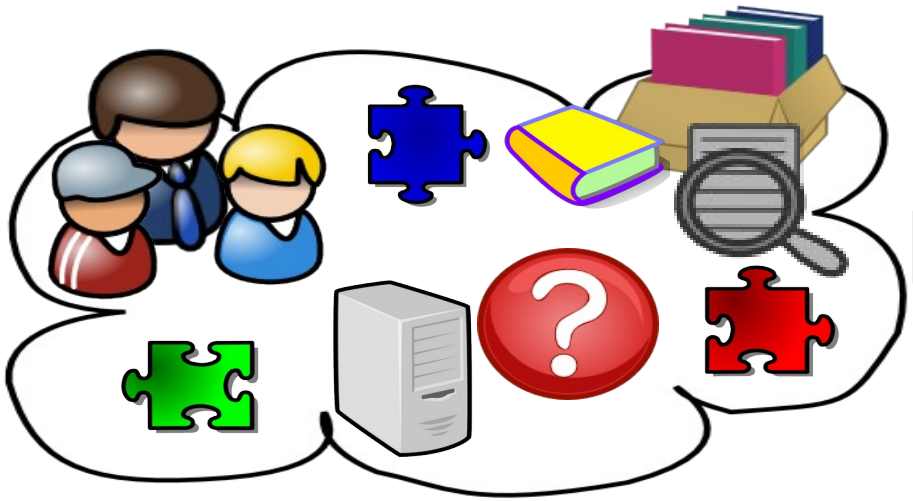
Anforderungen analysieren und dokumentieren

Ziele, Ideen, Anfragen, Wünsche
Vorschriften ...

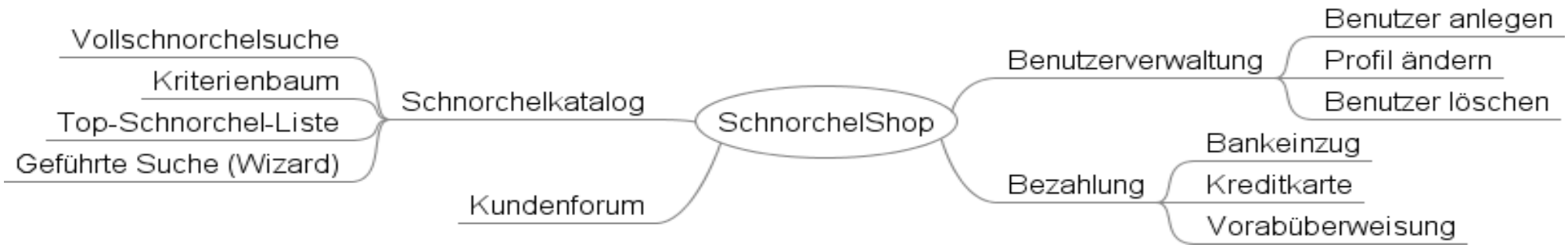


Analyse

(Anforderungs-)
Spezifikation

Hilfsmittel beim Sammeln von Anforderungen

Mindmaps



- Gruppierung zusammengehöriger Konzepte
- (schrittweise) Detaillierung / Verfeinerung

- Gut geeignet, um Gedanken und Ideen zu
  - sortieren
  - visualisieren
  - in der Gruppe zu entwickeln und diskutieren

- Isoliert betrachtet nicht immer selbsterklärend / eindeutig.

Anwendungsszenarien aufschreiben

- Szenario: Beschreibung einer **typischen Situationen**, in denen **Personen** mit einem **System** interagieren,
- einschließlich des **Zusammenhangs** (Kontexts).
- Kommunikationsmittel, auch von Nicht-Technikern verständlich und beurteilbar.
- Erzählstil, lieber **viele** kurze, auf eine Story **focussierte** Szenarien als eine/wenige überladene. Sollten „in Summe“ Themenbereich gut ausleuchten.

Szenario S017 – „Dinge tun, ohne zu wackeln“

Szenario S012 – „Sachen machen“

Szenario S009 – „Durchführungen durchführen“

Szenario S005 – „Weidemieren und folleppen“

Szenario S001 - „Studiausweis aktualisieren“

Der Student Jöndhard hat sich für das neue Semester zurückgemeldet und möchte seinen Studiausweis verlängern. Er geht zu einem Verlängerungsautomaten und steckt seinen Ausweis in den Kartenleser. Der Automat bietet ihm an, den Ausweis zu verlängern oder zu schreddern. Jöndhard wünscht ersteres. Der Automat prüft in maximal 15 Sekunden erfolgreich, dass Jöndhard seinen Semesterbeitrag gezahlt hat. Er aktualisiert die Angaben auf dem Ausweis, wirft diesen aus und zeigt den erfolgreichen Abschluss des Vorgangs an.

Szenarien analysieren

- Aus jedem Szenario kann nachfolgend eine Reihe unterschiedlicher Informationen herausanalysiert werden.

Akteur
(Rolle „Student“)

ggf. vom System zu prüfende **Voraussetzungen**

Szenario S001 - „Studiausweis aktualisieren“

*Der **Student** Jöndhard hat sich **für das neue Semester zurückgemeldet** und **möchte seinen Studiausweis verlängern**. Er geht zu einem Verlängerungsautomaten und steckt seinen Ausweis in den Kartenleser. Der Automat bietet ihm an, den Ausweis zu **verlängern** oder zu **schreddern**. Jöndhard **wählt** ersteres. Der Automat **prüft** in **maximal 15 Sekunden** erfolgreich, dass Jöndhard seinen **Semesterbeitrag gezahlt** hat. Er **aktualisiert die Angaben** auf dem Ausweis, **wirft diesen aus** und **zeigt den erfolgreichen Abschluss** des Vorgangs an.*

Ziel / Absicht / Kontext zum Verständnis

nichtfunktionale Anforderungen

(Teil-)**Funktionen** des Systems

Wie dokumentieren wir Anforderungen?

(... und für wen eigentlich?)

Wer nutzt „die Spec“ - und **wofür**?

Kunden:
Sind alle Anforderungen berücksichtigt?
Anforderungsänderungen

Softwarearchitekt / Entwickler:
Zu entwickelndes System verstehen

Tester:
Entwicklung von Tests

Betrieb / Wartung:
System und Beziehung zwischen seinen Bestandteilen verstehen;
„is it a bug or a feature?“

Manager:
Angebotserstellung
Planung des Entwicklungsprozesses

- Die **Anforderungsspezifikation** („Spec“) ist ein ganz zentrales Arbeitsdokument, es muß „**praktisch**“ sein. Dazu:
  - **Historie** mit Erstellungs-/Änderungsdatum, Bearbeiternamen
  - **Deckblatt** mit Version/Datum; Seitennummern, **Inhaltsverzeichnis**
  - **(Quer-)Bezüge** ermöglichen: Abschnittsnummern, ...

Wie kommt man zur Spec?

Ziele, Ideen, Anfragen,
Wünsche, Szenarien...

analysieren
ausformulieren
modellieren
vereinheitlichen

rückfragen, nacherheben,
präzisieren, Lücken füllen

(Anforderungs-)
Spezifikation

Entwurf 1

Entwurf 2

...

Version 1.0

...

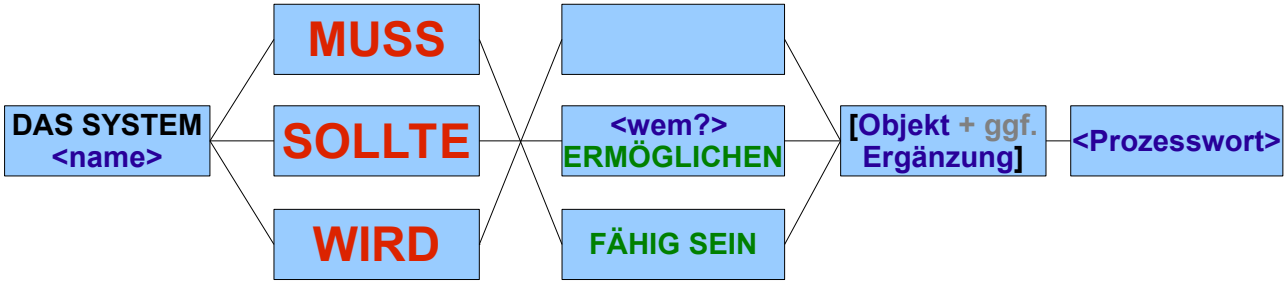
- Iterativer Prozess, in Abstimmung mit Stakeholdern
- Je (Teil-)Aspekt **geeignete** Beschreibungsmittel **wählen**
- Natürliche Sprache? (Semi-)Formale Modelle? ...

Natürlichsprachliche Dokumentation

- Unmittelbar von Stakeholdern ~~verstehbar~~ lesbar.
- Wenige Einschränkungen,
  alles lässt sich „irgendwie“ beschreiben.

- Aber Gefahr von
  - Mehrdeutigkeiten
  - Unvollständigkeiten
  - Unübersichtlichkeit
  - mangelnder Präzision
    („wir wissen schon, was *gemeint* ist“)
  - ...

Hilfsmittel „Satzschablonen“



- *„Das System MUSS fähig sein*, seine Benutzerdatenbank mit dem Unternehmens-Verzeichnisdienst zu synchronisieren.“
  - **muss**: verpflichtend, rechtlich verbindlich
  - „**fähig sein“:** Schnittstellenanforderung

- „Die Schnorchel-App *SOLLTE* jeden Login-Vorgang protokollieren.“
  - **sollte**: nicht verpflichtend, aber erhöht Stakeholderzufriedenheit
  - „ “ selbständige Systemaktivität

- *„Die GUI-Komponente WIRD dem eingeloggten Benutzer ermöglichen, die Schriftgröße einzustellen.“*
  - **wird**: *zukünftig* zu integrieren, verpflichtend - jetzt vorbereiten
  - „**jmd. ermöglichen“:** Benutzerinteraktion (Ergebnis, *nicht* „wie“)

Exotisch? Realweltbeispiel:

...

8-bit message content transmission **MAY** be requested of the server by a client using extended SMTP facilities, notably the "8BITMIME" extension, RFC 1652 [22]. 8BITMIME **SHOULD** be supported by SMTP servers. However, it **MUST NOT** be construed as authorization to transmit unrestricted 8-bit material, nor does 8BITMIME authorize transmission of any envelope material in other than ASCII. 8BITMIME **MUST NOT** be requested by senders for material with the high bit on that is not in MIME format with an appropriate content-transfer encoding; servers **MAY** reject such messages.

...

aus RFC 5321 (SMTP)

Siehe RFC2119 „Key words for use in RFCs to Indicate Requirement Levels“, http://www.ietf.org/rfc/rfc2119.txt

Alles klar?

Gegenschlag:

Glossar

- Sammlung von **Erklärungen** (mit Quelle) für
  - Fachliche Begriffe des Anwendungsumfelds (Bankwesen, Logistik, Waschbärenzucht, ...)
  - Abkürzungen (z.B. LGDaSMod? MUFI? DoSV?)
  - Synonyme (z.B. „Barwert“ und „Gegenwartswert“)
  - Homonyme (z.B. „Auskehrung“ und „Auskehrung“)

Glossar

- Projektweit mit Stakeholdern **abgestimmt** und projektweit sichtbar.
- Wird ständig weiterentwickelt.
- Begriffe projektweit **einheitlich** verwendet (insb. in Dokumenten)
  - vermeidet Missverständnisse
  - erleichtern neuen Projektmitarbeitern die Einarbeitung
  - liefert Vokabular für Bezeichner in der Software (z.B. für nachvollziehbare Klassen- / Modulnamen)

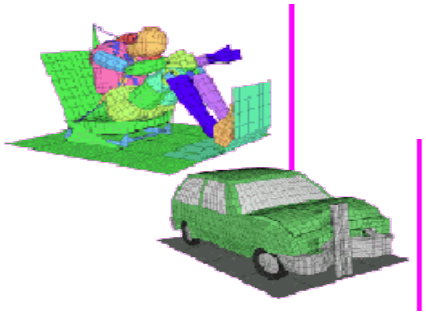
Verwendung von Modellen

- **Grafische** Information (Diagramme) besser erfassbar
- **Konzentration** auf bestimmten Systemaspekt
- **Definierte Bedeutung** der Diagrammelemente
- **Überblick** bei komplexen Problemen

- Modellierungssprache muss erlernt werden
- Modelle erfordern in der Regel eine begleitende (natürlichsprachliche) Erläuterung

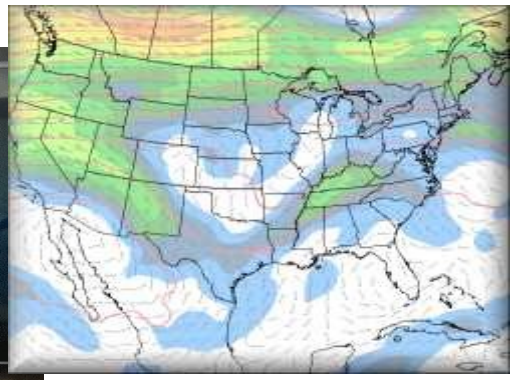
Was ist ein Modell?

- Ein **Modell** beschreibt einen relevanten Ausschnitt des interessierenden Systems. Dies wird erreicht durch...
  - Vereinfachung
  - Abstraktion
  - Weglassen unwichtiger Details
- Darstellung: Graphisch, Text, Formeln, ...
- Ermöglicht, **interessierende Aspekte** komplexer Systeme zu **untersuchen**, zu **verstehen** und anderen verständlich zu machen (zu **kommunizieren**).



Modelle im Automobilbau

Architekturmodell



Wettermodell

...er hat's erkannt

Was modelliert man bei Software?

- Teile der **Realwelt**
  - Wer soll mit dem System arbeiten?
  - Welche Daten sind relevant und
  - wie hängen sie zusammen?
  - Arbeitsabläufe
  - ...
- Teile des **Systems** und seiner **Umgebung**
  - Übersicht über das Gesamtsystem (Architektur)
  - Objektstrukturen
  - Zusammenspiel von Systemkomponenten (und Umgebung)
  - interessante Verarbeitungsschritte
  - Systemzustände
- Ziele wiederum: verstehen, kommunizieren, dokumentieren
- Hilfreich dabei: verbreitete/einheitliche Notation

Unified Modeling Language (UML)

- Anfang der 80er Jahre: Einzug der Objektorientierung in die Praxis
  (z.B. Programmiersprache *Smalltalk*, Entwicklung von C++)
- Entwicklung mehrerer objektorientierter Methoden,
  jeweils mit eigner Notation. Darunter:
  - Object Oriented Analysis and Design (Grady Booch)
  - Object Modeling Technique (Jim Rumbaugh)
  - Object Oriented Software Engineering (Ivar Jacobson)
- 1994: Rumbaugh wechselt zur Firma *Rational*,
  wo Grady Booch bereits arbeitet
- 1995: "Unified Method 0.8" von Booch und Rumbaugh;
  Rational kauft Ivar Jacobsons Firma *Objectory*
- Zusammenarbeit der "*drei Amigos*" bei Rational
- 1997: "Unified Modeling Language 1.0" (UML 1.0)
- Ende 2003: UML 2.0

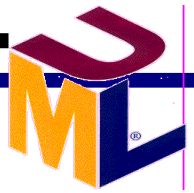
Struktur-Diagrammarten der UML

http://www.uml.org

Strukturaspekt (zeitunabhängig)

- **Klassen**diagramm, Objektdiagramm
  - Beziehungen zwischen Klassen bzw. konkreten Objekten
- **Verteilungs**diagramm
  - z.B. Verteilung von Systemteilen auf einzelne Rechner
- **Komponenten**diagramm
  - Systemkomponenten und deren Verbindungen
- **Kompositions-Struktur**-Diagramm
  - Schnittstellengruppierungen von Komponenten
- **Paket**-Diagramm
  - Ablagestruktur / Abhängigkeiten

Verhaltensdiagramme der UML



http://www.uml.org

- **Aktivitäts**diagramm
  - Abfolgen von Aktionen, z.B. in Geschäftsprozessen
- **Zustands**diagramm
  - Objektzustände und mögliche Zustandsübergänge
- **Anwendungsfall**-Diagramm
  - Beziehungen zwischen Akteure, Anwendungsfälle
- **Interaktionsdiagrammtypen:**
  - **Sequenz**diagramm, **Kommunikations**diagramm
    - Abfolge des Nachrichtenaustauschs
  - **Interaktions**-Übersicht - Kombination Sequenz+Aktiv.diagr
  - **Timing**-Diagramm - zeitl. Bedingungen an Obj.zustände

Anwendungsfall-Modellierung

WER? Akteure identifizieren

- In den gesammelten Szenarien identifizieren wir...
  - **Akteure** (Personengruppen / Rollen),
    die als Nutzer des Systems auftreten können
    - z.B. „Autofahrer“, „Chef“, „Angestellter“, ...
  - „spezielle Akteure“ (Schnittstellen zu externen Systemen)
    - z.B. WebShop ↔ Kreditkarten-System
- Die gleiche physische Person kann in verschiedenen **Rollen** mit dem System interagieren,
- Beispiel: Zeiterfassungssystem
  - Administrator kann es als „normaler Nutzer“ verwenden oder
  - (mit speziellen Rechten) das System konfigurieren

WAS? Anwendungsfall-Übersicht

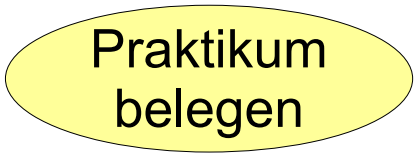
Akteur	Ziel / Funktion	Kurzbeschreibung
Lagerarbeiter	Zugänge verbuchen	Lagerarbeiter erfasst neue Lieferungen bei Eingang
Buchhalter	Zahlung auslösen	Nach vollständiger Lieferung wird eine Überweisung an Lieferanten ausgelöst
...		

- Tabelle mit Akteuren, deren Zielen und einer Kurzbeschreibung
- Verschafft ersten Überblick
- Kann mit Prioritäten ergänzt werden
- Wird später weiter ausgearbeitet

UML Anwendungsfalldiagramm: Elemente

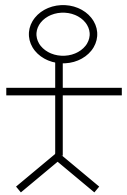
- Ein **Anwendungsfall** beschreibt eine abgeschlossene, typische Interaktion zwischen einem Benutzer (genauer: Akteur) und dem System, die zu einem erkennbaren Ergebnis führt (→Benutzer-Sicht)

  UML-Symbol:

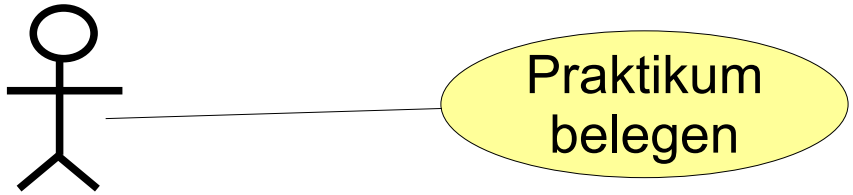


- Ein **Akteur** interagiert mit dem System, liegt aber *außerhalb* des Systems (Mensch, anderer Rechner, ...)

  UML-Symbol:

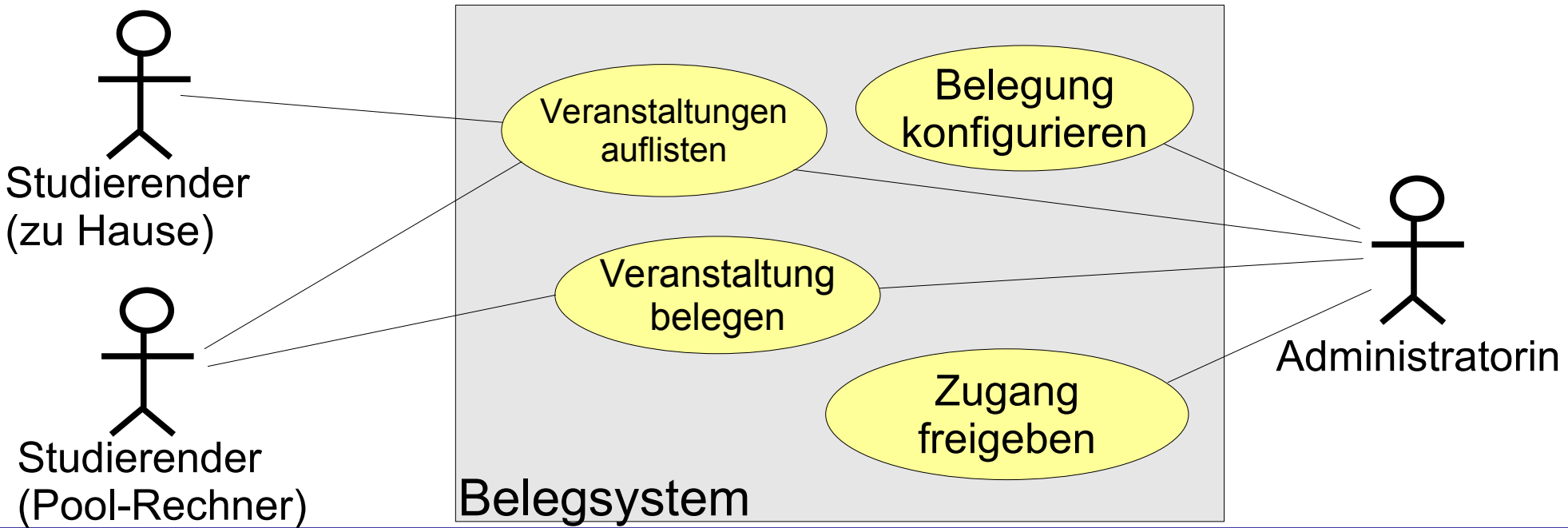


- **Linie** (Assoziation) verbindet Akteure mit Anwendungsfällen



Anwendungsfall-Diagramm (UML)

- Hilfsmittel zur Anforderungsermittlung ("***was***, nicht *wie*")
- Illustriert textuelle Beschreibungen, ersetzt sie aber *nicht*
- **Regeln**: Jeder Anwendungsfall
  - hat mindestens einen Akteur zugeordnet
  - hat einen *fachlichen* Auslöser
  - erzeugt ein (für seine Akteure) fachlich relevantes Ergebnis



Umfassendere Anwendungsfall-Beschreibung

- Zu **jedem** Anwendungsfall gibt es eine genauere Beschreibung
- Verschiedene Darstellungsformen möglich (tabellarisch, Fließtext, ...)

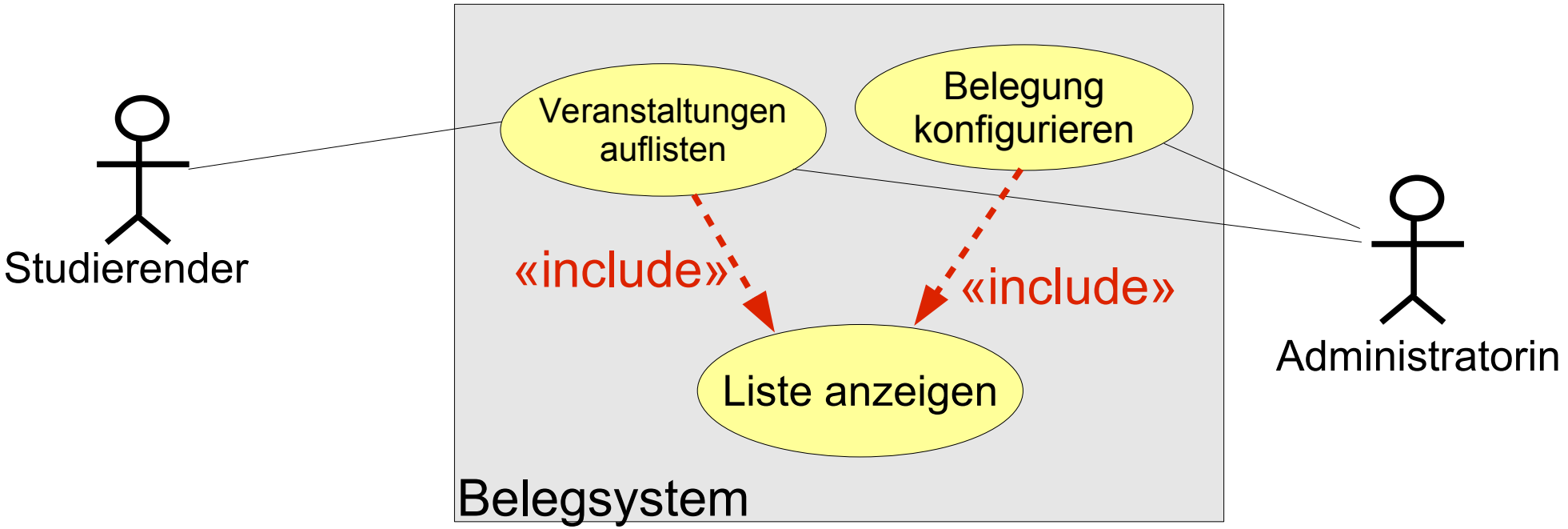
Titel: Veranstaltungsbelegung konfigurieren
Akteure: Administratorin
Fachlicher Auslöser: Kursbelegungen zum Semesterbeginn sind vorzubereiten
Vorbedingungen: Veranstaltungen sind in Datenbank eingerichtet
Standardablauf:
- 1. Administratorin: Veranstaltungen und Belegungszeitraum eingeben
- 2. System: Vollständigkeit und Plausibilität der Eingabe sicherstellen
- 3. System: Belegungsplan erzeugen und zur Bestätigung anzeigen
- 4. Administratorin: Planung bestätigen
- 5. System: Belegungsplan zur Freischaltung für vorgegebenen Zeitraum einplanen

Alternative Abläufe / Fehlersituationen / Sonderfälle:
- 4a Administratorin lehnt Planung ab
  - 4a1 System schlägt Alternativlösung vor
  - 4a2 Administratorin macht ggf. Anpassungen
  - 4a3 weiter bei 2

Nachbedingung/Ergebnis:
Anmeldesystem geht in Zustand „wartet auf Belegungsstart“ über
Nicht-funktionale Anforderungen
Reaktionszeit <10 sec.
Parametrisierbarkeit / Flexibilität:
Je Semester konfigurierbarer Hinweistext
Nutzungshäufigkeit / Mengengerüst: 2x jährlich (zum Semesterbeginn)

«include»-Beziehung

- „Ein Anwendungsfall ist **Bestandteil** eines anderen.“
  (*nicht* allgemeine Reihenfolge-Voraussetzung „nach A kann man B tun“)
- Beziehung nur zwischen Anwendungsfällen!
- Wiederverwendung, keine Vererbung
  (spiegelt sich auch in der Beschreibung)
- Notation: gestrichelter Pfeil mit Beschriftung «include»

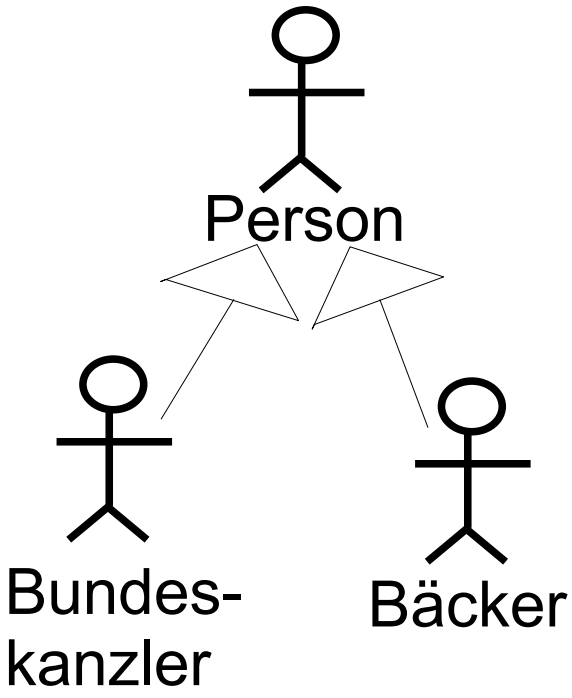


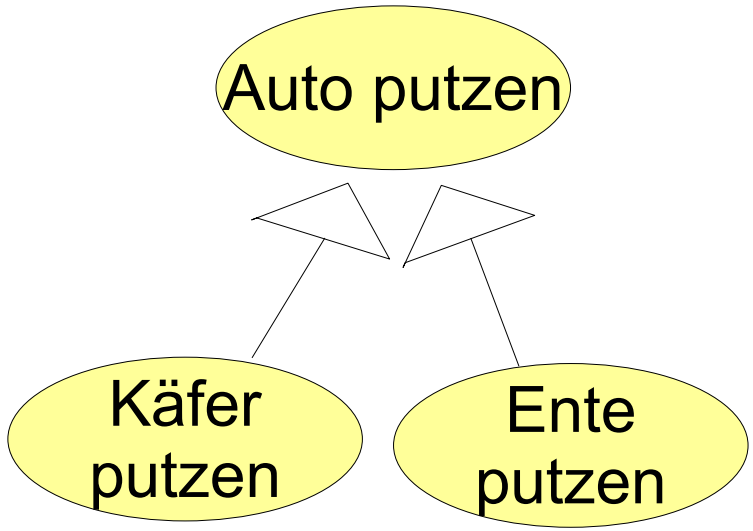
«extend»-Beziehung

- Ein Anwendungsfall ist **optionale Erweiterung** eines anderen (... der also auch ohne die Erweiterung gültig wäre)
- Basis-Anwendungsfall deklariert dazu "**Erweiterungsstellen**" (extension points), auf die sich der erweiternde Anwendungsfall bezieht
- *Keine* Vererbungsbeziehung (und keine allg. Reihenfolge-Abh.)
- Notation: gestrichelter Pfeil mit Beschriftung «extend»

Spezialisierungs-Beziehung

- "A" ist eine Spezialisierung von "B" ("*A* ***ist_ein*** *B*")
- Vererbung, „A“ erbt die Eigenschaften von „B“
- Notation: Durchgezogener Pfeil mit hohler Spitze
- Zulässig unter Akteuren und unter Anwendungsfällen
  (nicht gemischt! - was sollte das auch bedeuten?)





include / extend / Spezialisierung

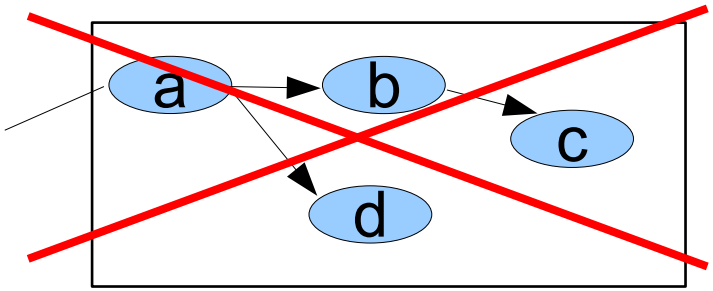
- **«include»**
  - **Herausziehen** gemeinsamen Teilverhaltens aus mehreren Anwendungsfällen
  - Vermeiden von Wiederholungen
- **Spezialisierung**
  - **Vererbung**
  - Hinzufügen / (teilweises) Verändern von Verhalten
- **«extend»**
  - ebenfalls Variation des "Basisverhaltens" (wie Spezialisier.)
  - kontrollierter, "zielgenauer" als Spezialisierung
  - stellt ***optionales*** **(Zusatz-)Verhalten** dar
    (erweiterter Use-Case ist *auch ohne* Erweiterung sinnvoll)

Obacht!

- **Nicht** "im Usecase-Diagrammen programmieren" (!!!!!!!!!!!)

!!

  - **keine** Datenflüsse
  - **keine** Abläufe/Zustandsübergänge im unten gezeigten Sinne
  - Reihenfolgen ggf. nur über Vor-/Nachbedingungen in der Use-Case-Beschreibung - include/extend dienen **nicht** dazu, Reihenfolgeabhängigkeiten zu modellieren
    (dazu gibt es die Vor-/Nachbedingungen in den Usecase-Beschreibungen)



„*zuerst muss man a machen,*
danach kann man b oder d,
wenn man b gemacht hat, kann man c nutzen“

FALSCH

- Use-Case-Diagramme dienen zur besseren **Übersicht**, man kann bei weitem **nicht alles** in Use-Case-Diagrammen ausdrücken (und soll das auch nicht)!
- Verteilen Sie Ihren Modellierungsdrang daher geschickt auf Begleittext und die richtigen Modell-Typen für denjeweils darzustellenden Aspekt (z.B. Aktivitätsdiagramme für Abläufe).

Hinweise

- Keine unnötigen Details, einfache Sätze verwenden
- Klarmachen, **wer** (Akteur bzw. das System) was tut
- **Außensicht** wahren (als nicht vom System aus die Welt sehen, sondern umgekehrt)
- Nicht zu kleinteilig beschreiben
- **Absicht** des Akteurs beschreiben,
  nicht einzelne Handgriffe
  - Gut: *Benutzer gibt Adressdaten ein*
  - Schlecht: *Benutzer klickt Straßenfeld ein, Benutzer gibt Strasse ein, Benutzer drückt Tab, Benutzer gibt Postleitzahl ein...*
  - User-Interface-Details vermeiden
- **IFs** / Verzweigungen vermeiden (nicht in UseCases „programmieren“!)

[Cockburn2001]