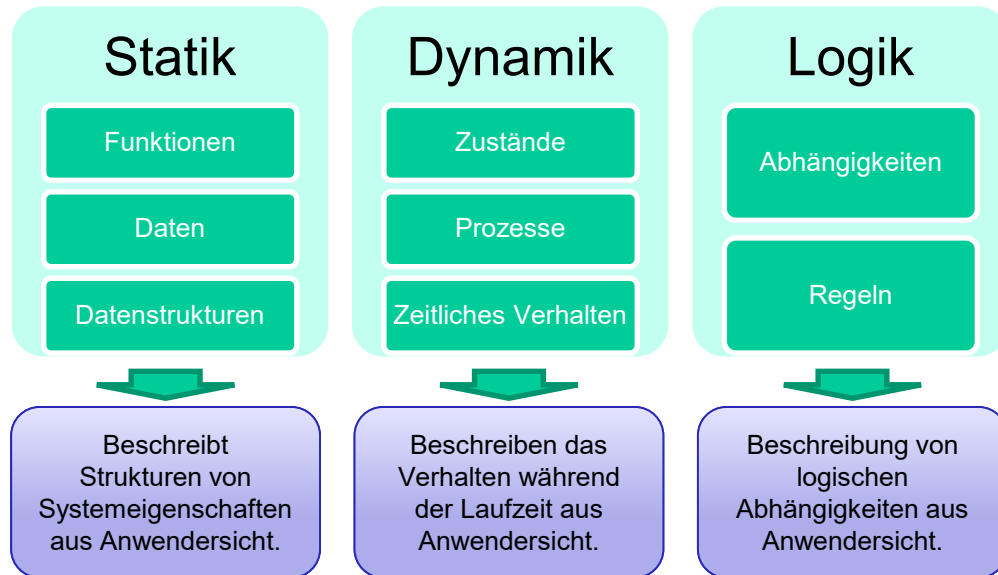
Wo wird modelliert ?



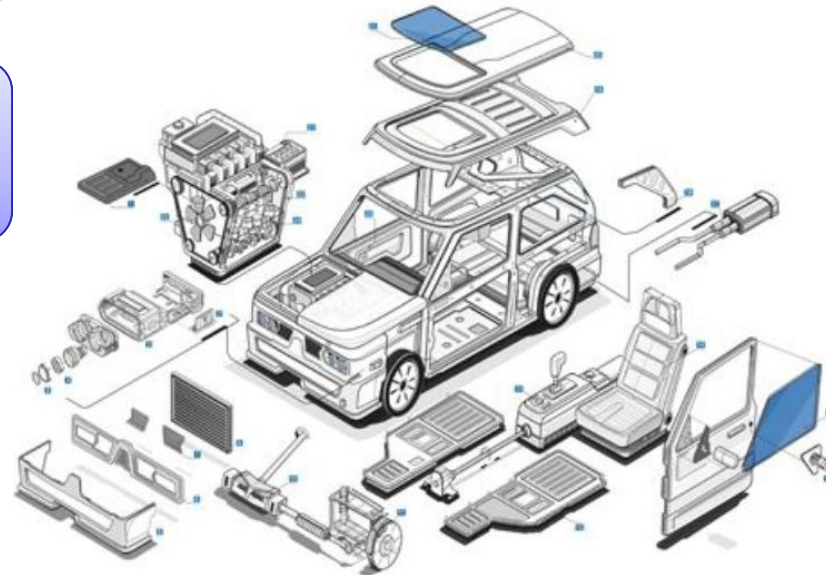
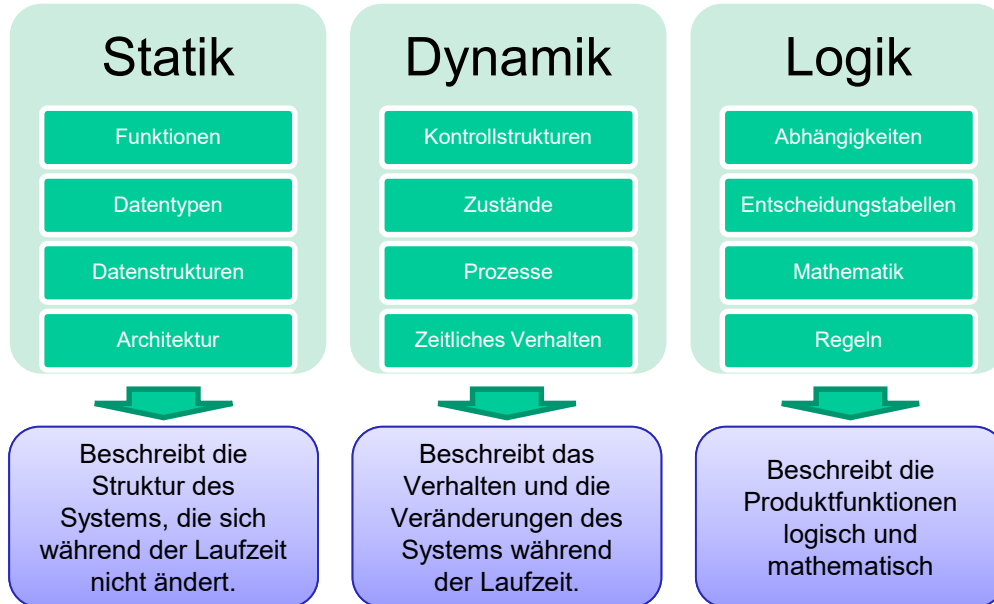


Modellierung unterstützt die Sichtbarmachung dieser unterschiedlichen Systemsichten!

Systemansichten beim Problemmodell



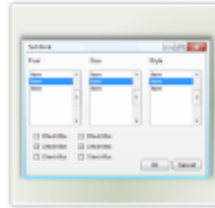
Systemansichten beim Lösungsmodell



Bekannte Modellierungsnotationen



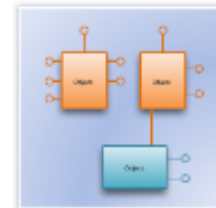
UML Model Diagram



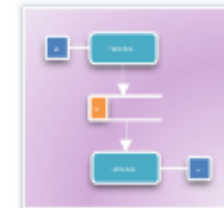
Windows 7 UI



Booch OOD



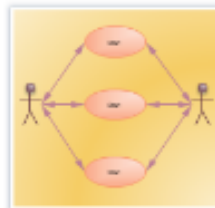
COM and OLE



Data Flow Model
Diagram



Enterprise
Application



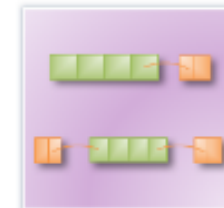
Jacobson Use Case



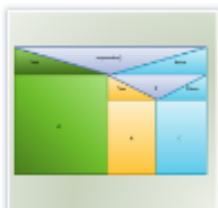
Jackson



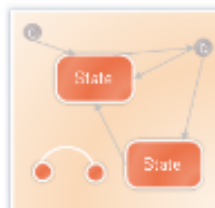
Program Flowchart



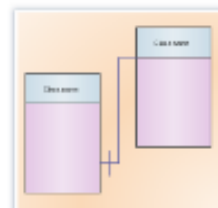
Program Structure



Nassi-Shneiderman



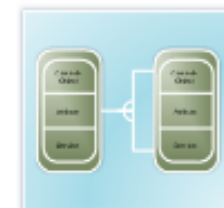
ROOM



Shlaer-Mellor OOA



SSADM

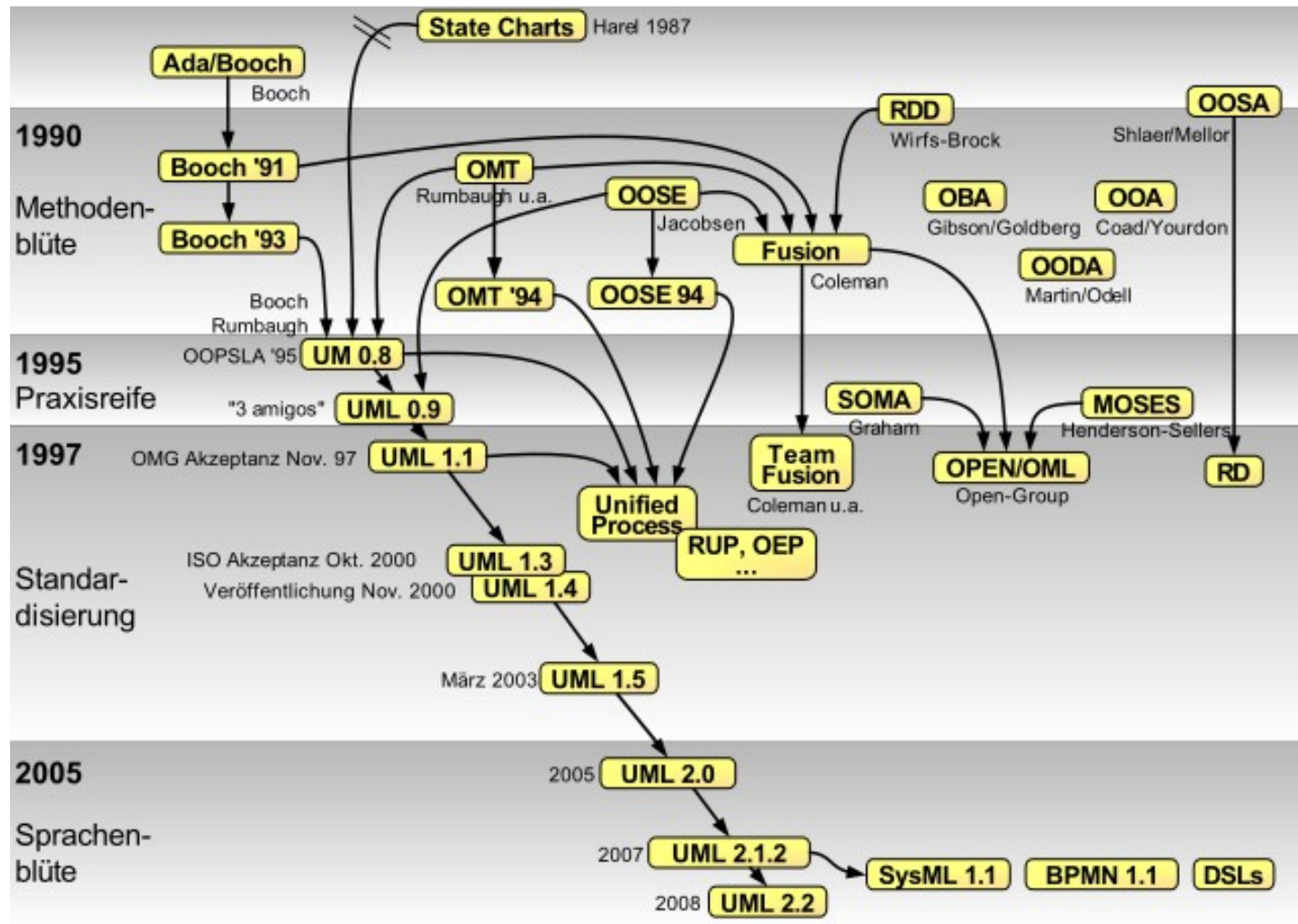


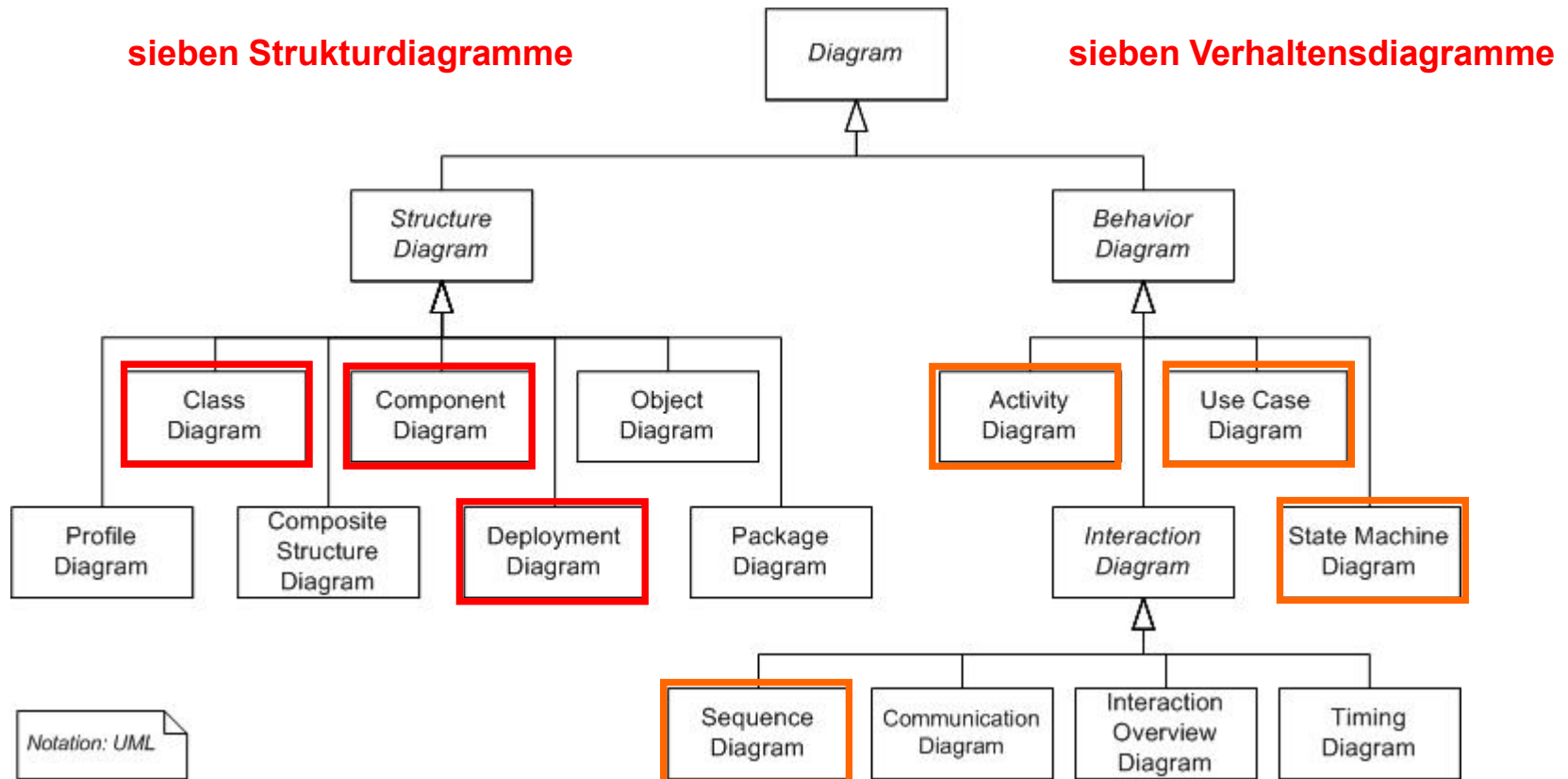
Yourdon and Coad

UML ist eine Sprache zur Beschreibung von Softwaresystemen

- **Grundgedanke:** Einheitliche Modellierungs-Notation
- Die UML besteht aus **verschiedenen Diagrammtypen**, die sich gegenseitig **ergänzen** können und verschiedene Systemaspekte **hervorheben**
- UML entstand aus mehreren bestehenden Notationen
- UML liefert Notationen, **keine Methoden** zu deren Anwendung!
- UML ist ein **Werkzeug** für Systemanalyse und Systemdesign
- UML wird durch viele CASE-Tools unterstützt

Geschichte der UML





→ Diagrammübersicht: <http://www.oose.de/uml>

→ <http://www.uml-diagrams.org/uml-24-diagrams.html>

UML: Strukturdiagramme

Klassendiagramm (Class Diagram)

(wichtigstes Diagramm: Klassen und ihre Beziehungen untereinander)

Paketdiagramm (Package Diagram)

(Gliedert Softwaresysteme in Untereinheiten)

Objektdiagramm (Object Diagram)

(Objekte, Assoziationen und Attributwerte während Laufzeit)

Kompositionsstrukturdiagramm (Composite Structure Diagram)

(Abbildung innerer Zusammenhänge einer komplexen Systemarchitektur, Darstellung von Design Patterns)

Komponentendiagramm (Component Diagram)

(Komponenten und ihre Beziehungen und Schnittstellen)

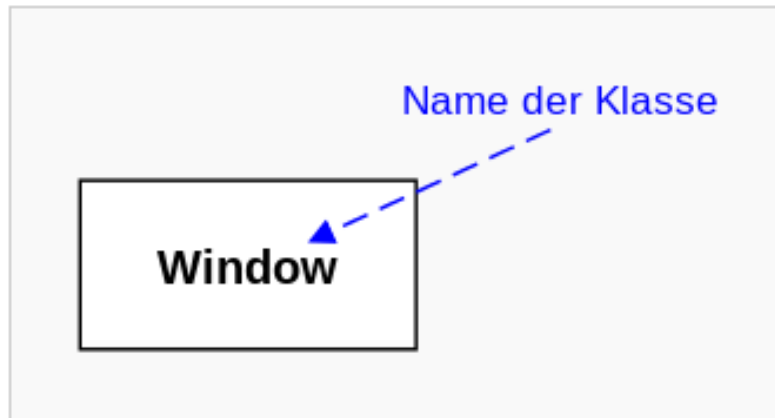
Verteilungsdiagramm (Deployment Diagram)

(Einsatzdiagramm, Knotendiagramm, Laufzeitumfeld)

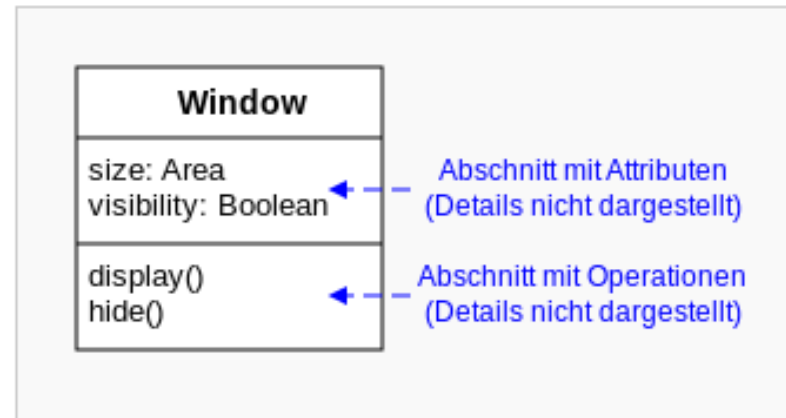
Profildiagramm (Profile Diagram)

Seit UML 2.2, um eigendefinierte Stereotypen-Sammlungen strukturieren zu können.

Klassendiagramme beschreiben Objekttypen ohne konkrete Objektdaten.
Objektdiagramme beschreiben instantiierte Objekte mit konkreten Daten.

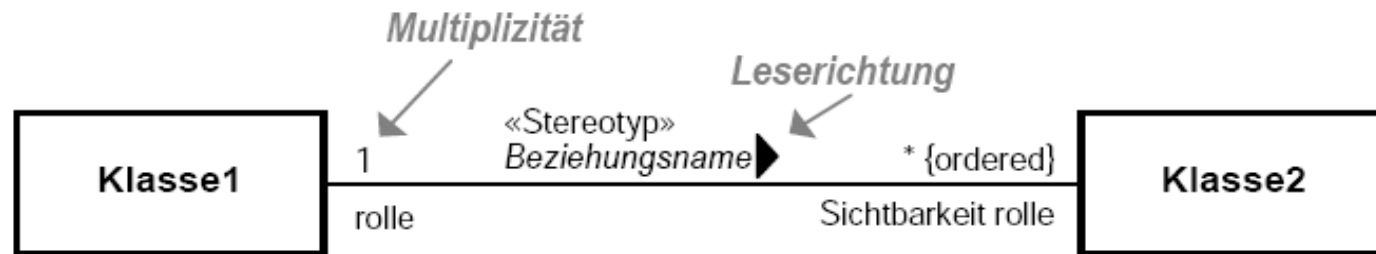


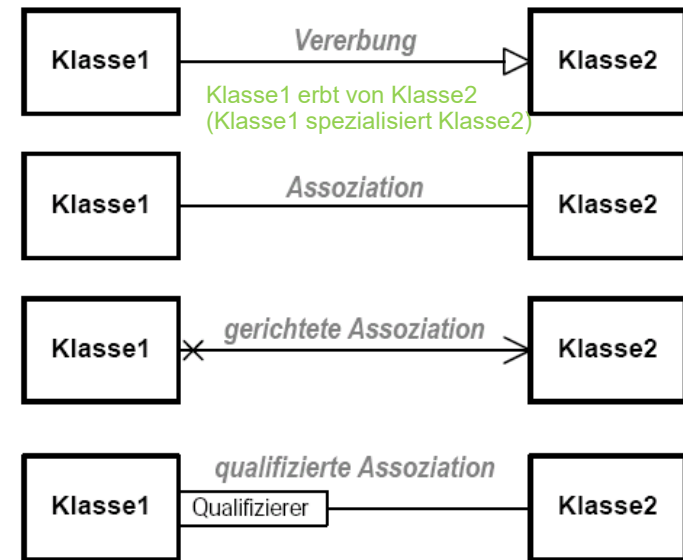
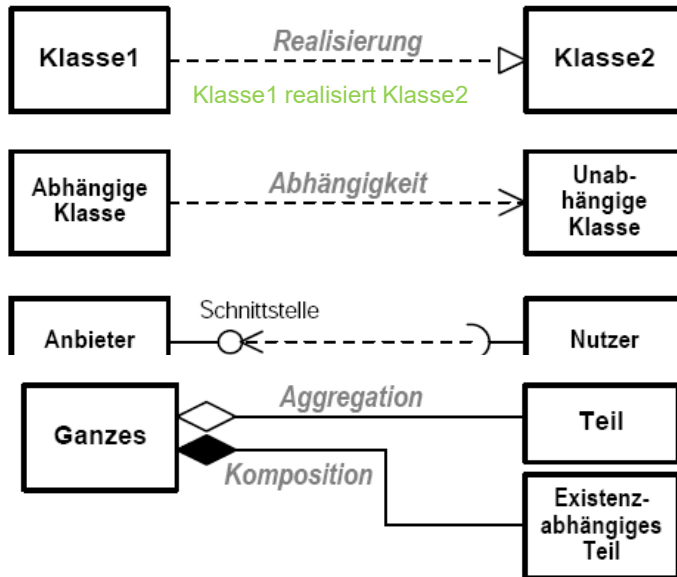
Einfachste Form der Darstellung für eine Klasse



Zusätzliche Darstellung von Attributen und Operationen

Klassenbeziehungen:





Aggregation:

Teil des Ganzen. Der Teil kann aber auch alleine bestehen.

Komposition:

„Existenzabhängiges Teil“, kann nur in Verbindung mit dem Ganzen existieren.

Vererbung:

Eine Klasse wird von einer anderen Klasse abgeleitet.

Assoziation:

Beziehungen zwischen Klassen. Eine gerichtete Assoziation erlaubt Navigierbarkeit nur in eine Richtung.

Dienen zur Darstellung von Aspekten der Implementierung.

- Dies betrifft sowohl die statische Code-Struktur als auch die (HW-) Systemstruktur. Speziell bei verteilten Anwendungen und Komponenten ist dies von besonderer Bedeutung.

Die UML sieht zwei Implementierungsdiagramme vor:

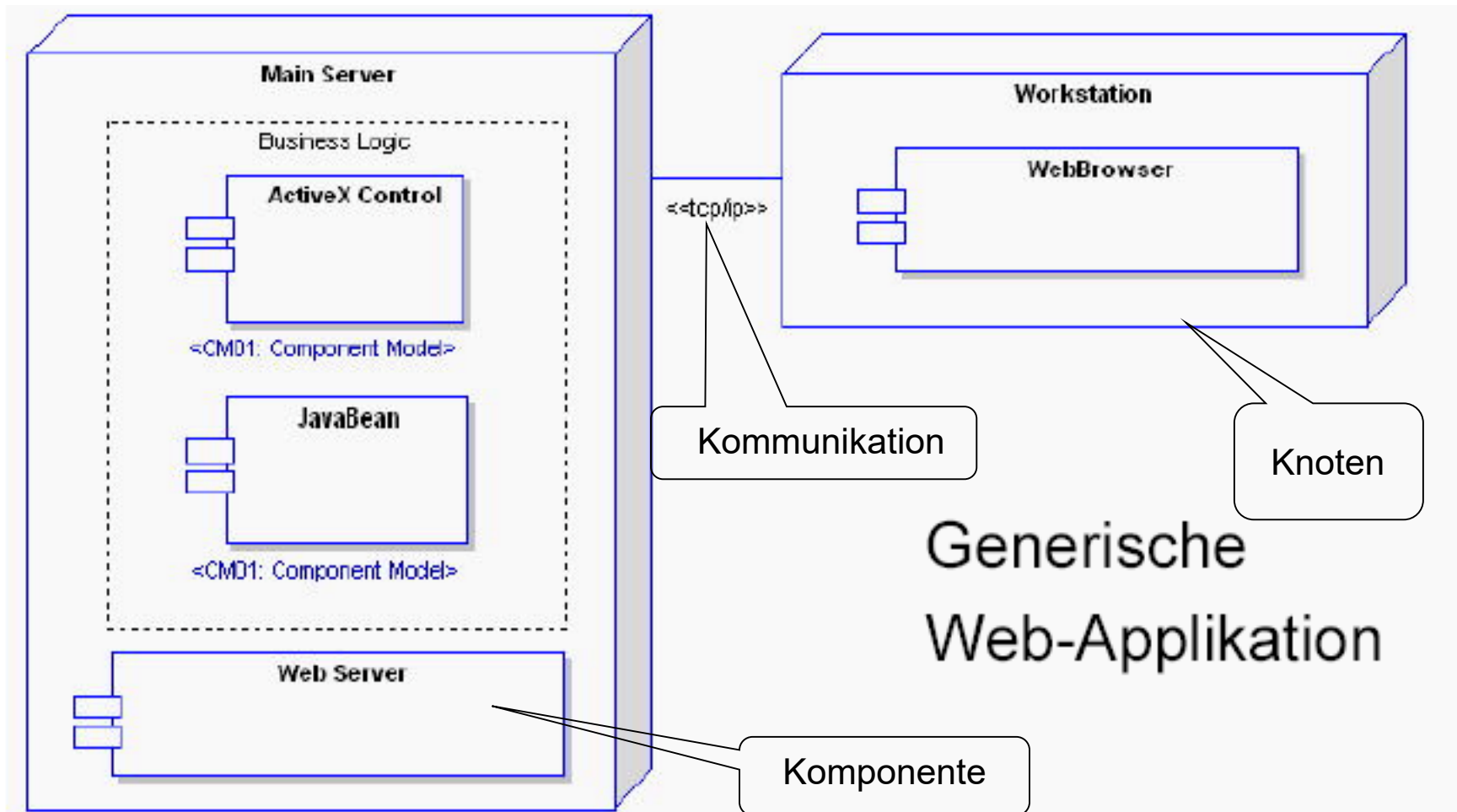
- **Verteilungsdiagramm**

- modelliert die physische (HW-)Struktur des Laufzeitsystems.

- **Komponentendiagramm**

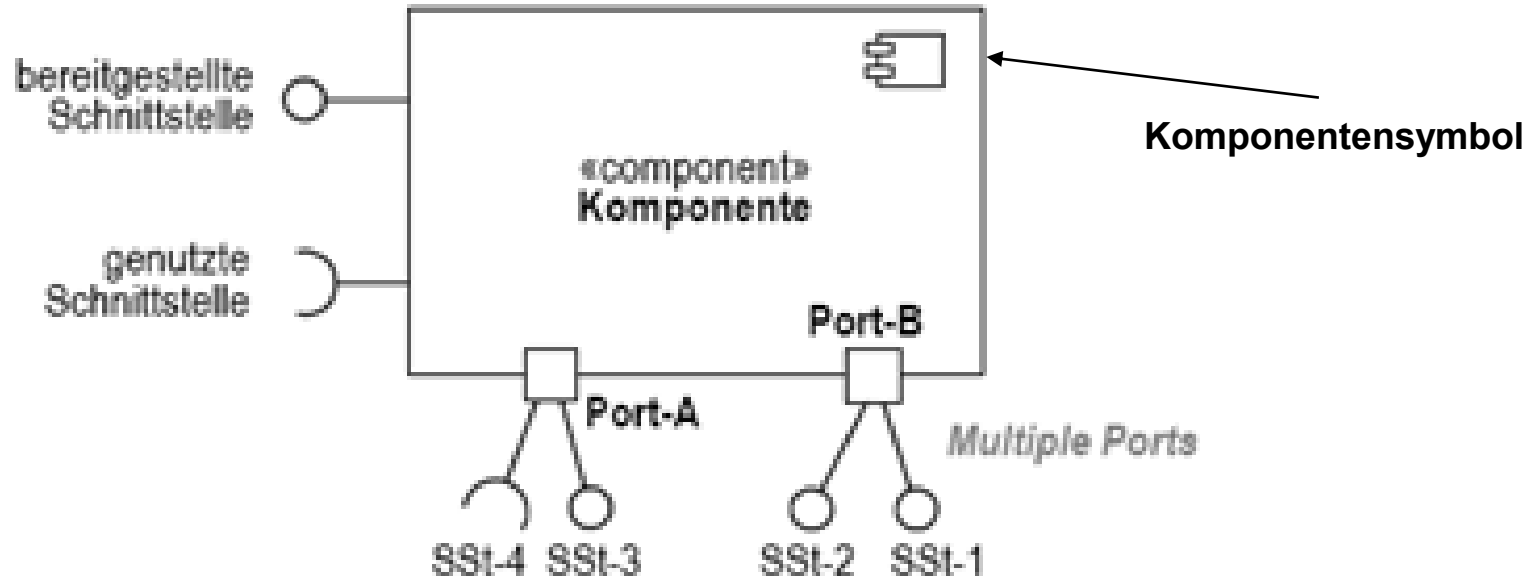
- modelliert den Aufbau des Applikationscodes

Bsp. Verteilungsdiagramm



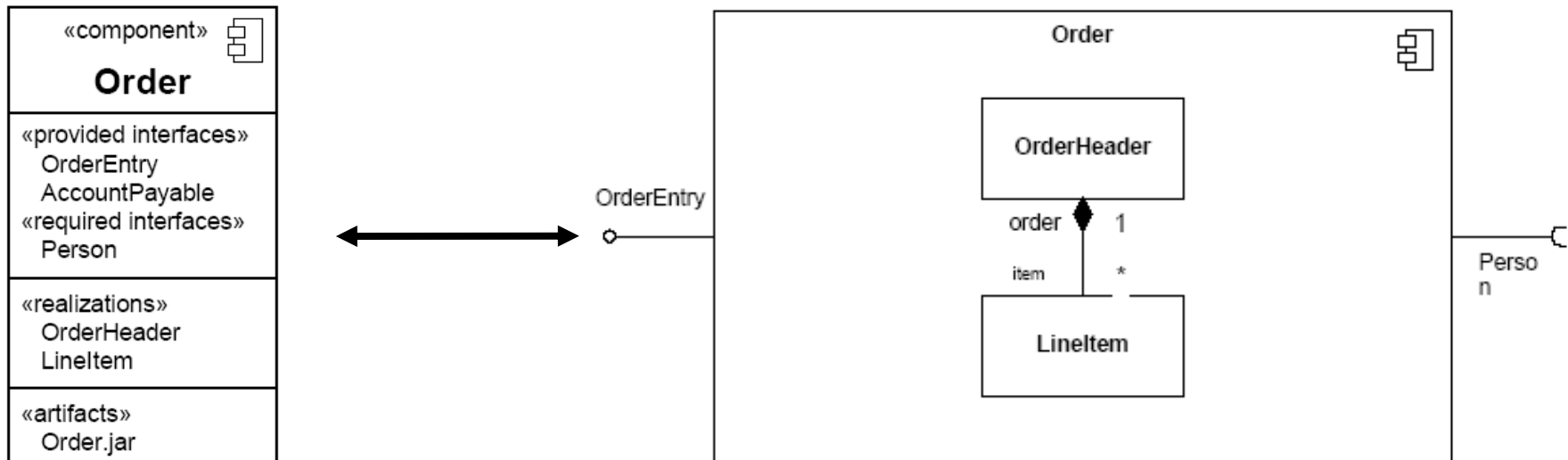
Komponentendiagramm (I)

- Eine Komponente ist eine ausführbare (ggf. austauschbare) Software-Einheit
- Komponenten bieten Schnittstellen und nutzen Schnittstellen anderer Komponenten
- Komponenten können Klassen, Pakete und Komponenten enthalten



Komponentendiagramm (II)

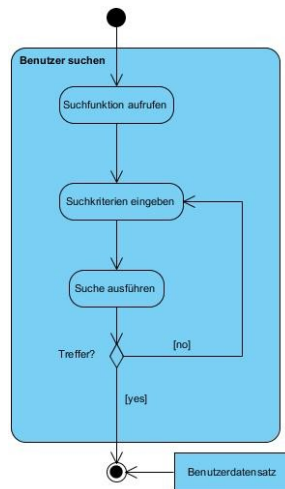
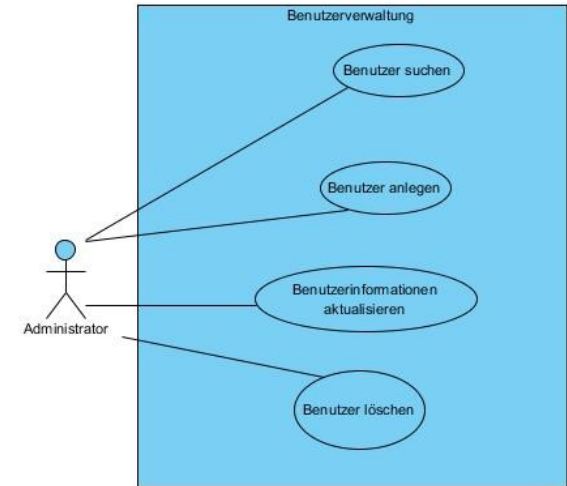
Verschiedene Darstellungsweisen:



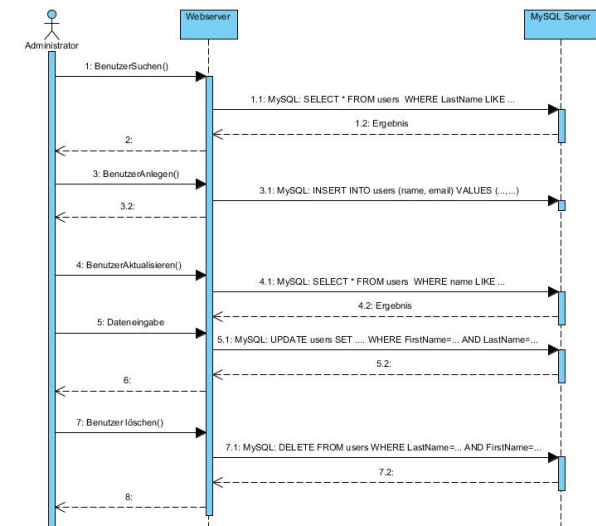
- **Anwendungsfalldiagramm** (Use Case Diagram)
(stellt Beziehungen zwischen Akteuren und Anwendungsfällen dar)
- **Aktivitätsdiagramm** (Activity Diagram)
(beschreibt Abläufe, die aus einzelnen Aktivitäten bestehen)
- **Zustandsdiagramm** (Statechart Diagram)
(beschreibt endliche Zustandsautomaten für ein Objekt oder System)
- **Sequenzdiagramm** (Sequence Diagram)
(beschreibt den zeitlichen Ablauf von Nachrichten zwischen Objekten)
- **Kommunikationsdiagramm** (Communication Diagram)
(Interaktionsdiagramm, zeigt Beziehungen und Interaktionen zwischen Objekten)
- **Zeitverlaufdiagramm** (Timing Diagram)
(Interaktionsdiagramm mit Zeitverlaufskurven von Zuständen)
- **Interaktionsübersichtsdiagramm** (Interaction Overview Diagram)
(Interaktionsdiagramm zur Übersicht über Abfolgen von Interaktionen, ähnlich Aktivitätsdiagramm)

Use Case Diagramm

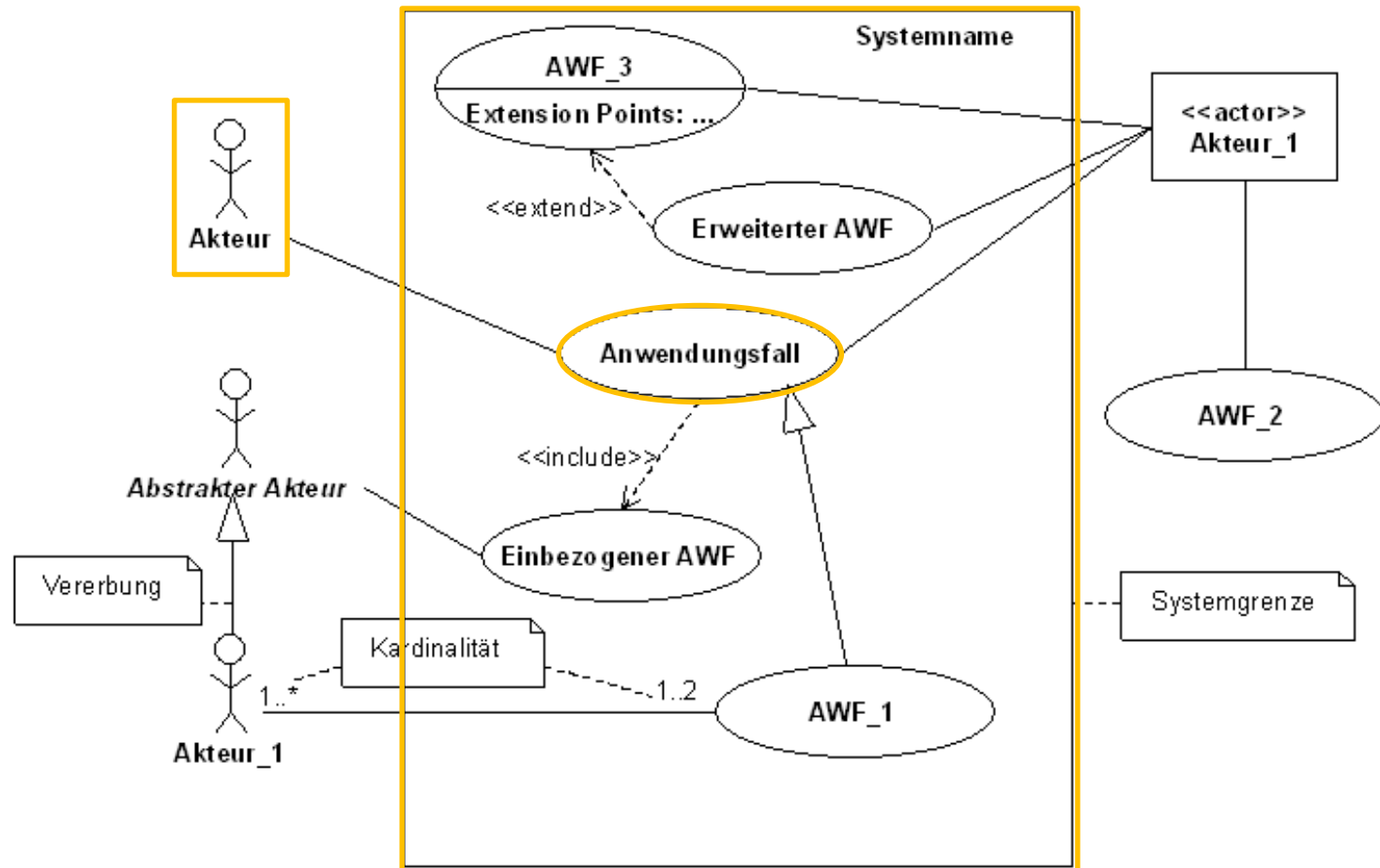
- Beschreibt das Zusammenwirken von Aktoren [bspw. Personen] mit einem System
- Beschreibt wesentliche Funktionen des Systems, die hervorgehoben und zueinander in Beziehung gesetzt werden können.
- Wird oft textuell in einer Liste beschrieben, mit Standard- und Alternativablauf

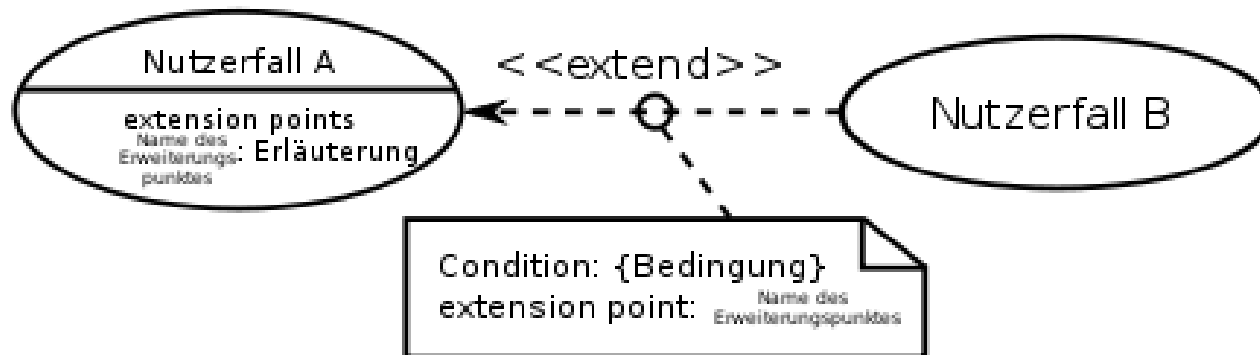
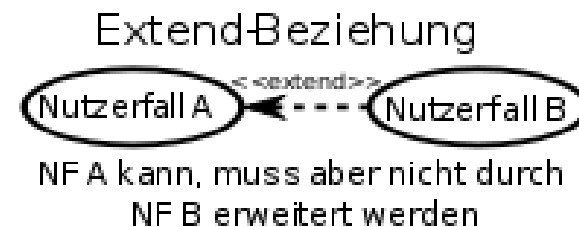
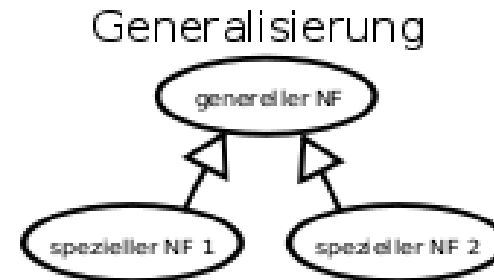
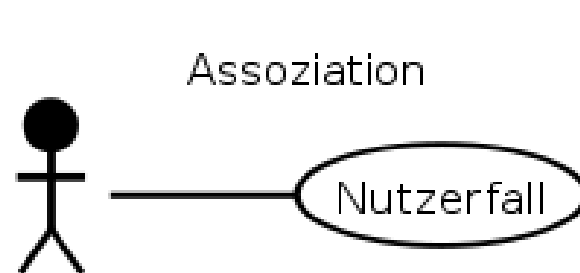


Das **Use-Case-Diagramm** der UML bietet keine Möglichkeit, eine Reihenfolge der Ausführung festzulegen. Es muss deshalb bspw. durch ein **Aktivitäts- oder Sequenzdiagramm** ergänzt werden, um eine vollständige Anwendungsfallbeschreibung zu ermöglichen!



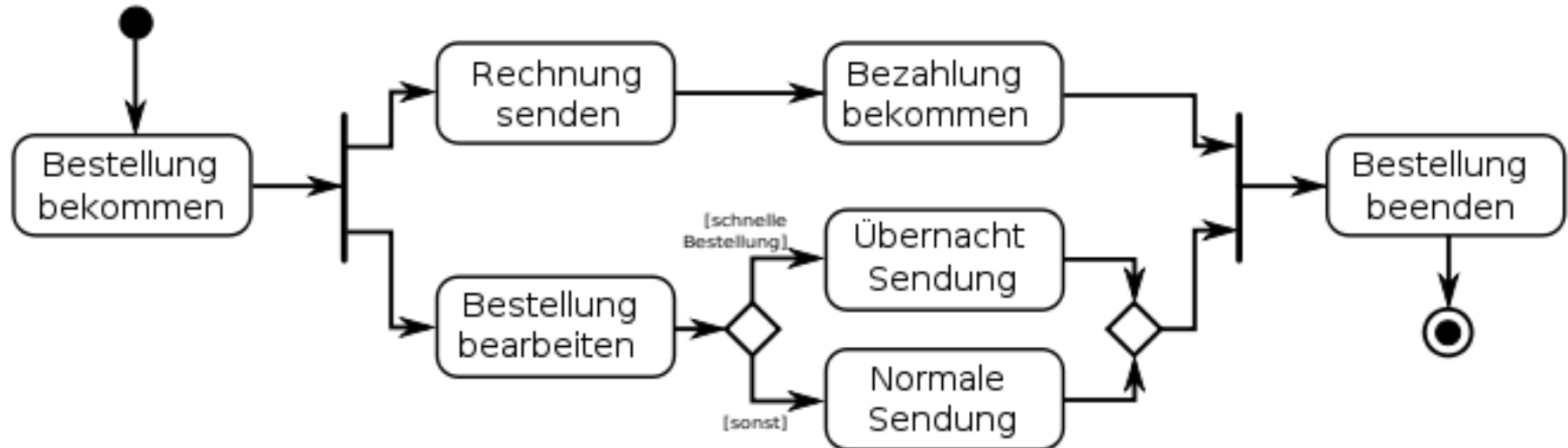
Use Case Diagramm: Notation



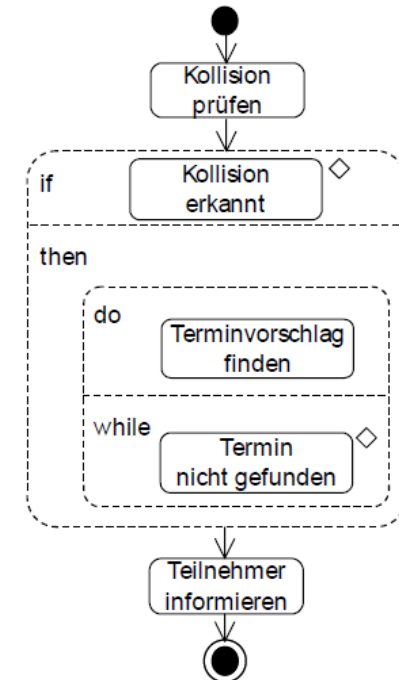
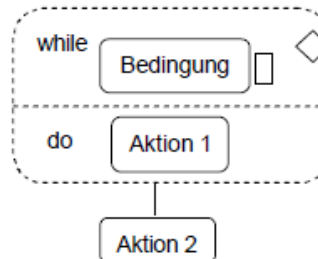
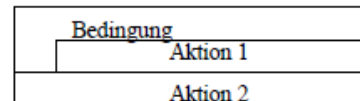
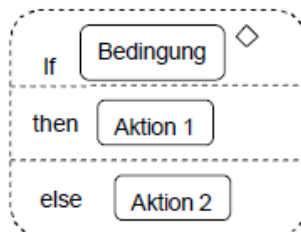
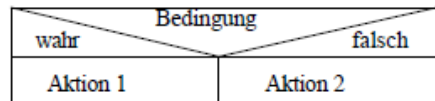


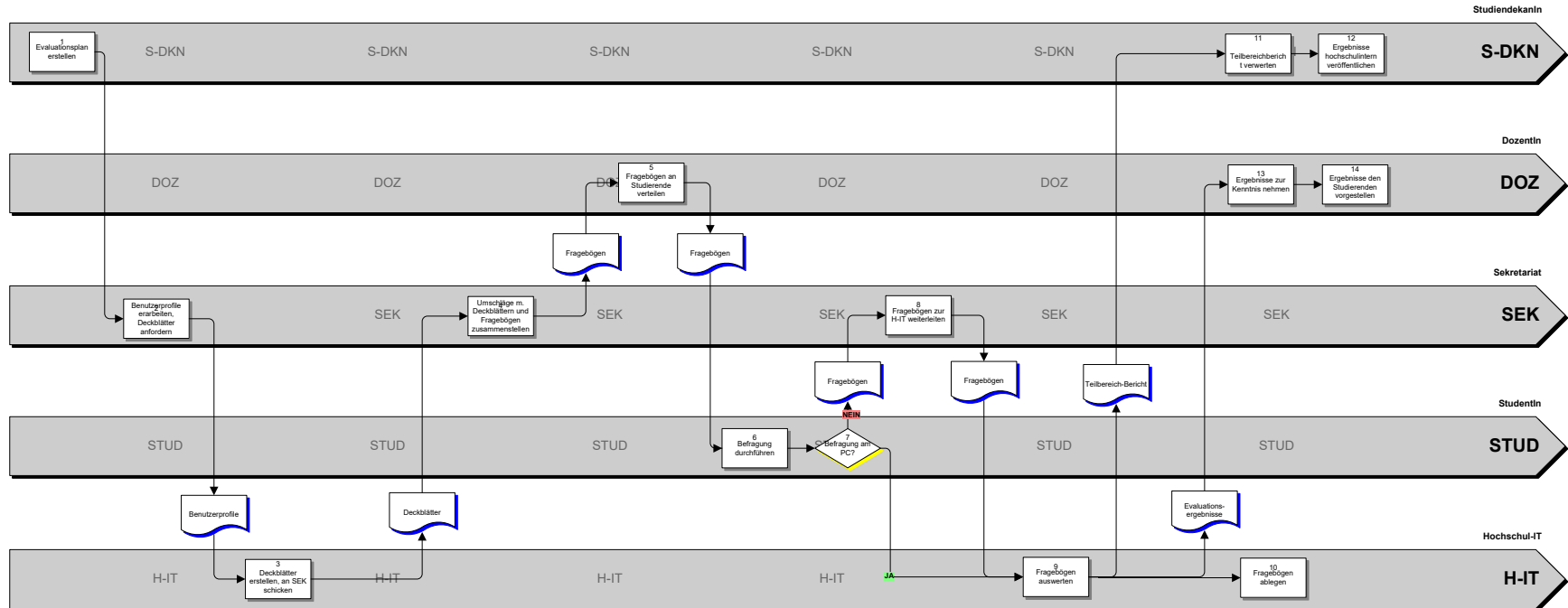
- Es gibt die sog. „**Flussnotation**“ und „**Knotennotation**“
- Flussnotation ist das UML-Äquivalent zum **Flussdiagramm**
- Knotennotation ist das UML-Äquivalent zum **Struktogramm**.

Beispiel:



- Zur Darstellung von Kontrollstrukturen und Algorithmen, ähnlich wie Struktogramme
- Unterstützt strukturierte Programmierung besser als Flussnotation
- Wird eher selten angewendet





Als Sonderform des Aktivitätendiagramms ist **BPMN** eine UML-Erweiterung für Geschäftsprozesse:

http://de.wikipedia.org/wiki/Business_Process_Modeling_Notation

Beschreiben **zeitliche** Abläufe (Aufrufsequenzen)
zwischen Objekten

Zwei semantisch äquivalente Darstellungen:
(Andere Darstellung, aber identischer Informationsgehalt)

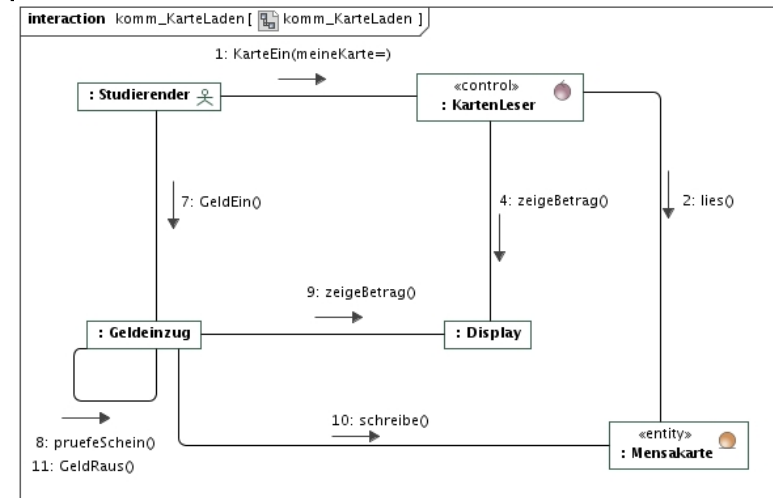
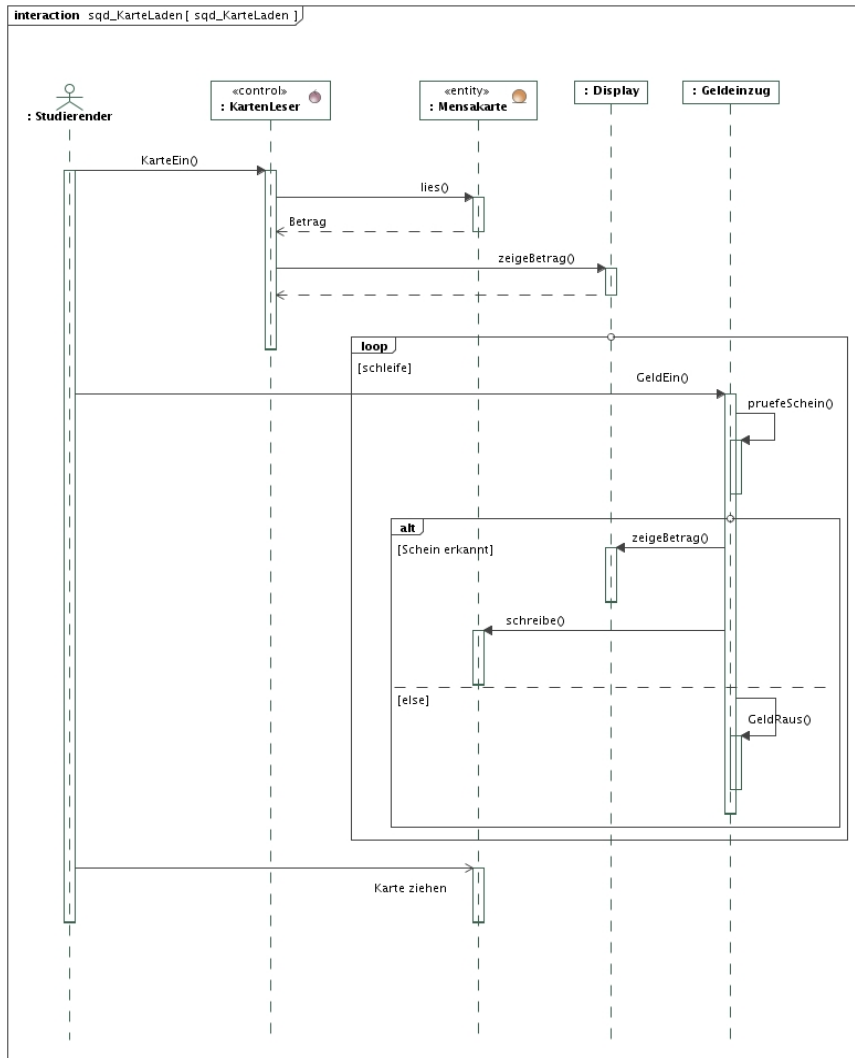
Sequenzdiagramm

- Verwendung bei wenigen Klassen
- Zeitablauf klar ersichtlich
- Basiert auf den Message Sequence Charts (MSCs) der ITU-T ([Z.120](#))

Kommunikationsdiagramm

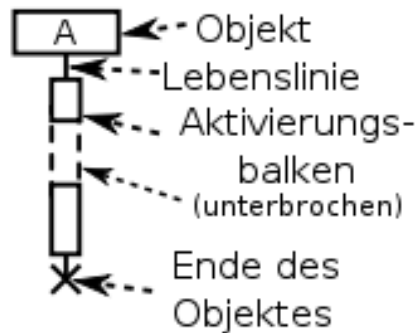
- Verwendung bei wenigen Nachrichten
- Zeitablauf weniger klar ersichtlich

Kommunikations- vs. Sequenzdiagramm



Beide Diagramme modellieren denselben Sachverhalt !

Sequenzdiagramm: Notation



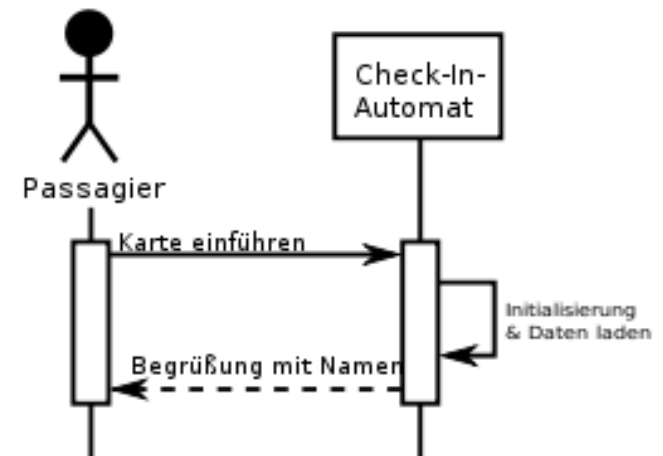
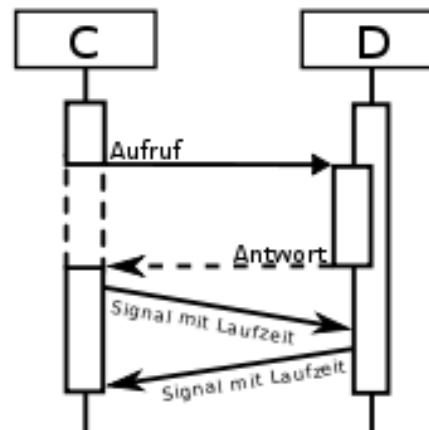
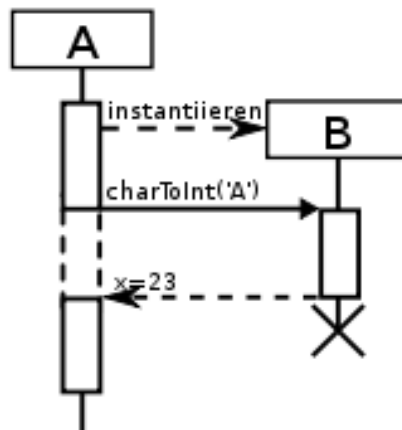
Beschreibung:

Ein synchroner Aufruf unterbricht den Aktivierungsbalken solange, bis eine synchrone Antwort eintrifft. Eine asynchrone Nachricht verändert nichts am Aktivierungsbalken.

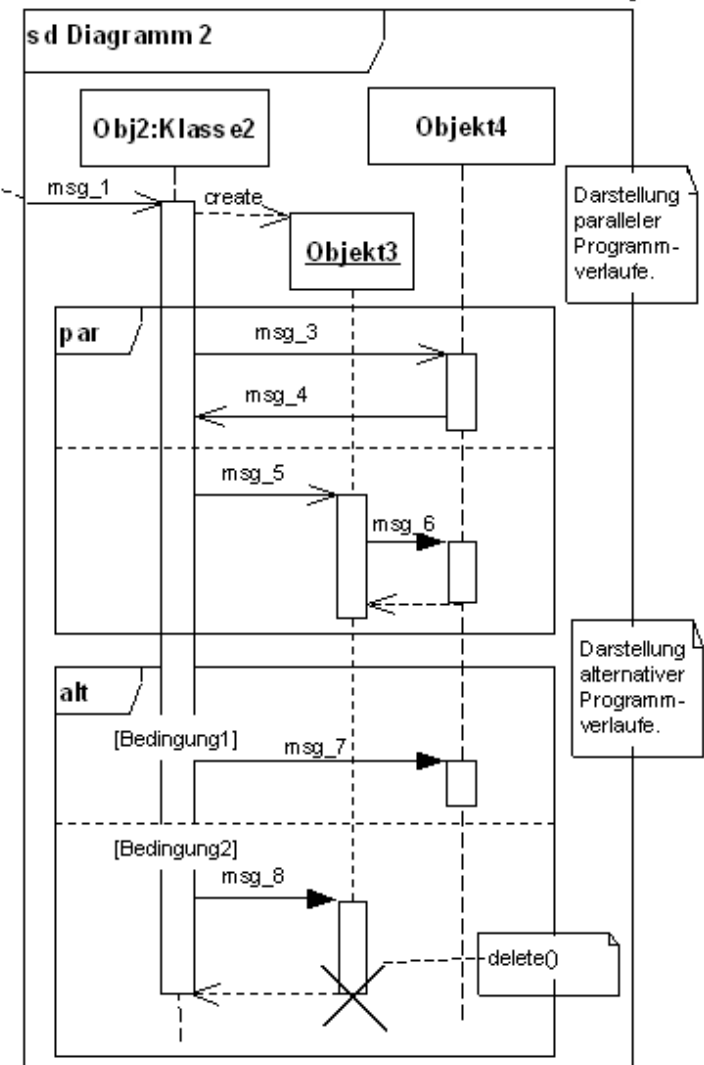
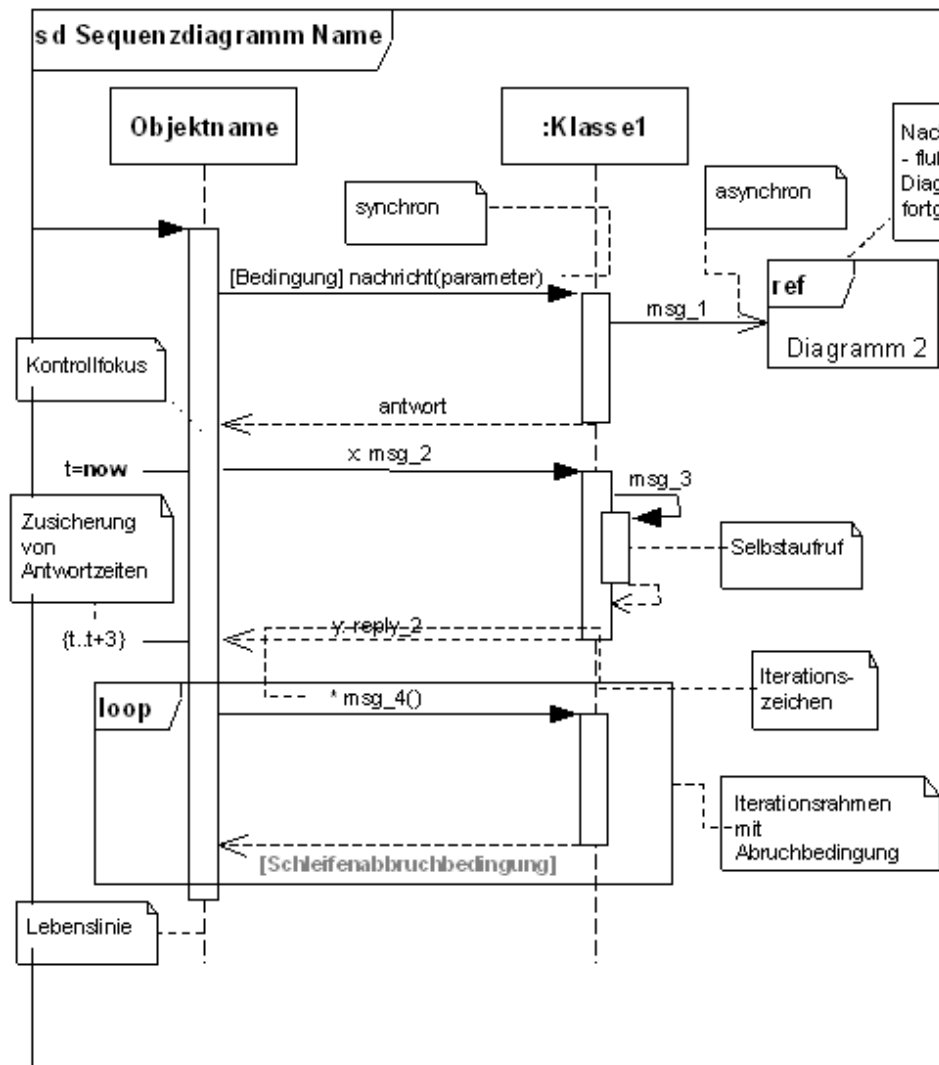
Notationen:

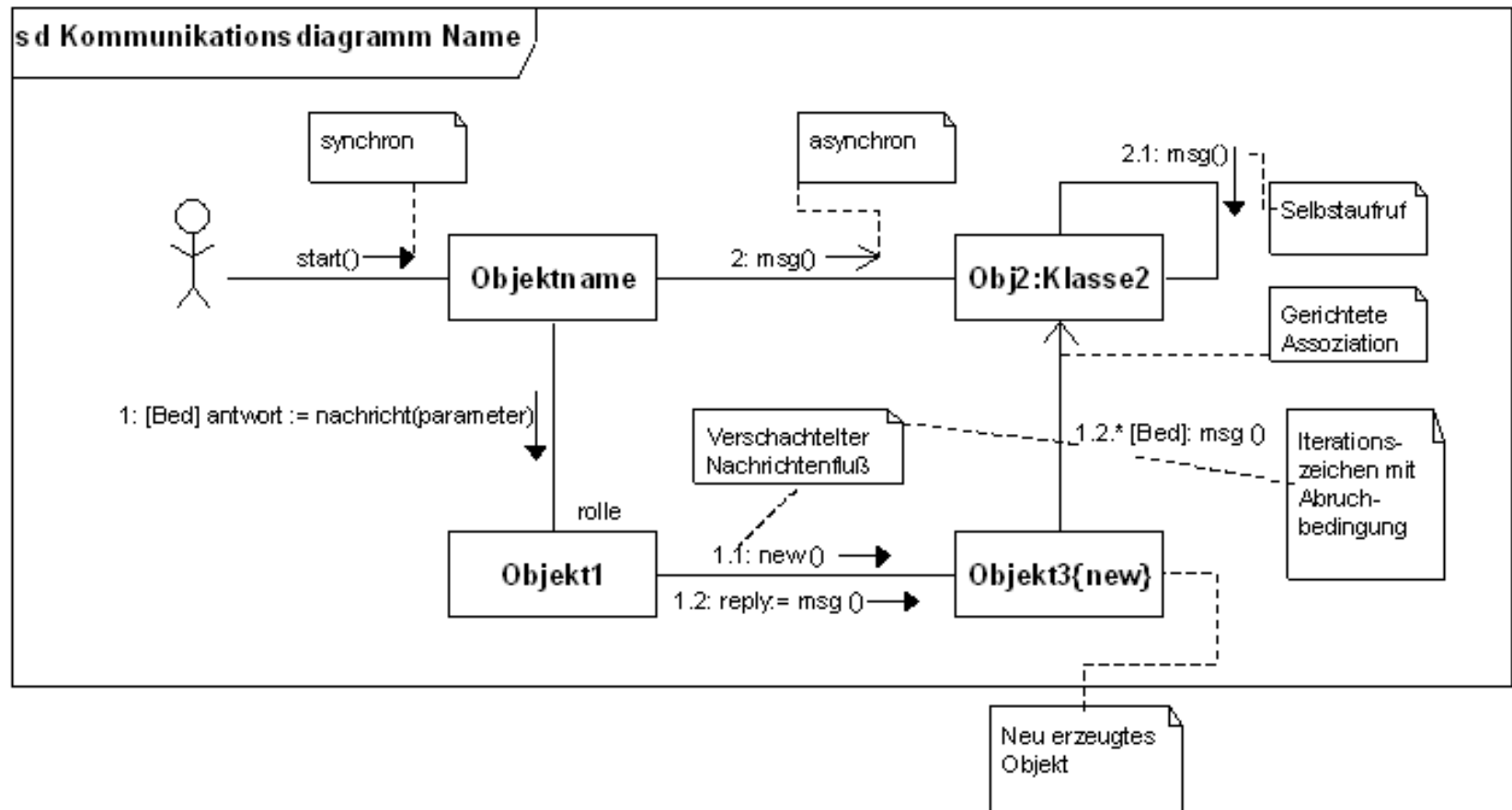
Nachricht ::= Aufruf | Antwort | Signal

→ Aufruf (synchrone Nachricht)
←-- Antwort (synchrone Nachricht)
→ Signal (asynchrone Nachricht)
← (schräg bei relevanter Laufzeit)



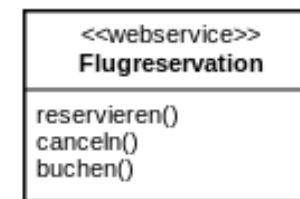
Sequenzdiagramm: Notation



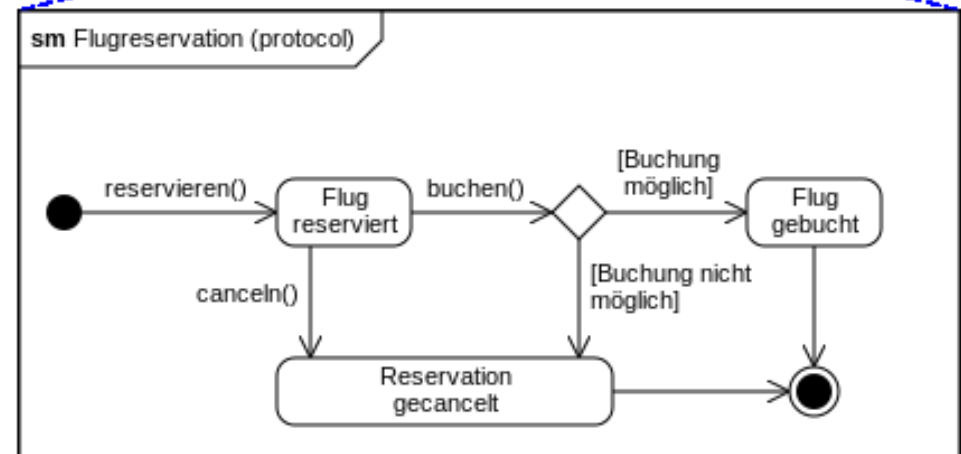
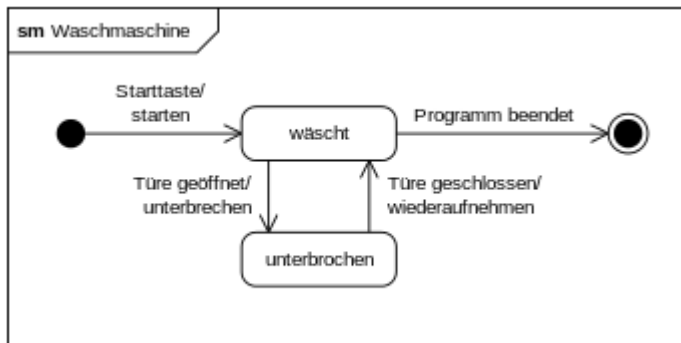


Ein Zustandsdiagramm zeigt die zur Laufzeit erlaubten Zustände eines Zustandsautomaten (z. B. eines Objektes oder Systems) an und gibt Ereignisse an, die seine Zustandsübergänge auslösen. Damit beschreibt ein Zustandsdiagramm eine hypothetische Maschine (endlicher Automat).

Protokollzustandsautomat

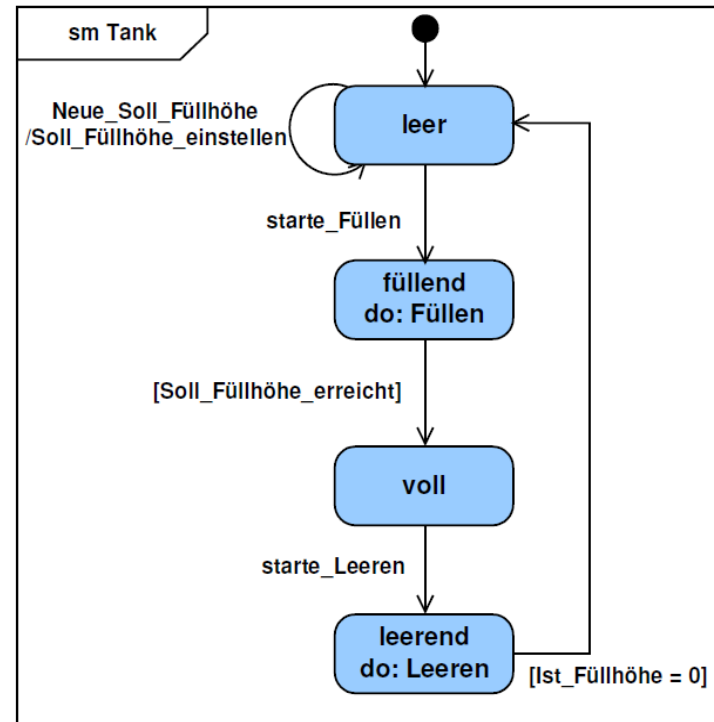
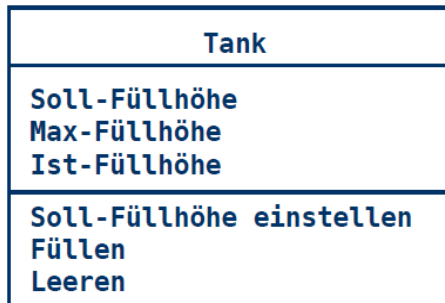


Verhaltenszustandsautomat

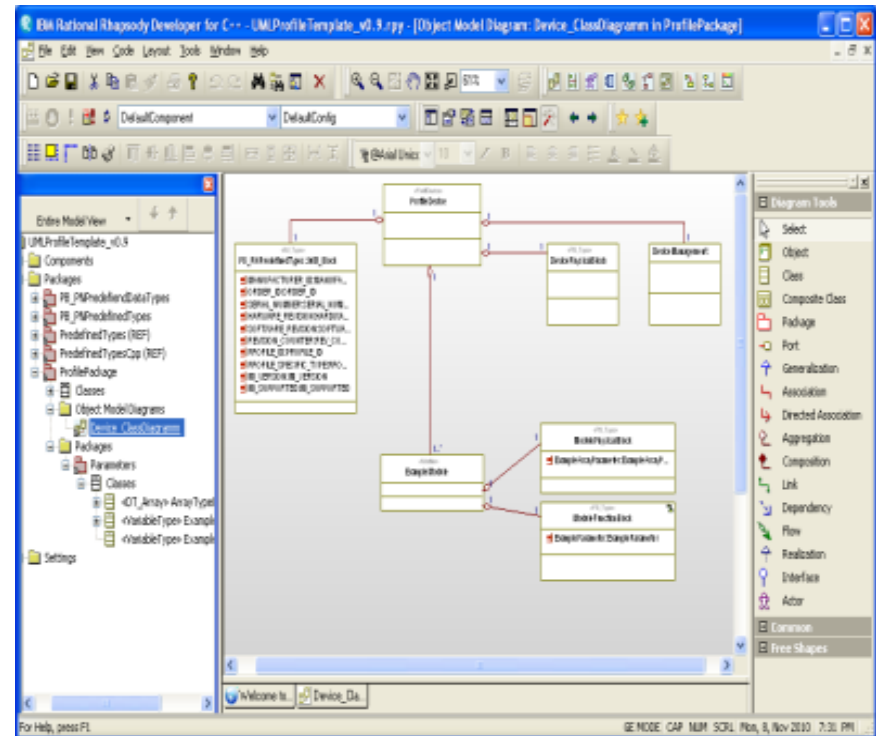


→ Wird in separater Vorlesung vertieft

Der Protokollzustandsautomat ist die typische Anwendung im Lösungsmodell zur Ergänzung von Klassendiagrammen. Das heisst, der Zustandsautomat beschreibt das dynamische Verhalten des durch die Klasse beschriebenen Objekts.



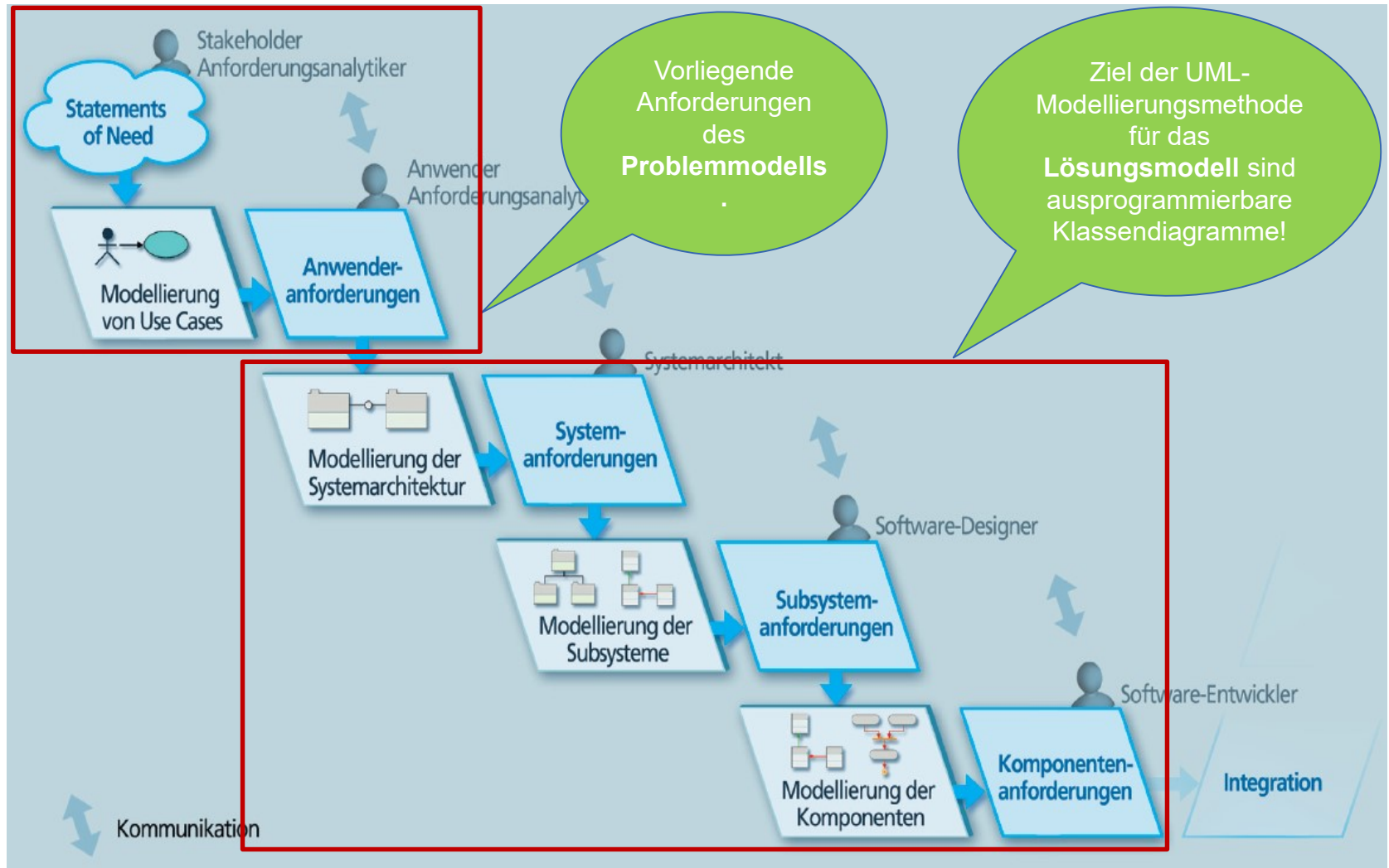
- <https://www.sparxsystems.de>
- <http://www.softwareideas.net/>
- <https://www.visual-paradigm.com/>
- ...und viele mehr,
sind meist ähnlich aufgebaut.



Bei der Anwendung beachten:

- Ziel einer UML-Modellierung sollte es sein, am Ende zu ausprogrammierbaren Klassendiagrammen zu gelangen, idealerweise per (teilweiser) Codegenerierung.
- Verwendung als rein informelles Malprogramm vermeiden – stattdessen möglichst formell korrekte Diagramme erstellen!
- Diagramme sinnvoll aufteilen und zu große Diagramme vermeiden
- UML-Diagramme hierarchisch korrekt strukturieren!
- Von Anfang an Namensgebung der Diagramme, Attribute, Methoden etc. in einer sinnvollen Nomenklatur
- Attribute, Methoden und Schnittstellen ordentlich klassifizieren und kommentieren
- ...

Systemanalyse und -design mit der UML



Am Besten hierarchisches Vorgehen (Top-Down):

- 1. Identifizierung** der grundsätzlichen Systemkomponenten (HW, SW und logische Einheiten) **aus den vorliegenden Anforderungen**
- 2. Verfeinerung** der Anforderungen bis der notwendige Detaillierungsgrad erreicht ist
- 3. Verteilung** der vorliegenden Anforderungen auf die Systemkomponenten
- 4. Definition** der notwendigen Funktionalitäten in den Systemkomponenten, bis ausprogrammierbare Klassendiagramme vorliegen.

Modellierungsmethodik (Vorgehen) ist abhängig vom Systemtyp und den gegebenen Anforderungen!

Folgende Systemarten werden im Folgenden betrachtet:

Verteiltes (eingebettetes) System:

→ **Kommunikationsorientiert**

Anwendungssystem:

→ **Bedienerorientiert (Workflow)**

Datenbanksystem:

→ **Datenhaltungsorientiert**

Beispiel: DHCP

→ RFC 2131

Anwendung: Automatische Netzkonfiguration

Gegeben:

Client-Server-Protokoll

DHCP-Paketformat

Sequenzdiagramme

Zustandsdiagramm für Client

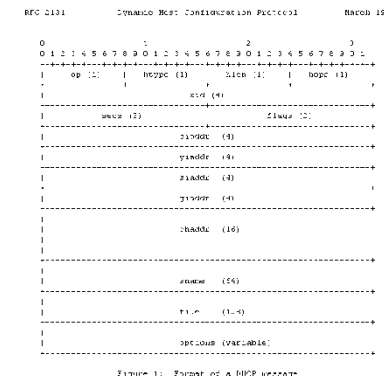
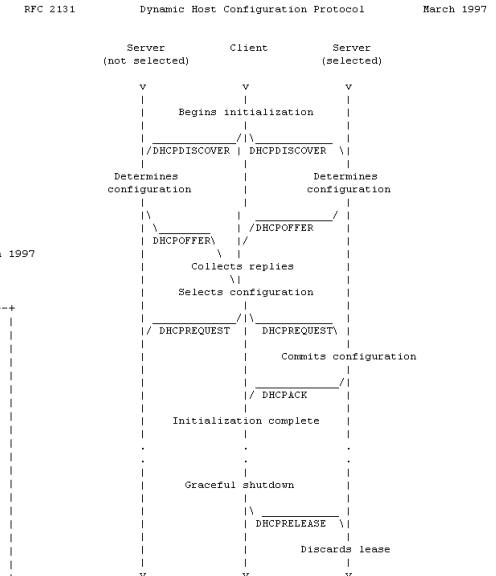
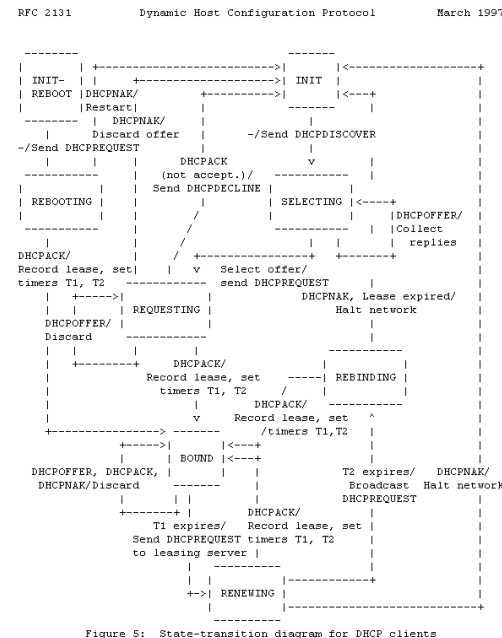
Gesucht:

Benutzerschnittstellen

(Konfigurationsdaten, Benutzerkonzept,...)

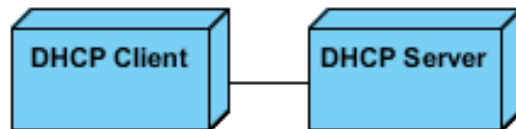
Konzept DHCP-Server

Protokollimplementierung

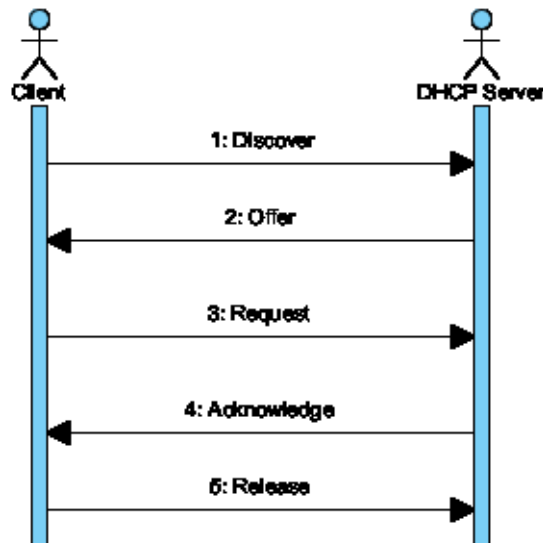


Protokollimplementierung

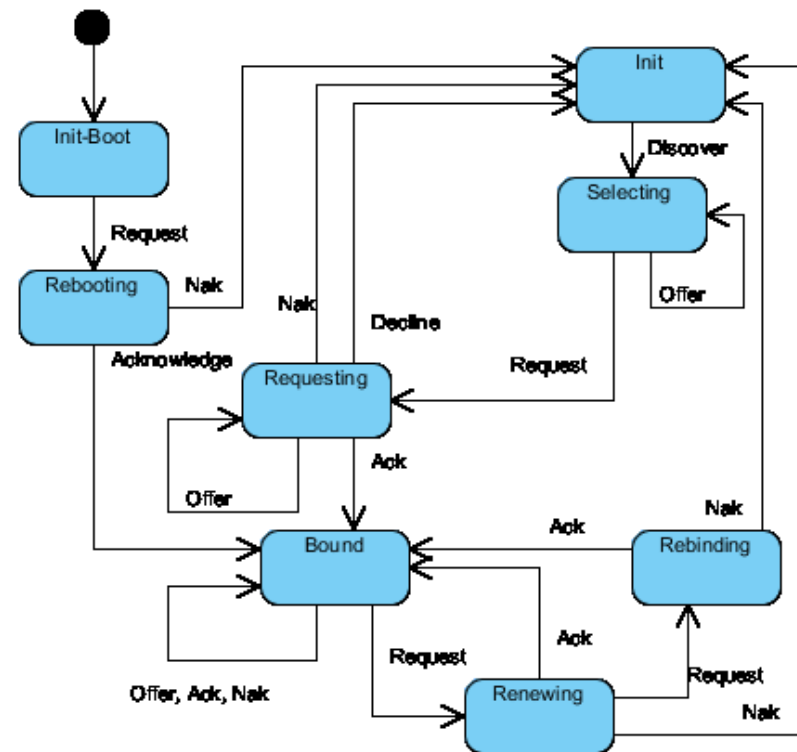
1) Verteilungsdiagramm



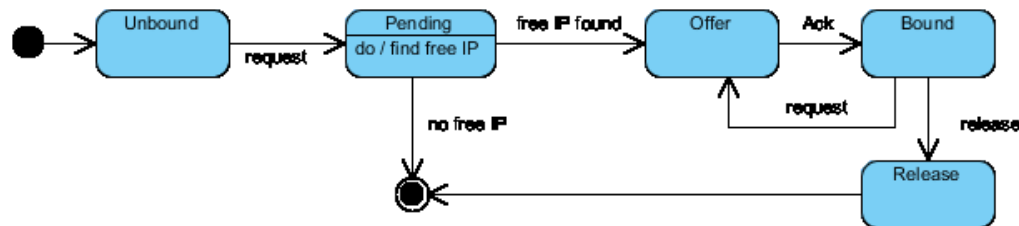
2) Sequenzdiagramme



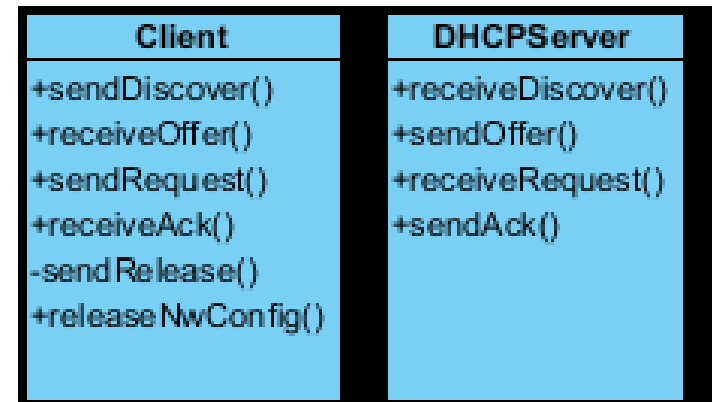
3) Zustandsdiagramm (Client)



4) Zustandsdiagramm (Server)



5) Klassendiagramme



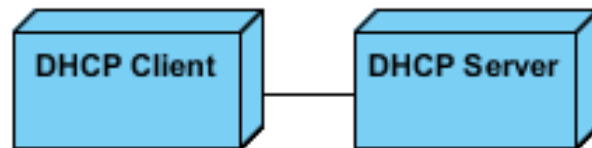
6) Implementierung

```
public class DHCP Server {  
  
    public void receiveDiscover() {  
        // TODO  
    }  
  
    public void sendOffer() {  
        // TODO  
    }  
  
    public void receiveRequest() {  
        // TODO  
    }  
  
    public void sendAck() {  
        // TODO  
    }  
  
}
```

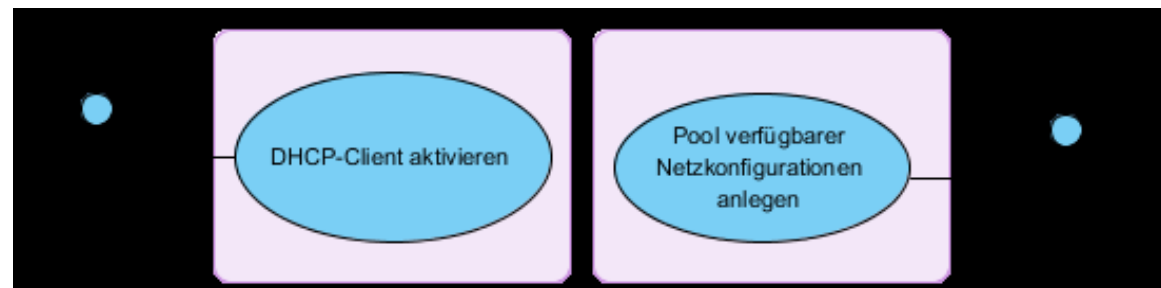
Ergebnis:
Netzwerkschnittstelle
DHCP Client & Server

Server- und Benutzerschnittstellenkonzept

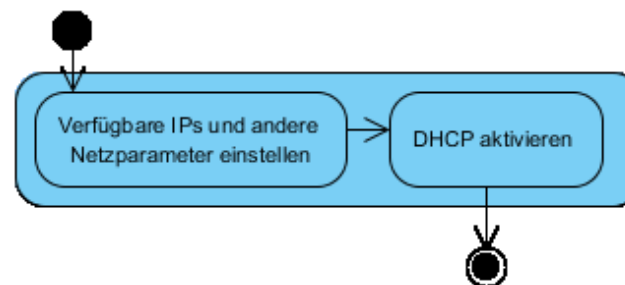
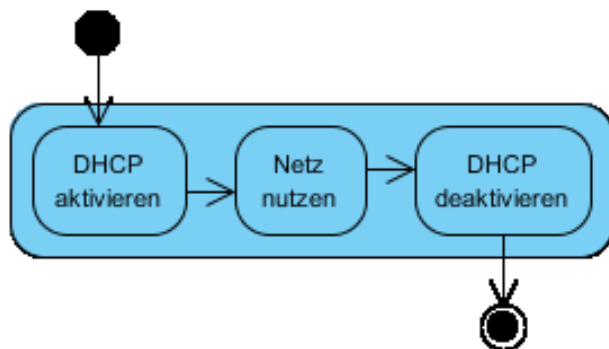
1) Verteilungsdiagramm



2) Use-Case Diagramm

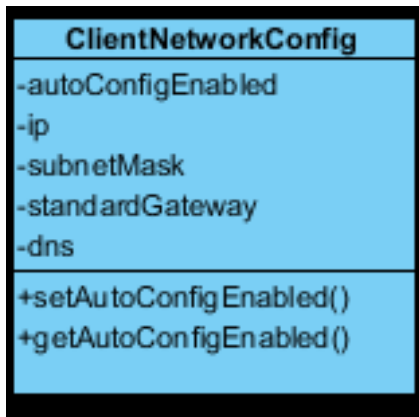


3) Aktivitätsdiagramm

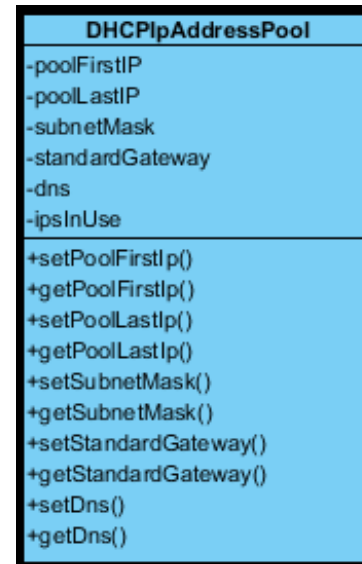


4) Klassendiagramm

Client:



Server:



5) Dialogskizze (kein UML-Diagramm)

☒ IP-Adresse automatisch beziehen

☐ Folgende IP-Adresse verwenden:

IP-Adresse: . .

Subnetzmaske: . .

Standardgateway: . .

Bevorzugter DNS-Server: . .

Alternativer DNS-Server: . .

Der DHCP-Server vergibt IP-Adressen

von

bis

Subnetzmaske

Standard Gateway

DNS

192	168	25	50
192	168	25	60
255	255	255	0

Systemmodellierung für verteilte (oft eingebettete) Systeme:

1. Systemarchitektur modellieren

- Verteilungsdiagramm zeichnen (HW-Komponenten)
- Logische Kommunikationspartner ermitteln und als (SW-)Komponenten im Verteilungsdiagramm platzieren
- Erstellen eines Klassendiagramms für jede SW-Komponente

2. Systemdynamik modellieren

- Ermitteln aller Aktoren und Uses Cases aus Benutzersicht und Sicht der Kommunikationspartner
- Jeden Use Case mit Szenarios (Aktivitätendiagramme, Sequenzdiagramme) beschreiben
- Für die Interaktion der **Kommunikationspartner** Sequenzdiagramme ableiten
- Ermitteln aller Events, Inputs, Outputs, etc. aus Black-Box-Sicht der Kommunikationspartner
- Für die Klassendiagramme anhand der Sequenzdiagramme Zustandsdiagramme erstellen

3. Systemobjekte und -daten identifizieren

- In den Use-Case-Szenarios die Systemobjekte und -daten identifizieren
- Für die identifizierten Systemobjekte und –daten Klassendiagramme erstellen

4. SW-Systemarchitektur modellieren

- Klassendiagramme evtl. verfeinern und in Komponentendiagrammen sinnvoll gruppieren
- Schnittstellen entwerfen

5. Implementierung

- Klassen ausprogrammieren

Beispiel: Webbasiertes Tool zur Benutzerverwaltung



Gegeben:

Benutzerdaten



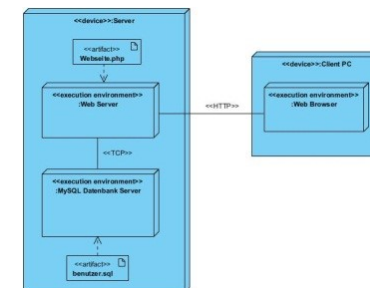
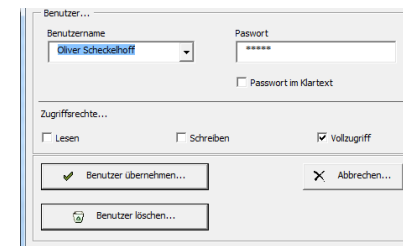
Gesucht:

Dialoge

Workflow

Systementwurf/-implementierung

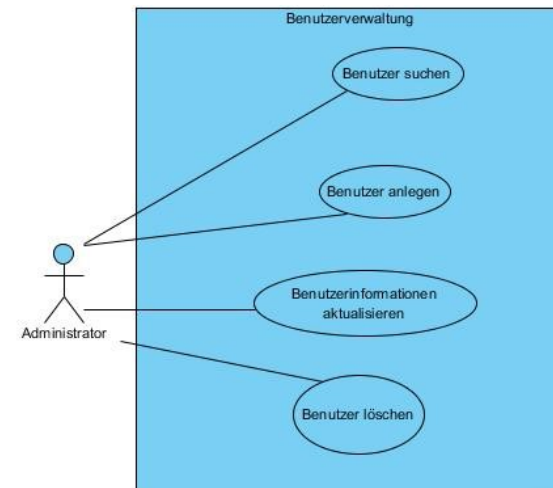
- Systemarchitektur
- Systemschnittstellen



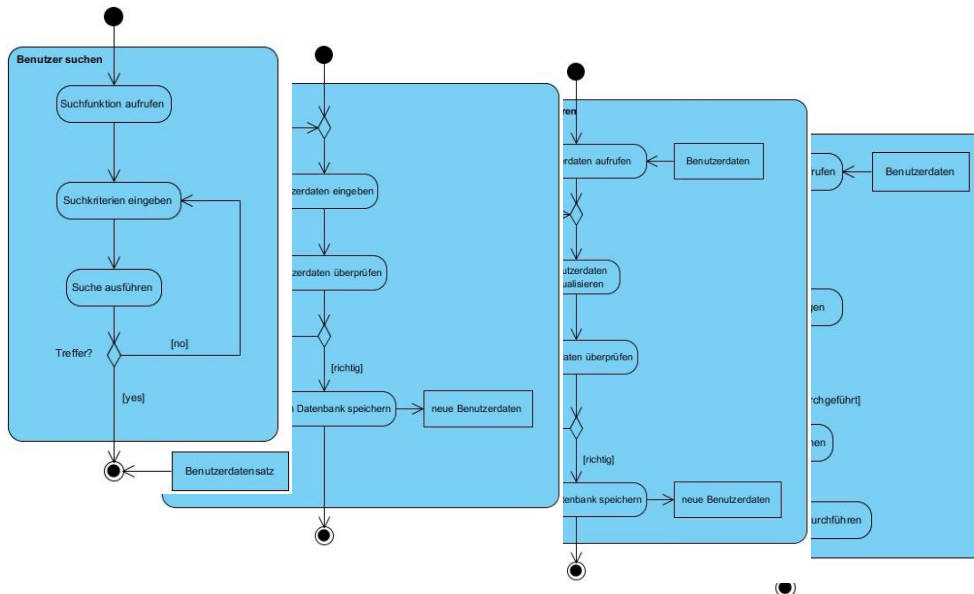
1) Klassendiagramm



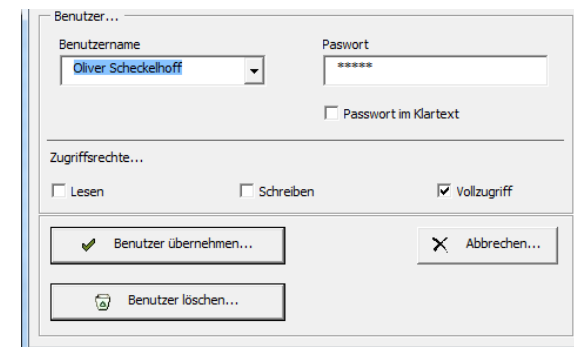
2) Use-Case Diagramm



3) Aktivitätsdiagramm(e)



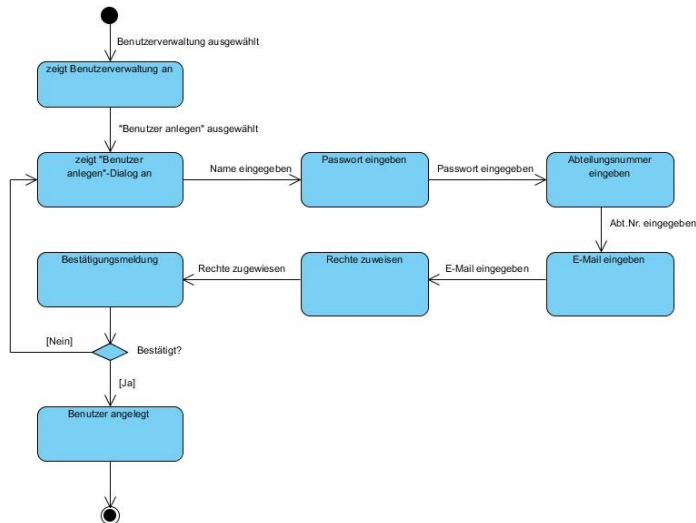
4) GUI Entwurf



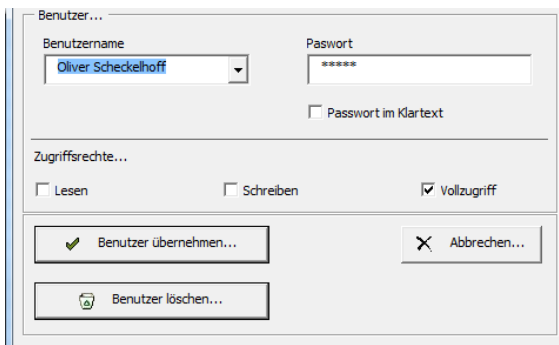
The screenshot shows a dialog box titled **Benutzer...** with the following elements:

- Benutzername:** A dropdown menu showing 'Oliver Scheckelhoff'.
- Paswort:** A text field with masked characters '*****'.
- ☐ **Passwort im Klartext**
- Zugriffsrechte...** section with:
 - ☐ **Lesen**
 - ☐ **Schreiben**
 - ☒ **Vollzugriff**
- Buttons: **Benutzer übernehmen...** (with a green checkmark icon) and **Abbrechen...** (with a red X icon).
- Buttons: **Benutzer löschen...** (with a trash can icon).

5) Zustandsdiagramm



7) GUI Entwurf



Benutzer...

Benutzername:

Passwort:

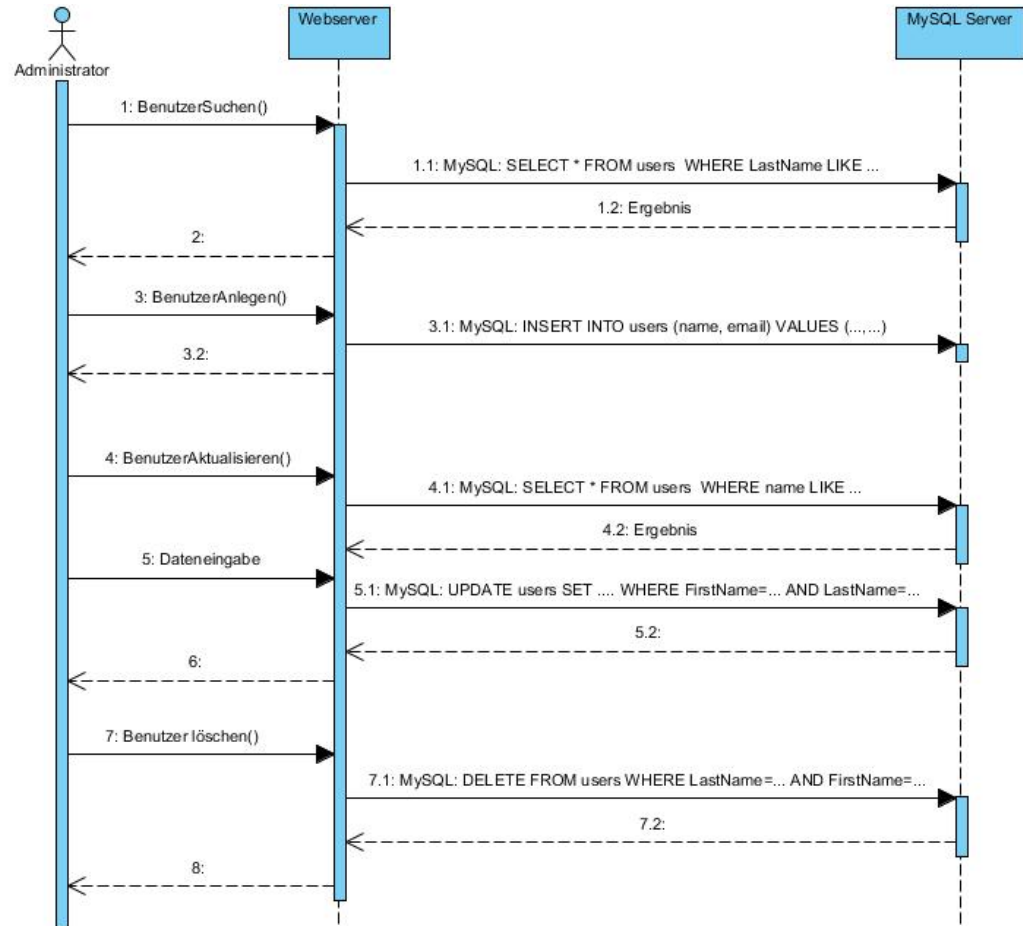
☐ Passwort im Klartext

Zugriffsrechte...

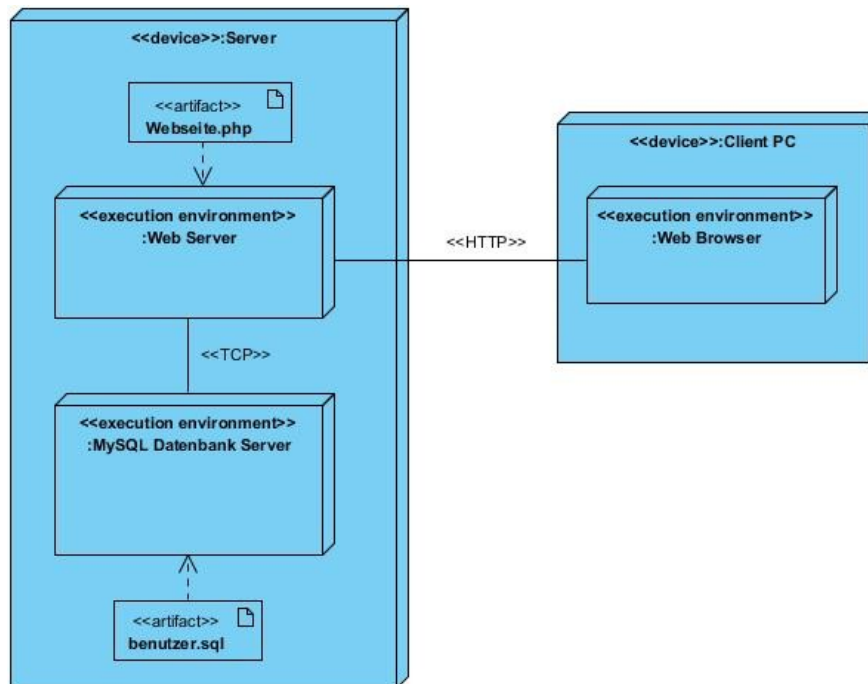
☐ Lesen ☐ Schreiben ☒ Vollzugriff

6) Sequenzdiagramm

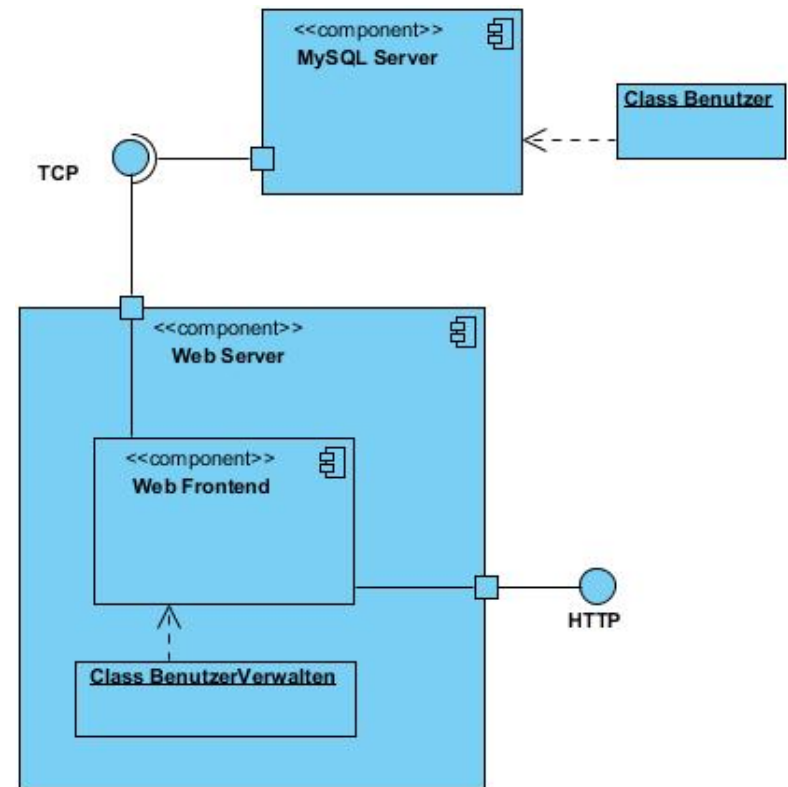
Umsetzung in MySQL Anweisungen



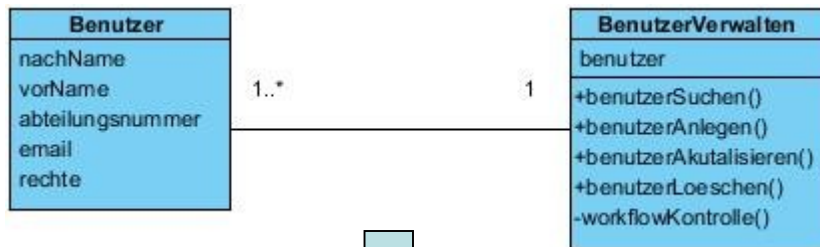
7) Verteilungsdiagramm



8) Komponentendiagramm



9) Klassen ausprogrammieren



```
public class BenutzerVerwalten {

    public void benutzerSuchen() {
        // TODO
    }

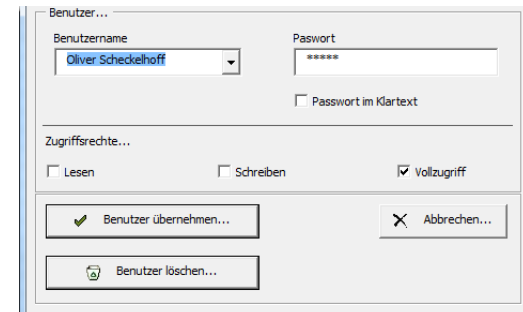
    public void benutzerAnlegen() {
        // TODO
    }

    public void benutzerAktualisieren() {
        // TODO
    }

    public void benutzerLoeschen() {
        // TODO
    }

    public void workflowKontrolle() {
        // TODO
    }

}
```



Systemmodellierung für Datenhaltungsbasierte Systeme:

1. Systemdaten identifizieren

- Anhand der Anforderungsbeschreibungen Systemdaten identifizieren
- Die identifizierten Daten sinnvoll gruppieren und Klassendiagramme erstellen
- Methoden definieren, die auf den Daten operieren sollen

2. Anwendungsfälle ermitteln

- Ermitteln der Aktoren und Uses Cases für die Systemdaten aus Benutzersicht
- Jeden Use Case mit Szenarios (Aktivitätendiagramme, Sequenzdiagramme) beschreiben

3. Systemarchitektur modellieren

- Anhand der Anforderungsbeschreibungen und Szenarien die Systemobjekte identifizieren
- Klassendiagramme zu Systemobjekten definieren
- Beziehungen zu Systemdaten festlegen
- Klassendiagramme in Komponentendiagrammen sinnvoll gruppieren
- Schnittstellen entwerfen
- Komponenten im Verteilungsdiagramm sinnvoll gruppieren

4. Systemdynamik modellieren

- Anhand der Use-Case-Szenarios für die Interaktion der Aktoren und Objekte Sequenzdiagramme ableiten
- Für die Klassendiagramme anhand der Sequenzdiagramme Zustandsdiagramme erstellen

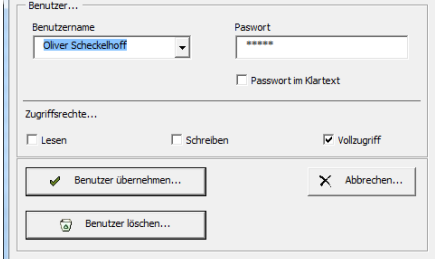
5. Implementierung

- Klassen ausprogrammieren

Beispiel: Benutzerverwaltung

Gegeben:

Benutzerschnittstelle (Dialog-Skizzen)



Benutzer...

Benutzername: Passwort:

☐ Passwort im Klartext

Zugriffsrechte...

☐ Lesen ☐ Schreiben ☒ Vollzugriff

☒ Benutzer übernehmen...

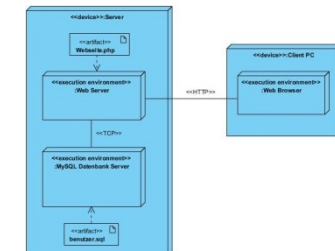
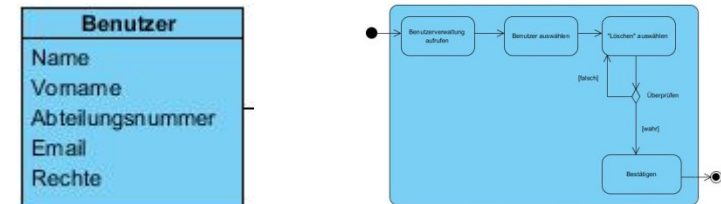
☒ Benutzer löschen...

Gesucht:

Benutzerdaten

Konzept Benutzerverwaltung (Workflow)

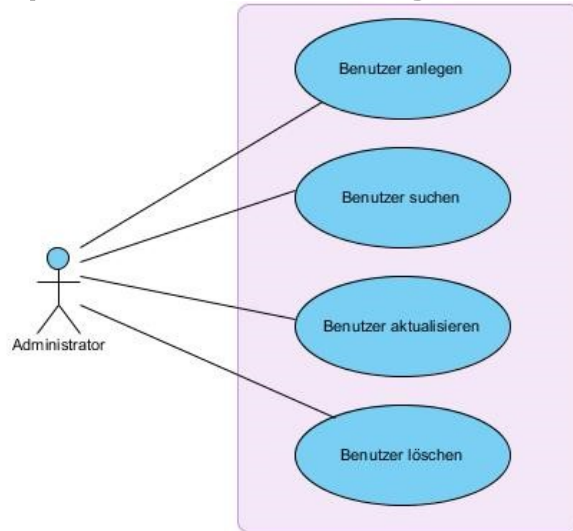
Implementierung der Funktionen



Verhaltensorientierte Methode (I)

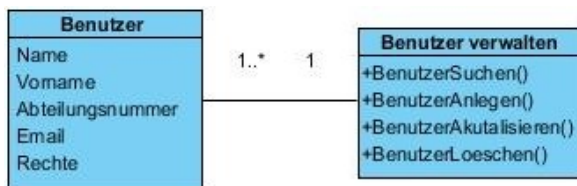
Konzept Benutzerverwaltung (Workflow)

1) Use-Case Diagramm



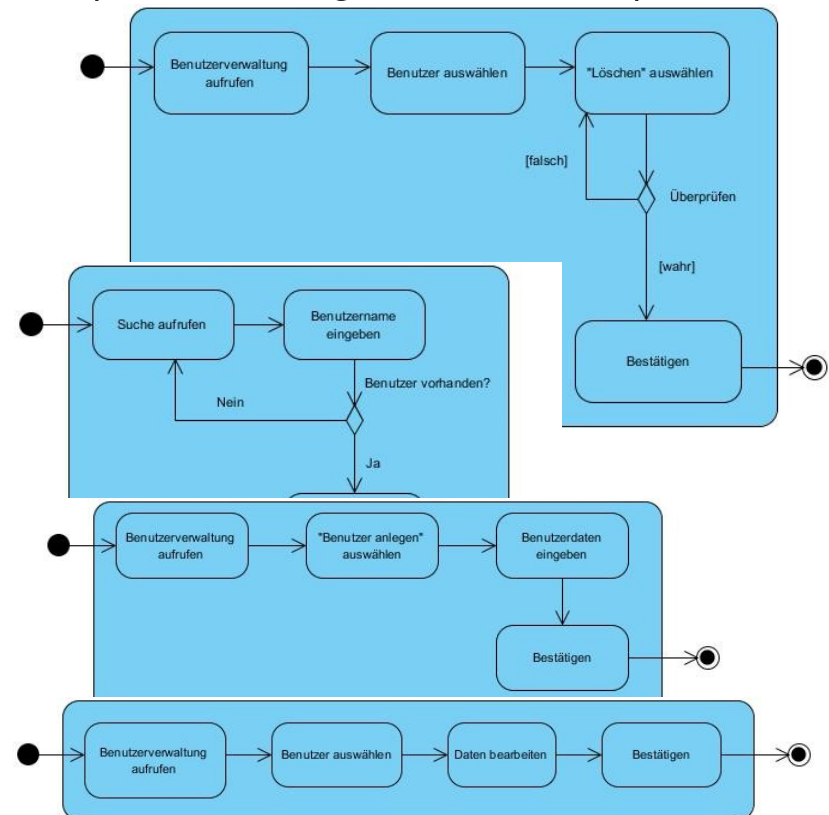
3) Klassendiagramm

(aus Benutzerdaten)



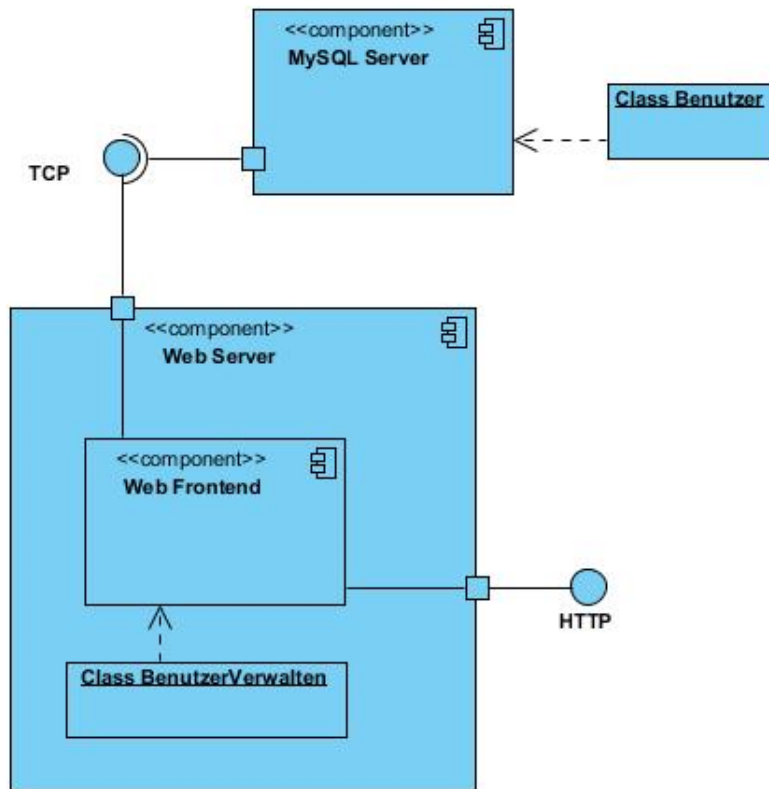
2) Aktivitätsdiagramme

(Umsetzung in Workflow)



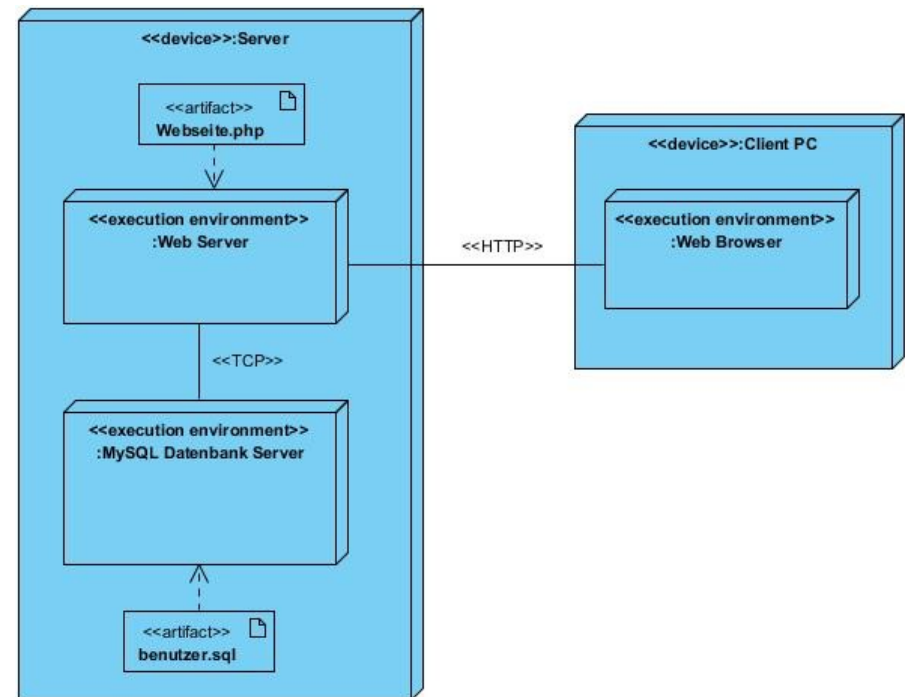
4) Komponentendiagramm

(Schnittstellen)

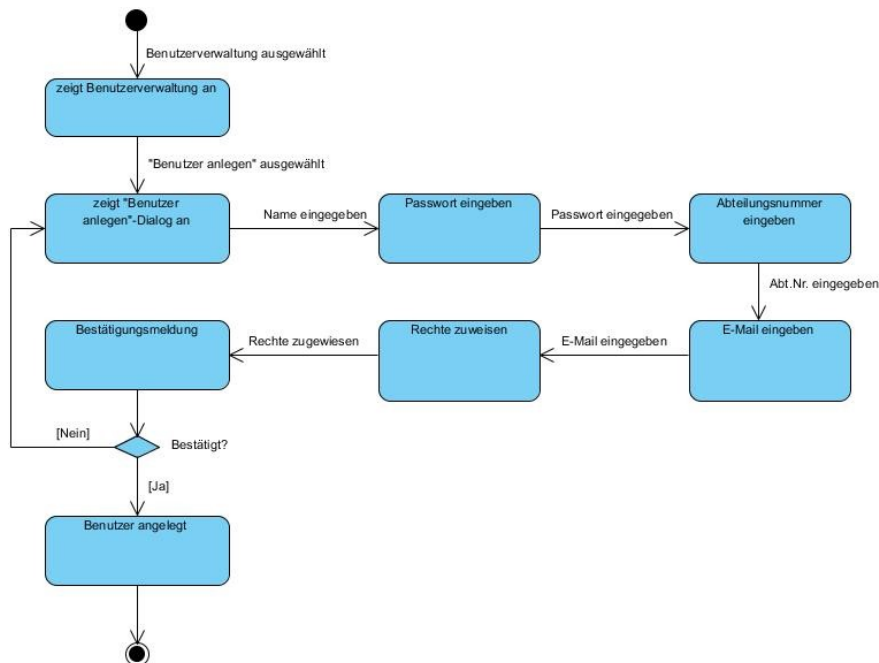


5) Verteilungsdiagramm

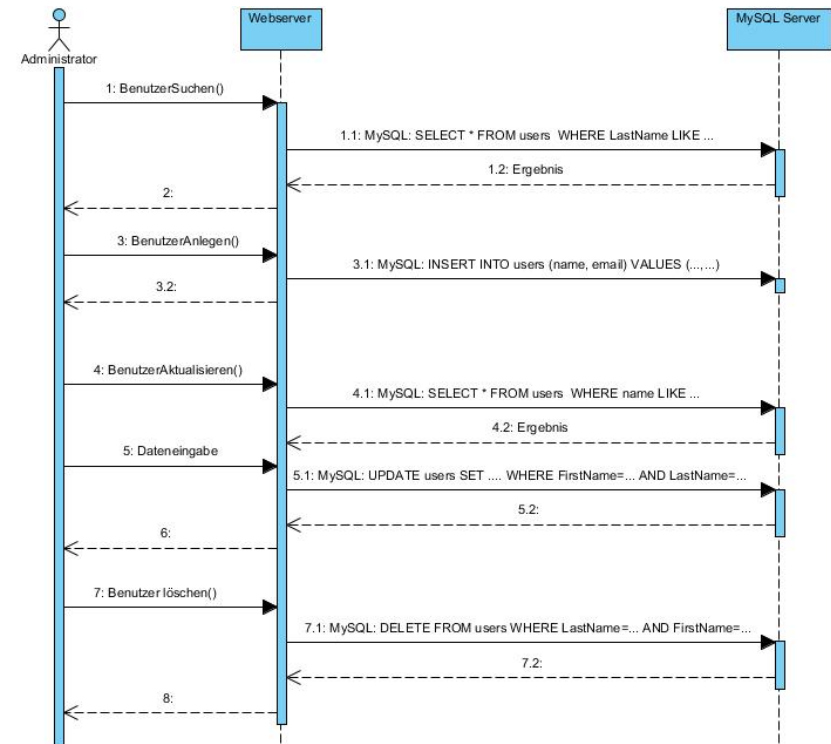
(Zuordnung zur Hardware)



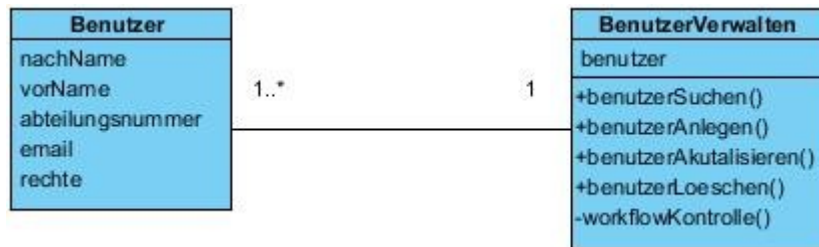
6) Zustandsdiagramm (Workflowkontrolle)



7) Sequenzdiagramm (Umsetzung in MySQL Anweisungen)



8) Klassendiagramm



```
public class BenutzerVerwalten {

    public void benutzerSuchen() {
        // TODO
    }

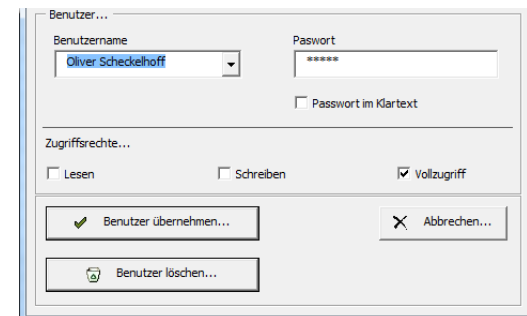
    public void benutzerAnlegen() {
        // TODO
    }

    public void benutzerAktualisieren() {
        // TODO
    }

    public void benutzerLoeschen() {
        // TODO
    }

    public void workflowKontrolle() {
        // TODO
    }

}
```



Systemmodellierung für Workflow-basierte Systeme:

1. Anwendungsfälle ermitteln

- Ermitteln aller Aktoren und Uses Cases aus Benutzersicht
- Jeden Use Case mit Szenarios (Aktivitätendiagramme, Sequenzdiagramme) beschreiben

2. Systemobjekte und -daten identifizieren

- In den Use-Case-Szenarios die Systemobjekte und -daten identifizieren
- Für die identifizierten Systemobjekte und -daten Klassendiagramme erstellen

3. Systemarchitektur modellieren

- Klassendiagramme in Komponentendiagrammen sinnvoll gruppieren
- Schnittstellen entwerfen
- Komponenten im Verteilungsdiagramm sinnvoll gruppieren

4. Systemdynamik modellieren

- Anhand der Use-Case-Szenarios für die Interaktion der Aktoren und Objekte Sequenzdiagramme ableiten
- Für die Klassendiagramme anhand der Sequenzdiagramme Zustandsdiagramme erstellen

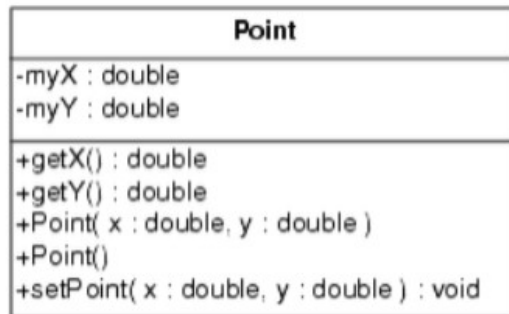
5. Implementierung

- Klassen ausprogrammieren

Beschreiben Sie die folgenden Szenarien durch jeweils ein Sequenz- oder ein Kommunikationsdiagramm.

- a) Ein Pärchen streitet sich. Sie macht ihm Vorwürfe, dass er sich zu wenig um den Haushalt kümmert. Er rechtfertigt sich, dass er den ganzen Tag arbeitet und gerne das Spiel schauen möchte. Optional kann er sie beschwichtigen indem er ihr zustimmt und Besserung verspricht.
- b) Eine Gruppe von fünf Leuten spielt „Stille Post“. Die Nachricht des ersten Spielers lautet „Katzenfutter“. Der letzte Spieler teilt allen Anderen mit, dass bei ihm „Katastrophe“ ankam.

Analysieren Sie die Zusammenhänge und Besonderheiten in Code und Diagramm:



```
/**
 * Point - a double x,y coordinate
 */
public class Point
{
    // Attributes

    private double myX;
    private double myY;

    // Constructors

    public Point(double x, double y)
    {
        myX = x; myY = y;
    }

    public Point()
    {
        myX = 0.; myY = 0.;
    }

    // Methods

    public double getX()
    {
        return myX;
    }

    public double getY()
    {
        return myY;
    }

    public void setPoint(double x, double y)
    {
        myX = x; myY = y;
    }
}
```


Erstellen Sie ein Klassendiagramm aus folgenden Java-Code:

```
public class Hund {  
    private String rasse;  
    private String fellfarbe;  
    public int anzahlBefehle; // Befehle, die der Hund kennt  
  
    public Hund(String rasse, String fellfarbe) {  
        this.rasse = rasse;  
        this.fellfarbe = fellfarbe;  
    }  
    public int befehleBerechnen(int alter, String hundeschule) {  
        String trainingsziel = "";  
        int faktorisiertesAlter = alter > 10 ? alter * 100 : alter * 50;  
        switch(faktorisiertesAlter) {  
            case 100:  
                trainingsziel = hundeschule + ", Gold-Diplom";  
                break;  
            default:  
                trainingsziel = hundeschule + ", Standard-Diplom";  
                break;  
        }  
        return faktorisiertesAlter;  
    }  
}
```

Erstellen Sie ein Sequenzdiagramm:

Gegeben ist das folgende Java-Programm. Die Operation **doMore()** wird von einem Objekt mit der Bezeichnung **einObjekt** aktiviert wird.

```
class ClassB { public void doSomething { ... }  
               public void work (int w) { ... }  
             }  
class ClassC { public void doSomethingElse() { ... }  
             }  
class ClassA {  
    private ClassB b;  
    private ClassC c;  
    public void doLess(int param) { b.work(param);}  
    public int calculateP(int param) { int p = 2 * param; return p;}  
    public void doMore(int data) { b = new ClassB();c = new ClassC();  
                                   for (int j = 1; j <= 5; j++) doLess (j);  
                                   int p = calculateP (data);  
                                   if (p <1){ b.doSomething();b.work(p);}  
                                   else c.doSomethingElse(); }  
}
```

Praxisprojekt

- Erstellen Sie für Ihre Systemmodellierung ein UML-Modell in Anwendung eines der Design Patterns
- Benutzen Sie dazu ein CASE-Tool