

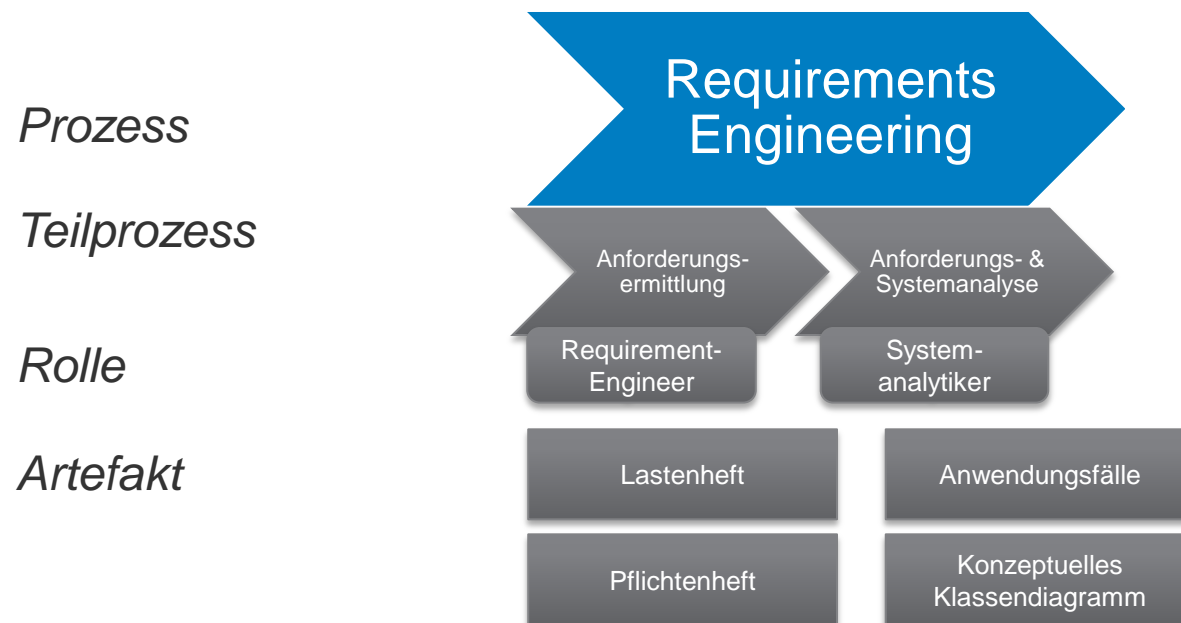
Einführung in die Softwaretechnik

2. Requirements Engineering

Prof. M. Broy (I4), Prof. B. Brügge, (I1), Prof. F. Matthes (I19), **Prof. A. Pretschner (I22)**

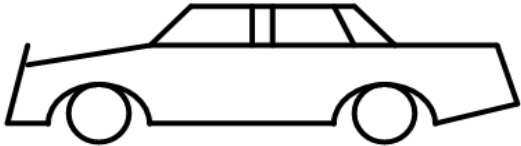
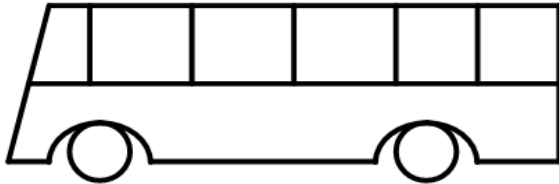
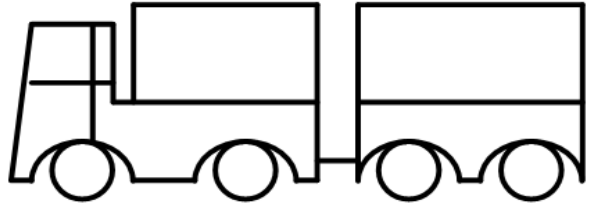
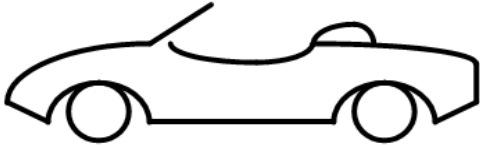
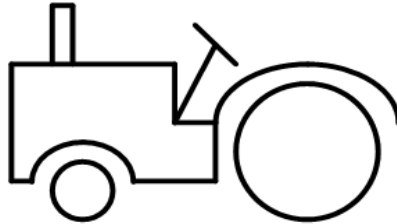
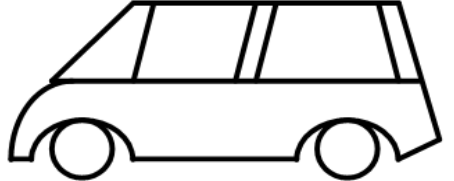
2 Requirements Engineering

- Überblick und Motivation
- Anforderungsermittlung
- Anforderungs- und Systemanalyse mit UML
 - Modellierung von Szenarien mit UML Anwendungsfalldiagrammen
 - Konzeptuelle Datenmodellierung mit UML Klassen- und Objektdiagrammen



- **Requirements** (dt. Anforderungen) beschreiben das gewünschte Verhalten und geforderte/vereinbarte Eigenschaften des zu entwickelnden Systems.
- Anforderungen müssen:
 - erarbeitet werden (finden, bewerten)
 - dokumentiert, modelliert und strukturiert werden
 - analysiert und qualitätsgesichert werden
 - verwaltet werden
- **Requirements Engineering** ist der Teil des Software Engineering, in dem Datenstrukturen und Methoden dafür entwickelt werden.
- Innerhalb des Gesamtprozesses steht das Requirements Engineering an erster Stelle
- Besonders bei neuartigen („innovativen“), sicherheitskritischen und komplexen Systemen (Bsp. eingebettete Systeme) ist ein systematisches und effizientes Requirements Engineering unerlässlich ...

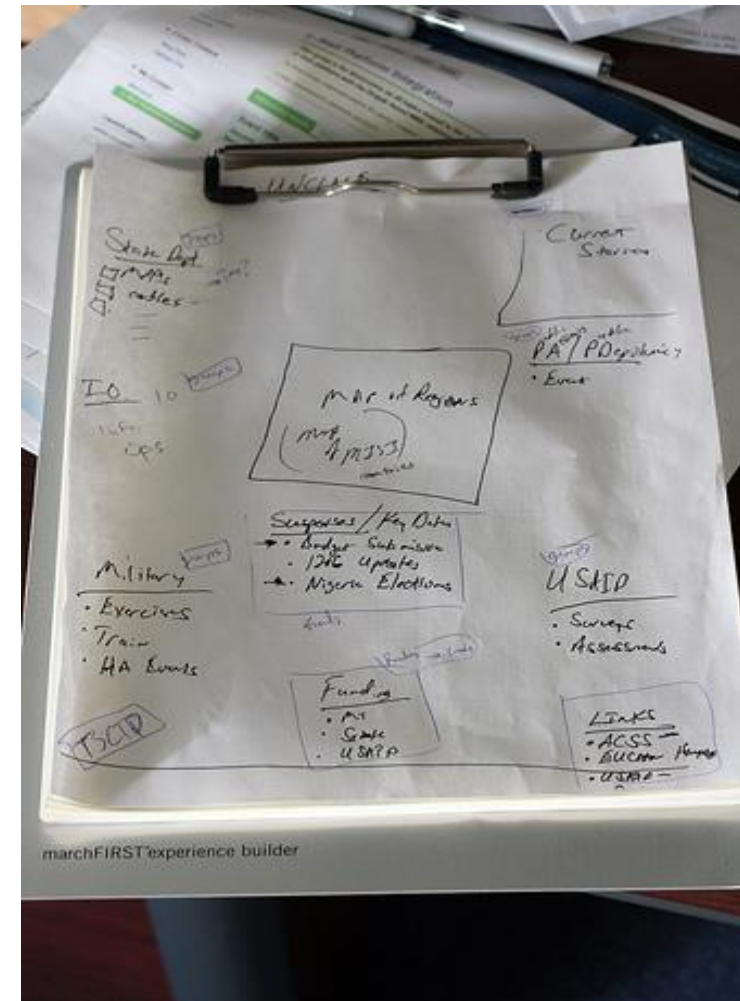
Bedeutung des Requirements Engineering

<p>Was der Anwender wollte</p> 	<p>Wie es der Anwender dem Programmierer sagte</p> 	<p>Wie es der Programmierer verstanden hat</p> 
<p>Was der Programmierer bauen wollte</p> 	<p>Was der Programmierer tatsächlich gebaut hat</p> 	<p>Was der Anwender tatsächlich gebraucht hätte</p> 

[Si02]

Bedeutung des Requirements Engineering in der Praxis

- Requirements Engineering wird oft
 - unterschätzt
 - nicht sorgfältig genug durchgeführt oder
 - ungenügend beherrscht
- Requirements Engineering bestimmt mit seinen Festlegungen entscheidend
 - die Funktionalität
 - die Kosten und damit
 - die Nützlichkeit und Komplexität eines Systems/Teilsystems
- Fehlentscheidungen im Requirements Engineering belasten den ganzen folgenden Entwicklungsprozess und das Produkt im Feld
 - bis zu 50% von Kundenreklamationen sind auf ungenügendes Requirements Engineering zurückzuführen
 - instabile oder unzutreffende Anforderungen sind ein erfolgskritisches Projektrisiko



- Kunden/Auftraggeber
 - sind ihre tatsächlichen Anforderungen zu Beginn oft nicht bewusst, da sie oft nur schwer einschätzen können, was es bedeutet, ein neues System einzuführen und welches Potential dadurch gegeben ist.
 - können ihre Bedürfnisse oft nicht angemessen formulieren
 - sind oft nicht in der Lage, RE-Dokumente zu lesen und zu bewerten
 - (und manchmal gibt es die so direkt gar nicht)
- Entwickler denken in Lösungen, nicht in Problemstellungen und Anforderungen
- Anforderungen ändern sich im Lauf einer Entwicklung und verursachen hohe Änderungskosten
- Unterschiedliche Interessenlagen führen auf Konflikte bei der Anforderungsfestlegung
- Anforderungen umfassen fachliche, technische, organisatorische, ökonomische Aspekte, die oft im Gegensatz zueinander stehen und in ihrer Gesamtheit kaum zu überblicken sind
- Fehler in den Anforderungen werden oft erst bei der Systemerprobung sichtbar

- ABER: Das ist auch nicht so einfach!

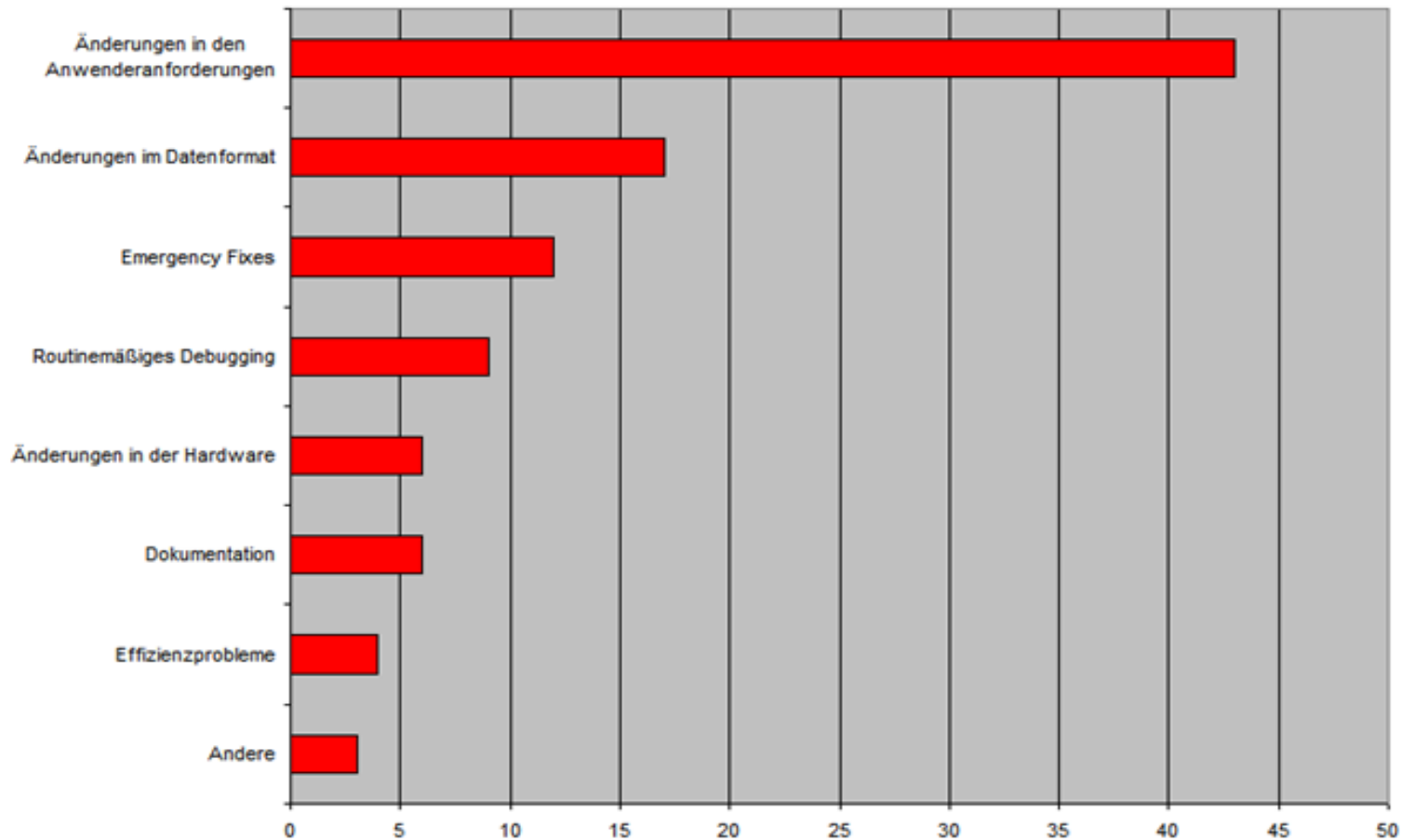
Statistiken: Chaos-Report (1994) der Standish Group

- 44% der Gründe für das Scheitern eines Projektes liegen im unzureichenden Umgang mit Anforderungen
- Häufigster Grund für das Scheitern eines Projektes sind unvollständige Anforderungen

Faktoren, die den Projekterfolg verhindern	% der Antworten
Unvollständige Anforderungen	13,1
Mangelnde Einbeziehung der Nutzer	12,4
Zu wenig Ressourcen	10,5
Unrealistische Anforderungen	9,9
Mangelnde Unterstützung durch das Management	9,3
Sich ändernde Anforderungen	8,7
Mangelhafte Planung	8,1
Produkt wurde nicht mehr benötigt	7,5
Mangelhafte technische Infrastruktur	6,2
Mangelndes Verständnis der verwendeten Technologien	4,3
Sonstige	9,9

Statistiken:

- Chaos-Report (1994): Über 40% der späteren Wartungskosten haben ihre Ursache in Änderungen an den Anforderungen:



1. Gesetz von Boehm

Fehler passieren am häufigsten während des Requirements Engineerings und des Designs; je später sie entdeckt und entfernt werden, desto teurer sind sie.

Empirische Beobachtung der Kostenentwicklung:

Wird ein Fehler erst „im Feld“, d.h. beim Kunden entdeckt, kostet seine Beseitigung etwa 30-mal so viel als während der Implementierungsphase.

[EnRo03]

Eine Anforderung (engl. *Requirement*) ist:

- eine Bedingung/Fähigkeit/Eigenschaft, die ein **Stakeholder** * für ein Produkt oder einen Prozess fordert, um ein Problem zu lösen oder ein Ziel zu erreichen
- eine Bedingung/Fähigkeit/Eigenschaft, die ein System erfüllen/haben muss, um einen **Vertrag**, einen Standard, eine Spezifikation oder andere formal vorgegebene Dokumente zu erfüllen
- eine dokumentierte **Repräsentation** einer Bedingung/Fähigkeit/Eigenschaft wie in 1. oder 2. definiert
[IE90]

* „Interessenvertreter“

Aufgabenbereiche des **Requirements Engineering** sind:

- Anforderungen identifizieren und erfassen (*Elicitation*)
- Anforderungen abstimmen, bewerten, priorisieren und konsolidieren
- Anforderungen dokumentieren, modellieren, strukturieren und begründen
- Anforderungen analysieren und validieren (bezüglich Inhalt und Darstellung)

Das **Requirements Management** beschäftigt sich mit dem Verwalten und Pflegen von Anforderungsdokumenten, insbesondere:

- Anforderungen ändern (Änderungsmanagement)
- Anforderungen verfolgen und verknüpfen (*Tracing*)
- Anforderungen verifizieren

- **Effiziente** Erarbeitung einer **qualitativ hochwertigen** Anforderungs- und Systemspezifikation
 - abgestimmt mit allen Stakeholdern (unterschiedliche Zielsetzung und Interessen)
 - abgestimmt mit allen Vorgaben und Rahmenbedingungen
 - bewertet nach Wirtschaftlichkeit und Machbarkeit
- Spezifikationsdokumente sind Grundlage
 - der Auftragsabstimmung und Vergabe
 - der Abstimmung zwischen den Stakeholdern (Kunde, Nutzer, Entwicklungsingenieure ...)
 - für Entwurf, Realisierung und Integration
 - der Abnahme (auch Testspezifikation)
 - zukünftiger (Weiter-) Entwicklungen

Ein **Stakeholder** ist eine Person, die an dem zu entwickelnden Produkt ein Interesse hat.

Am Beispiel „Auto“:

- Kunde (Käufer, z.B. auch Taxi-Unternehmen, Polizei...)
- Nutzer (z.B. Fahrer)
- Hardware-Entwickler (z.B. Mechaniker, Designer, Klimaanlageexperte...)
- Software-Entwickler
- Projektmanager
- Controlling
- Marketing
- Vertrieb
- Schulungspersonal (z.B. Handbuchschrreiber, Hotline, Fahrschulen)
- Werkstattpersonal
- TÜV
- ...

Stakeholder haben oft unterschiedliche Anforderungen – ein Beispiel



- Alle relevanten Informationsquellen und Interessensgruppen („Stakeholder“) einbeziehen.
- Ziele und Anforderungen sammeln, strukturieren, gruppieren, bewerten.
- Aus Zielen grobe („abstrakte“) Anforderungen ableiten.
- Mögliche Alternativen sauber herausarbeiten, abwägen, Entscheidungskriterien festlegen und transparent entscheiden. Entscheidungsgründe dokumentieren!
- Anforderungen präzise dokumentieren.
- Aus groben feine („konkrete“, messbare) Anforderungen ableiten.
- Details durch Modellbildung genau erfassen.
- Von Anforderungen zur Systemspezifikation übergehen
- Durch eine frühe Modellbildung werden Widersprüche, Mehrdeutigkeiten und Unvollständigkeiten in Anforderungen erkannt und können gezielt bereinigt werden.
- Dieser Ansatz ist „artefaktorientiert“. Im Rahmen agiler Methoden sieht man das ein wenig anders.

Hauptrolle im Requirements Engineering: Der Requirements Engineer



Berufsbild

- Verantwortlich für Anforderungsermittlung und Systemmodellierung

Voraussetzungen

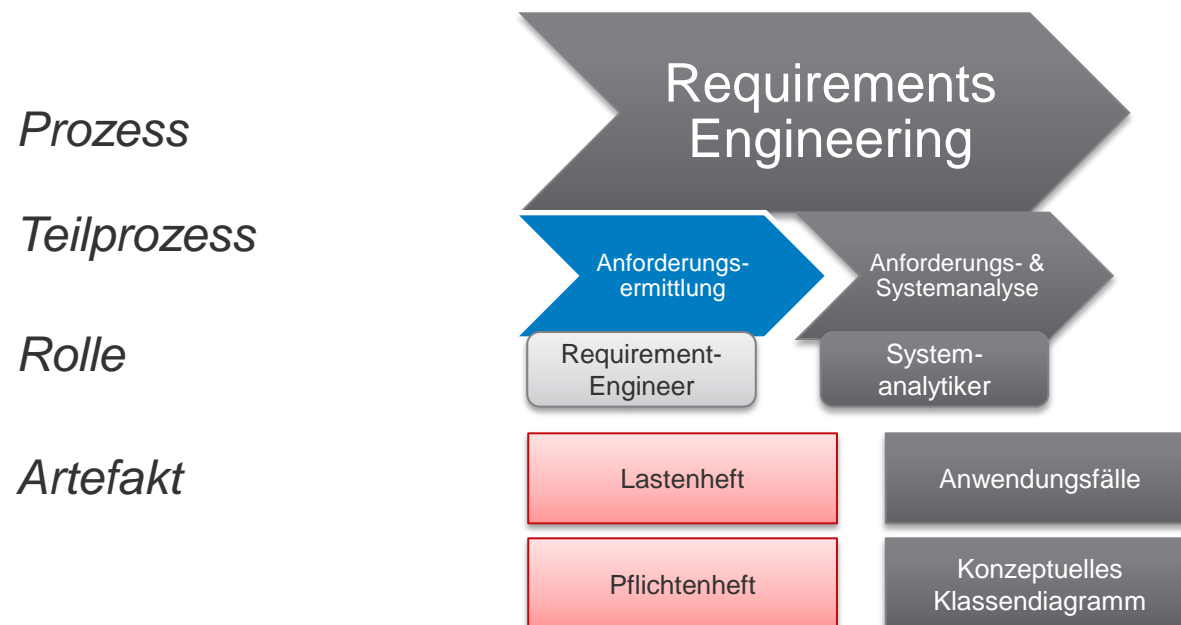
- Fähigkeit abstrakte Konzepte zu verstehen und anzuwenden
- Fähigkeit zur logischen Strukturierung und Analyse unscharf definierten Materialien
- Fähigkeit zur raschen Einarbeitung in spezielle Anwendungsgebiete
- Gute Verhandlungs- und Präsentationstechnik
- Kenntnisse in Entwicklungsmethoden und Systemarchitekturen
- Gute und klare Ausdrucksfähigkeit in Wort und Schrift

- Modellorientierung der Methoden:
 - Betonung der Systemmodellierung und Formalisierung in vielen Software-Entwicklungsmethoden und Lehrbüchern
 - In der Praxis: Anforderungsermittlung erfordert eine kreative, analytische und kommunikative Leistung des Requirements Engineers
 - Berücksichtigung aller relevanten Perspektiven bei der Systemmodellierung als Herausforderung des Requirements Engineers
- Zusammenspiel von Beobachten - Analysieren - Modellieren
- Hohe Bedeutung sozialer Kompetenz (offene, freundschaftliche, vertrauensfördernde Zusammenarbeit)

Attempt to understand before attempting to be understood.
(Alan Davis in IEEE Software, Juli/August 1998)

2 Requirements Engineering

- Überblick und Motivation
- Anforderungsermittlung
 - Dokumente
- Anforderungs- und Systemanalyse mit UML
 - Modellierung von Szenarien mit UML Anwendungsfalldiagrammen
 - Konzeptuelle Datenmodellierung mit UML Klassen- und Objektdiagrammen



Funktionale und nicht-funktionale Anforderungen sowie Beschränkungen



Die Menge der Anforderungen ist sehr heterogen → Klassifikation nützlich

Unterscheidung in funktionale und nicht-funktionale Anforderungen sowie Beschränkungen:

- **Funktionale Anforderungen** beschreiben die Interaktionen zwischen dem System und seiner Umgebung unabhängig von deren Realisierung.
Lassen sich als Aktivitäten formulieren:
Wenn der Fahrer den Blinkerhebel runterdrückt, muss der linke Blinker angehen.
- **Nicht-funktionale Anforderungen** beschreiben Eigenschaften des Systems, welche nicht Interaktionen darstellen. Lassen sich als negative Zusicherungen formulieren:
Die Antwortzeit muss unter einer Sekunde liegen.
- **Beschränkungen** (Pseudoanforderungen) bestimmen den Lösungsraum für die Realisierung.
Die Implementierung muss in Java erfolgen.

Verwendbarkeit (*Usability*)

- Der Grad der Einfachheit, mit dem Systemfunktionen verwendet werden können.
- Verwendbarkeit ist ein häufig missverständlich gebrauchter Ausdruck („Das System ist einfach zu verwenden.“)
- **Verwendbarkeit** muss **messbar** sein, ansonsten ist sie reines **Marketing**
Beispielhafte Kenngröße ist die Zahl der Schritte, um Online eine Bestellung abzuschließen, oder das Ergebnis von empirischen Benutzerstudien.

Robustheit (*Robustness*)

- Die Fähigkeit eines Systems eine Funktion aufrecht zu erhalten, obwohl
 - der Benutzer eine Fehleingabe macht oder
 - die Umgebung des Systems sich verändert.*Das System muss bei Temperaturen bis zu 90° C arbeitsfähig bleiben.
Das System darf auch beim Erreichen von Speicherengpässen keine Fehlbuchungen vornehmen.
Das System muss auch einen Schlafzustand eines Laptops unterstützen.*

Wartbarkeit (*Maintainability*)

- Die Einfachheit, mit welcher eine Funktion entsprechend neuer Anforderungen angepasst werden kann.

Verfügbarkeit (*Availability*)

- Das Verhältnis zwischen der erwarteten Up-Time eines Systems und der Summe von Up- und Down-Time.
Das System soll pro Woche nicht länger als fünf Minuten ausfallen.

Performanz (*Performance*)

- Beschreibt die Leistungsfähigkeit des Systems hinsichtlich Benutzeranfragen.
Das System muss 100 simultane Nutzer unterstützen.
Die Antwortzeit für alle Operationen bis auf die Suchoperation darf 500ms nicht überschreiten.

Anforderungen an die Benutzerschnittstelle:

- Gibt es verschiedene Arten von Benutzern des Systems?
- Welche Arten von Benutzern verwenden das System?
- Welche Eingabeunterstützung braucht ein Systembenutzer?

Dokumentation:

- Welche Art von Dokumentation ist notwendig?
- Welche Zielgruppen spricht die Dokumentation an?

Hardware-Umgebung:

- Auf welcher Zielhardware soll das System verwendet werden?
- Welche Eigenschaften hat die Zielhardware (z.B. Arbeitsspeicher)?

Lastverhalten:

- Gibt es Erfordernisse hinsichtlich Geschwindigkeit oder Durchsatz des Systems?
- Gibt es Größen- oder Kapazitätsbeschränkungen hinsichtlich der vom System zu verarbeitenden Datenmenge?

Hilfestellung zur Formulierung nicht-funktionaler Anforderungen (2)



Ausnahmebehandlung:

- Wie soll das System auf Eingabefehler reagieren?
- Wie soll das System auf außergewöhnliche Umstände reagieren?

Systemschnittstellen:

- Muss das System Eingaben von anderen Systemen entgegennehmen?
- Muss das System Ausgaben an andere Systeme weiterleiten?
- Gibt es Restriktionen hinsichtlich der Kommunikationsmedien?

Qualitätserfordernisse:

- Wie lange braucht das System zum Wiederaufstart nach einem Fehler?
- Wie lange darf das System in 24 Stunden nicht erreichbar sein?

Betriebsumgebung:

- In welcher Betriebsumgebung wird das System zum Einsatz gebracht?
- Gibt es eine oder mehrere (verteilte) Betriebsumgebungen?

Systemanpassungen:

- Welche Bestandteile des Systems unterliegen dem höchsten Anpassungsdruck?
- Welche Art von Anpassungen sind zu erwarten?

Sicherheitsfragestellungen:

- Muss der Zugriff auf das System besonders geschützt werden?
- Spielt physikalische Sicherheit des Systems eine Rolle?

Ressourcen- und Managementfragestellungen:

- Wie häufig werden die Systemdaten gesichert (Backup)?
- Wer ist für die Sicherung des Systems verantwortlich?
- Wer ist für die Systemwartung zuständig?

Was sind keine Anforderungen im Requirements Engineering?



- Vorgaben hinsichtlich der Systemstruktur
- Auswahl einer Entwicklungsmethodik
- Vorbestimmung einer Entwicklungsumgebung
- Vorgabe einer Programmiersprache
- Wiederverwendbarkeit

Es ist wünschenswert, dass keine derartigen Vorgaben durch den Kunden gemacht werden. Dies kann in der Realität dennoch passieren.

Beispiel für ein Kommunikationsproblem bei einer Systemanforderung

Londoner U-Bahn verlässt Bahnhof ohne Fahrer

Was ist passiert?

- Die U-Bahn weigerte sich anzufahren, da eine Türe eines Passagierabteils offen geblieben war – sie hatte sich verklemmt.
- Der Fahrer des Zuges verlässt sein Führerhaus, um diese Türe zu schließen.
- Dabei lies er die Fahrtüre offen, da er wusste, dass der Zug mit einer offenen Türe nicht losfährt.
- Als der Fahrer die Türe des Passagierabteils schloss, setzte sich die U-Bahn in Bewegung und fuhr aus dem Bahnhof.



Grund: Die Fahrtür wurde durch das Steuerungssystem nicht als Türe behandelt.

Anforderungen werden sinnvollerweise mit Attributen versehen und strukturiert:

- Identifikation
- Name
- Anforderungsbeschreibung (textuell)
- Quelle, Autor
- Datum
- Klasse (z.B. Geschäfts-, Nutzungs-, Systemanforderung)
- Priorität
- Begründung
- Kommentar
- Links (z.B. zu Designmodellelementen, zu Zielen/Geschäftsanforderungen...)

Daher oft Verwaltung von Anforderungen in **Tabellen** (unter Nutzung von Datenbanken oder spezieller Anforderungswerkzeuge z.B. Doors, Excel) oder **Karteikarten** (z.B. Volere, CaliberRM).

„Anforderungen“ an eine Werkzeugunterstützung für das Anforderungsmanagement



- Abspeicherung der Anforderungen in einem zentralen Datenspeicher
 - Unterstützung für Mehrbenutzerzugriff auf die Anforderungen
 - Automatische Erzeugung eines Spezifikationsdokuments aus den Anforderungen
 - Unterstützung für die Änderung von Anforderungen
 - Änderungsverfolgung für die Anforderungen
-
- Integration von Anforderungsdatenbank und Fehlerdatenbank

DOORS (Telelogic, jetzt IBM)

- Werkzeug für das Anforderungsmanagement , speziell für geographisch verteilte Teams.

RequisitePro (IBM/Rational)

- Werkzeug mit Integration in MS Word. Unterstützt Vergleiche zwischen verschiedenen Projekten.

RD-Link (<http://www.ring-zero.com>)

- Bietet Abgleich von Informationen zwischen RequisitePro & DOORS.

Unicase (<http://unicase.org>)

- Forschungswerkzeug des Lehrstuhls Prof. Brügge für die geographisch verteilte Entwicklung von Systemmodellen.

Aktueller Stand der Praxis: Doors

'Functional Requirements' current 1.0 (Initial) in /Sports utility vehicle 4x2/Requirements (Formal module) - DOORS

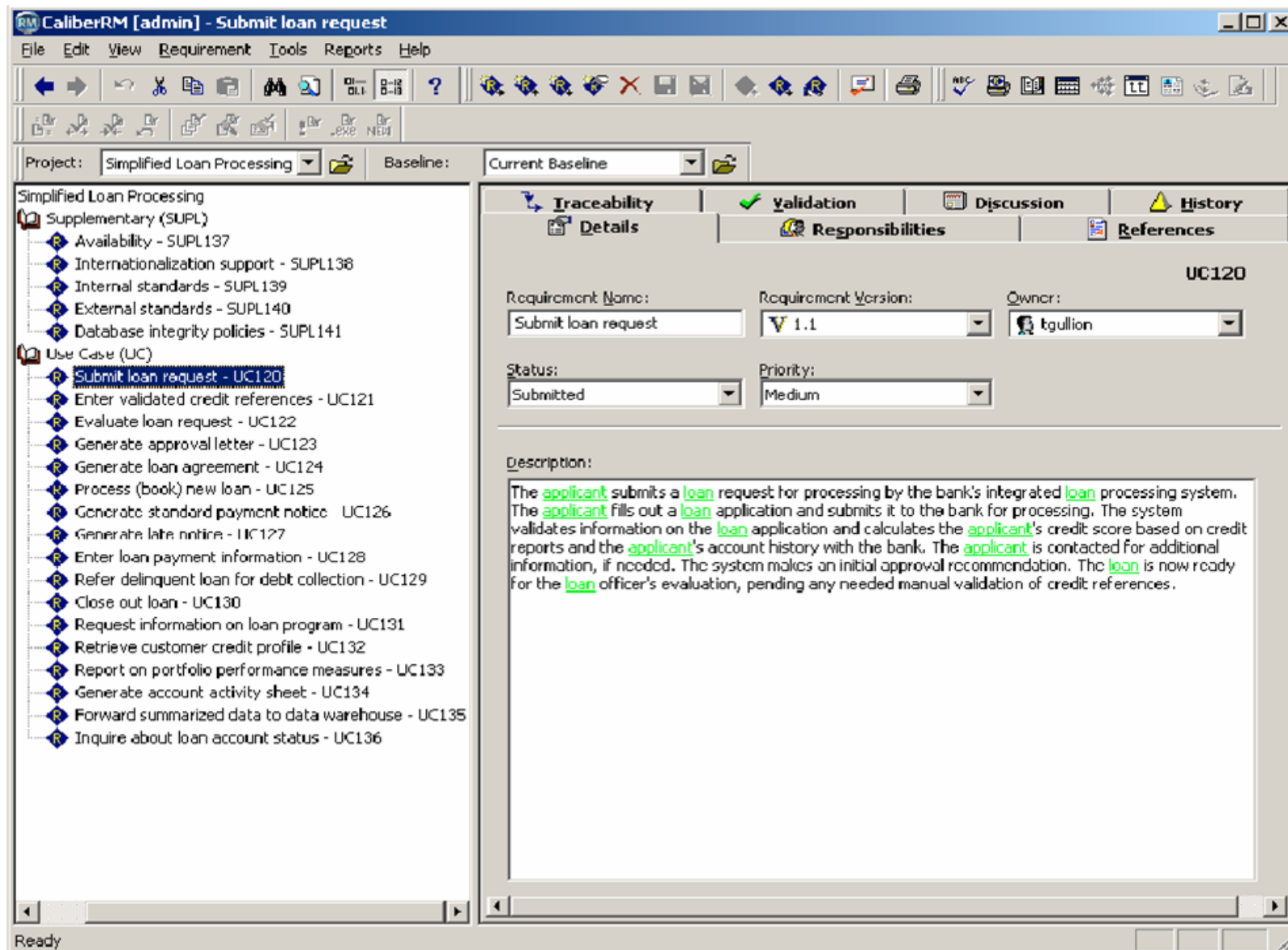
File Edit View Insert Link Analysis Table Tools User Analyst Help

View Test Run 002 All levels

Functional System requirements for SUV 4x2	Expected Test Result	Actual Test Result	Test Engineer 2	Test Status 2
the user requirements.				
The car will have a world wide market.	N/A			Pass
2 Functional Requirements				Exempt
2.1 Power car				Exempt
2.1.1 Move car				Exempt
2.1.1.1 Move forwards				Exempt
The car shall be able to move forwards at all speeds from 0 to 200 kilometers per hour on standard flat roads with winds of 0 kilometers per hour, with 180 BHP.	Car can travel forwards at 200kph.	202.4kph	Lew Hamilton	Pass
2.1.1.2 Move backwards				Exempt
The car shall be able to move backwards to a maximum speed of 20 Kilometers per hour on standard flat roads with winds of 0 kilometers per hour, with 180 BHP.	Car can travel in reverse at 20kph.	20.8kph	Steve Hart	Pass
2.1.2 Accelerate car				Exempt
The car shall be able to accelerate from 0 to 100 Kilometers per hour in 10 seconds on standard flat roads with winds of 0 kilometers per hour.	Car can accelerate from 0-100kph in under 10s.	8.7s	Steve Hart	Pass
The car shall be able to accelerate from 100 to 150 kilometers per hour at a rate of 5 kilometers per second on standard flat roads with winds of 0 kilometers per hour.	Car can accelerate from 100-150kph in 10s.	9.3s	Steve Hart	Pass
The car shall be able to accelerate from 150 to 200 kilometers per hour at a rate of 3 kilometers per second on standard flat	Car can accelerate from 150-200kph in 15s.	14s	Steve Hart	Pass

Username: John Ramsey Exclusive edit mode

Aktueller Stand der Praxis: CaliberRM



Aktueller Stand der Praxis: ACC

ID	Anforderungsbeschreibung	Abstimmung	Anforderungsklasse
68	Abstand zu vorausfahrendem Fahrzeug in Fahrgasse in Abhängigkeit von der Istgeschwindigkeit und der Abstandsvorgabe (1,2,3) halten	abgestimmt	AnforderungKunde
1	Fahren auf Gerade ohne Objekt komfortabel beschleunigen, bis Wunschgeschwindigkeit erreicht, dann Wunschgeschwindigkeit halten	abgestimmt	AnforderungKunde
4	Kurz nach Objektverlust in Kurve und Wunschgeschwindigkeit noch nicht erreicht: zuerst konstant weiterfahren, dann ganz langsam beschleunigen bis komfortable Querschleunigung erreicht	abgestimmt	AnforderungKunde
5	Kurz nach Objektverlust auf Gerade und Wunschgeschwindigkeit noch nicht erreicht: komfortabel beschleunigen bis Wunschgeschwindigkeit erreicht	abgestimmt	AnforderungKunde
6	Kurz nach Objektverlust und Blinker Rechts gesetzt: Geschwindigkeit halten, da möglicherweise Autobahnausfahrt	abgestimmt	AnforderungKunde
7	Kurz nach Objektverlust bei höherer Geschwindigkeit und Blinker Links gesetzt: Beschleunigen, da möglicherweise Überholvorgang durchgeführt wird	abgestimmt	AnforderungKunde
8	Kurz nach Objektverlust und Gaspedal vom Fahrer betätigt: zügig beschleunigen	abgestimmt	AnforderungKunde
9	Annäherung an Kurve voraus: frühzeitig verzögern, so daß bei Kurveneintritt komfortable Kurvengeschwindigkeit erreicht wird	abgestimmt	AnforderungKunde
10	Bremspedal vom Fahrer betätigt: MCC ausschalten	abgestimmt	AnforderungKunde
11	Gaspedal vom Fahrer betätigt: wenn Drehmomentanforderung des Fahrers größer als die des MCC/ Tempomats, hat diese Vorrang. Nach Loslassen des Gaspedals soll Übergang in Wunschgeschwindigkeit komfortabel gestaltet werden	abgestimmt	AnforderungKunde
12	glatte Fahrbahn oder geringe Sichtweite: Wunschgeschwindigkeit anpassen, nur wenig beschleunigen	abgestimmt	AnforderungKunde
13	bei Veränderung der Wunschgeschwindigkeit durch Fahrer: komfortabel beschleunigen oder verzögern, bis neue Wunschgeschwindigkeit erreicht	abgestimmt	AnforderungKunde
15	Langsames Objekt, näher als Sollabstand & vorausfahrendes Fahrzeug bremst: deutlich verzögern	abgestimmt	AnforderungKunde
17	Neues, deutlich langsames Objekt, näher als Sollabstand: verzögern, um Kollision zu vermeiden, jedoch nicht stärker als ca. -0.2g	abgestimmt	AnforderungKunde
18	Neues, schnelleres Objekt, näher als Sollabstand: gleichmäßig weiterfahren und ggf. Sollabstand aufbauen	abgestimmt	AnforderungKunde
19	Neues, deutlich langsames Objekt, weiter als Sollabstand: konstant verzögern, jedoch nicht stärker als ca. -0.2g, bis Sollabstand erreicht, danach Sollabstand halten	abgestimmt	AnforderungKunde
20	Neues, nur wenig langsames Objekt, weiter als Sollabstand: mit geringer, konstanter Relativgeschwindigkeit annähern, dann aperiodisch in Sollabstand übergehen, danach Sollabstand halten	abgestimmt	AnforderungKunde

- *Elicitation*: Erfassen und Erarbeiten von Anforderungen
- Quellen/Techniken zur Erarbeitung von Anforderungen:
 - Konkurrenzprodukte
 - Analyse von Geschäftsprozessen (Beobachtung der Nutzer in ihrem Tagesgeschäft)
 - Vorgängerprodukte
 - Prototypen
 - Marktstudien
 - Gesetze und Normen
 - Interviews, Workshops (Befragung der Nutzer anhand vordefinierter Fragen)
 - Szenariomodellierung (Beschreibung der Systemverwendung als eine Folge von Interaktionen zwischen Nutzern und dem System)
 - Use-Cases (Abstraktionen, welche eine Klasse von Szenarien beschreiben)
- Systematisches Ableiten von Anforderungen: Systemanforderungen aus Nutzeranforderungen...

Typische Probleme

- „Stakeholder sind Menschen“
 - verlangen lieber zu viel als zu wenig
 - geben lieber eine falsche Antwort als keine
 - geben lieber keine Antwort, wenn sie dringendes zu tun haben
 - schützen ihre Interessen (z.B. Fachwissen)
 - kämpfen für mehr Einfluss
 - haben keine Lust
- Problemadäquate Kommunikation
 - Projektbeteiligte haben unterschiedlichen Hintergrund
 - Stakeholder haben Wissen über die *Problemdomäne*
 - Requirements Engineers haben Wissen über die *Lösungsdomäne*
- Festlegung der Systemgrenze und Weglassen ungewünschter Funktionen
- Unzweideutige Formulierung der Spezifikation

Anforderungen **hoher Priorität** (MUSS-Anforderungen)

- werden während der Phasen Analyse, Entwurf und Realisierung adressiert
- müssen zwingend in einer ersten Iteration umgesetzt werden

Anforderungen **mittlerer Priorität** (SOLL-Anforderungen)

- werden in den Phasen Analyse und Entwurf adressiert
- werden meist in einer zweiten Iteration umgesetzt

Anforderungen **niedriger Priorität**

- werden lediglich in der Analysephase betrachtet
- stellen dar, wie sich das System in der Zukunft entwickeln könnte
- nützlich für den Softwarearchitekten!

- Funktionsumfang ⇔ Benutzbarkeit
- Kosten ⇔ Robustheit
- Effizienz ⇔ Portabilität
- Kurze Entwicklungsdauer ⇔ Funktionsumfang
- Wiederverwendbarkeit ⇔ Kosten
- Abwärtskompatibilität ⇔ Wartbarkeit (in der Zukunft)

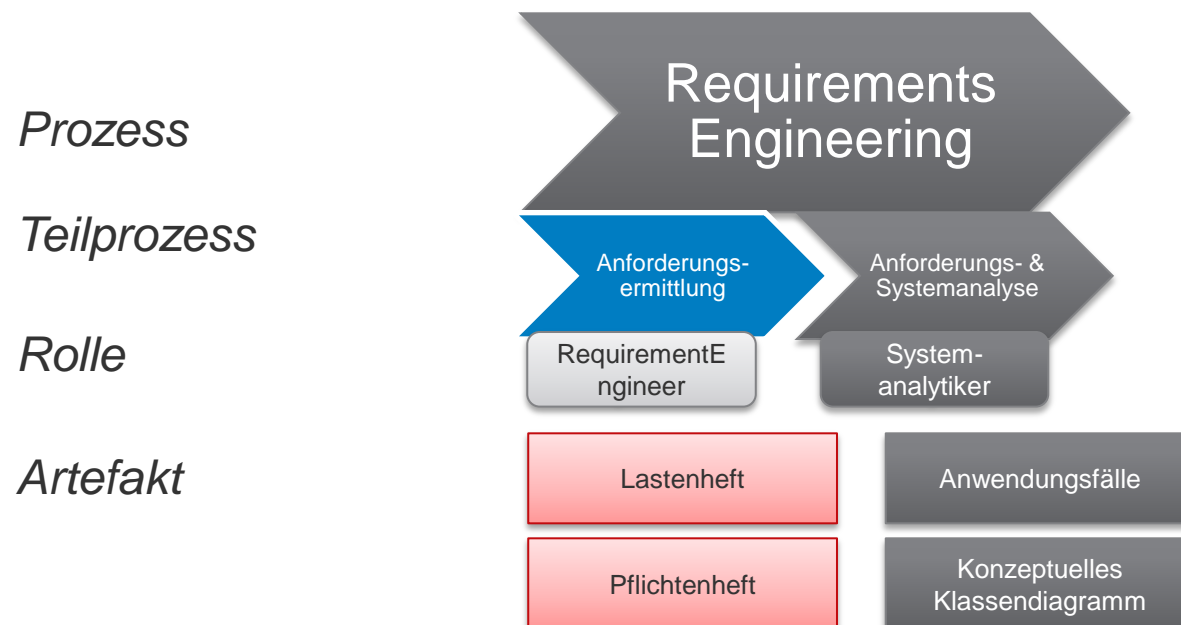
- Anforderungen kommen aus unterschiedlichsten Quellen
- Anforderungen sind oft Ausdruck von sehr individuellen Wünschen
- Konsequenz:
 - **Abstimmung:**
 - Welche Anforderungen fließen in die Spezifikation ein, welche werden ignoriert oder auf später verschoben
 - **Priorisierung:**
 - Welche Anforderungen müssen auf jeden Fall erfüllt werden (MUSS-Anforderungen), welche sollten erfüllt werden (SOLL-Anforderungen)
 - Oft wird im Vertrag festgelegt, dass 100% der MUSS und beispielsweise 75% der SOLL-Anforderungen erfüllt werden müssen
 - Oft verbunden mit **Release-Planung**: Welche Anforderungen sollen zuerst umgesetzt werden, welche später
 - **Konsolidierung:**
 - Beseitigen von Widersprüchen, Dokumentation des gemeinsamen Standes, Dokumentation des Einigungsprozesses

- Die erarbeiteten und abgeleiteten Anforderungen müssen in geeigneter Form dokumentiert werden, z.B.
 - als strukturiertes Textdokument (z.B. Word)
 - als attribuierte Anforderungen im Listenformat (z.B. Excel, Report generiert aus Doors)
 - als attribuierte Anforderungen im Karteikartenformat (z.B. Volere, CaliberRM)
- Lastenheft und Pflichtenheft (➔ nächste Folien)

- Wiederum der Hinweis: Diese Artefektorientierung wird bisweilen in Frage gestellt (-> Agilität)

2 Requirements Engineering

- Überblick und Motivation
- Anforderungsermittlung
 - Dokumente
- Anforderungs- und Systemanalyse mit UML
 - Modellierung von Szenarien mit UML Anwendungsfalldiagrammen
 - Konzeptuelle Datenmodellierung mit UML Klassen- und Objektdiagrammen



Lastenheft

Ziele, Strategie
Anforderungen
Rahmenbedingungen
+
Projektplanung
Risikobetrachtung

problemorientiert und abstrakt

Pflichtenheft

Detaillierte Anforderungs-
und Systemspezifikation
+
Detaillierte Projektplanung
Detaillierte Risikobetrachtung

*konkret, präzise, konsistent,
abgestimmt, vollständig, messbar*

- Vor der Erteilung eines Auftrags muss folgendes festgelegt werden
 - Was soll das System leisten?
 - Welche Qualitätseigenschaften soll das System haben? (Zusätzliche Forderungen)
 - Welche Kosten entstehen bei der Entwicklung des Systems?
- Der **Auftraggeber** erstellt dafür zunächst ein **Lastenheft**
 - Beschreibung aller Anforderungen aus Sicht des Auftragsgebers
 - Das Lastenheft ist Bestandteil der Ausschreibung
- Der potentielle **Auftragnehmer** erstellt eine Leistungsbeschreibung und schließlich ein **Pflichtenheft**
 - Konkretisierung aller im Lastenheft enthaltenen Anforderungen
 - Enthält auch eine Beschreibung der angestrebten Lösungsansätze
 - Pflichtenheft ist **Vertragsbestandteil** bei der Auftragsvergabe

1. Zielbestimmung
 - beschreibt die Ziele, die durch das Produkt erreicht werden sollen
2. Produkteinsatz
 - legt die Akteure, die mit dem Produkt arbeiten sollen, sowie deren Anwendungsbereiche fest
3. Produktumgebung
 - beschreibt den Nutzungskontext des Produkts, im Speziellen benachbarte Systeme
4. Produktfunktionen
 - beschreiben die Funktionen des Produkt (decken funktionale Anforderungen ab)

5. Produktdaten

- gibt die Kerndatenobjekte des Systems an und liefert eine Abschätzung zu erwarteten Datenmengen

7. Produktleistungen

- beschreibt wesentliche nicht-funktionale Anforderungen an das Produkt

8. Qualitätsanforderungen

- stellt (optional) das Produkt in den Kontext anwendbarer Qualitätsstandards
- priorisiert die nicht-funktionalen Anforderungen

9. Glossar

- gibt eine alphabetische Liste aller Kern- und Fachbegriffe der Anwendungsdomäne, wie sie im Lastenheft verwendet wurden

1. Zielsetzung
2. Allgemeine Beschreibung
3. Gegenwärtiges System (nur, falls vorhanden)
4. Vorgeschlagenes System
 1. Funktionale Anforderungen
 2. Nicht-funktionale Anforderungen
 3. Beschränkungen („Pseudoanforderungen“)
 4. Systemmodell → wird später diskutiert
 1. Szenarien
 2. Anwendungsfallmodell
 3. Objektmodell
 1. Datenkatalog
 2. Konzeptuelles Klassendiagramm
 4. Verhaltensmodell
 5. Benutzerschnittstelle
5. Zu erwartende Evolution des Systems
6. Glossar

Zielsetzung:

- Freitextformulierung des Verwendungszwecks des Systems

Allgemeine Beschreibung:

- Umgebung und *Scope* (Systemgrenzen)
- Generelle Funktion
Die Software deckt alle Funktionen zur Anlage und Verwaltung von Konten ab.
- Restriktionen
- Benutzer
Benutzer sind Schulkinder im Alter von 8 bis 12 Jahren.

Funktionale Anforderungen:

- Möglichst quantitativ, z.B. Tabellenform
Jeder Vorgang wird erfasst, so dass er nachvollziehbar ist.
- Eindeutig identifizierbar, z.B. eindeutige Nummerierung

Nicht-funktionale Anforderungen:

- Formulierung quantitativ und messbar
Das System muss 90% der Anfragen in weniger als 1 Sekunde beantworten.
- Eindeutig identifizierbar durch Nummerierung

Beschränkungen:

- Formulierung als Gebote oder Verbote
Das System muss in Java implementiert sein.

Zu erwartende Evolution des Systems:

- Aussagen hinsichtlich Betriebsdauer
- Grobe Planung zukünftiger Releases und zu erwartender Erweiterungen

Glossar:

- Sortierte Auflistung wichtiger Begriffe aus der Anwendungsdomäne

- Gliederung
 - abhängig von verwendeten Methoden und Sprachen
 - Wahl der Gliederung hat großen Einfluss auf Verständlichkeit
 - Gliederung teilweise durch Normen und Richtlinien bestimmt
- Bekannte Normen und Richtlinien
 - Allgemein: IEEE 830-1993, Volere Template, Software Cost Reduction
 - Spezifisch: VDMA-Systemspezifikation, VDA-Komponentenlastenheft
- Werkzeuge sind hilfreich
 - Strukturierung von Anforderungen und Tabellarische Anforderungsspezifikation (z.B. DOORS)
 - Graphische Beschreibung von Anforderungen mittels UML Struktur- und Verhaltensdiagrammen (z.B. Rational Rose)

Beispiel: IEEE Standard-Gliederung – IEEE Std 830-1998 „Software Requirements Specification“



Table of Contents of an SRS

1. Introduction
 - 1.1 Purpose
 - 1.2 Scope
 - 1.3 Definitions, acronyms, and abbreviations
 - 1.4 References
 - 1.5 Overview
2. Overall description
 - 2.1 Product perspective
 - 2.2 Product functions
 - 2.3 User characteristics
 - 2.4 Constraints
 - 2.5 Assumptions and dependencies
3. Specific requirements *(the largest & most important part of an SRS, organized in different manner)*
- Supporting information
 - Table of contents and Index
 - Appendixes *(e.g. Samples I/O formats, background, description of problem to be solved by the software)*

Teil 3 einer SRS kann enthalten:

- Externe Schnittstellen
- Funktionen
- Leistungserfordernisse
- Anforderungen an die logische Datenspeicherung
- Entwurfsbeschränkungen

Teil 3 einer SRS kann gegliedert werden nach:

- *system operation mode* (Betriebsmodus)
- *user class* (Benutzerkategorie)
- *feature* (Funktionen)
- ...

SRS Part 3 – organized by operation mode

3.1 External interface requirements

- 3.1.1 User interfaces
- 3.1.2 Hardware interfaces
- 3.1.3 Software interfaces
- 3.1.4 Communication interfaces

3.2 Functional requirements

...

3.1.i Mode i

- 3.1.i.1 Functional requirement i.1
- 3.1.i.2 Functional requirement i.2

...

3.1.i.n Functional requirement i.n

...

3.3 Performance requirements

3.4 Design constraints

3.5 Software system attributes

3.6 Other requirements

SRS Part 3 – organized by feature

3.1 External interface requirements

- 3.1.1 User interfaces
- 3.1.2 Hardware interfaces
- 3.1.3 Software interfaces
- 3.1.4 Communication interfaces

3.2 System features

...

3.1.i System feature i

- 3.1.i.1 Introduction/Purpose of feature
- 3.1.i.2 Stimulus/Response sequence
- 3.1.i.3 Associated functional requirements

...

3.1.i.3.m Functional requirements m

...

...

3.3 Performance requirements

3.4 Design constraints

3.5 Software system attributes

3.6 Other requirements

Korrektheit:

- Entspricht die aufgeschriebene Anforderung den tatsächlichen Bedürfnissen der Stakeholder?

Verständlichkeit:

- Ist die Anforderung klar verständlich oder ist sie unnötig komplex formuliert?

Eindeutigkeit:

- Ist die Anforderung eindeutig formuliert oder lässt sie Raum für Interpretationen?

Testbarkeit:

- Lässt sich die Erfüllung der Anforderung (z.B. bei der Abnahme der Software durch den Kunden) messen / belegen?

Strukturiertheit:

- Besteht eine nutzenstiftende Klassifikation der Anforderungen?

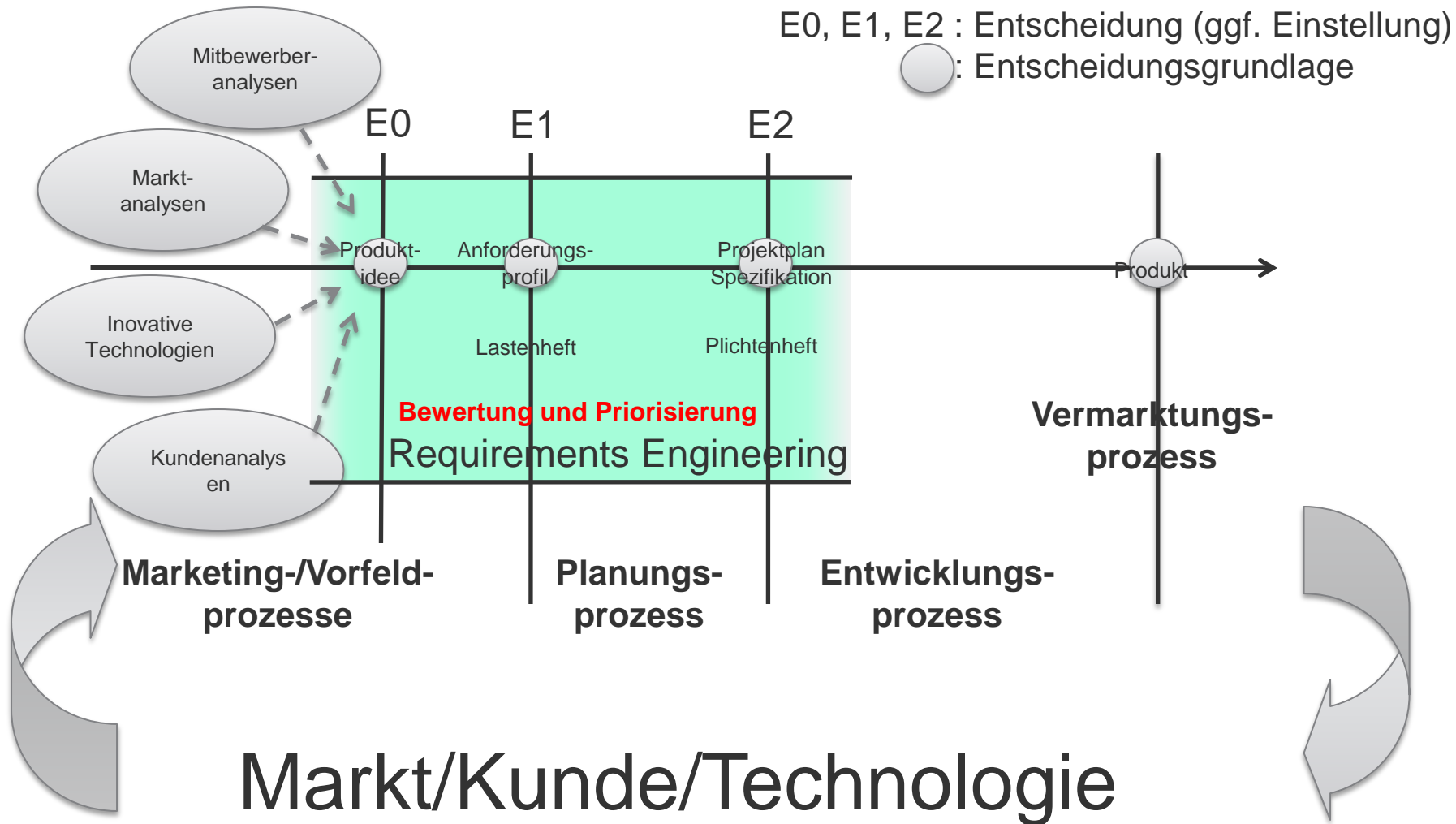
Vollständigkeit:

- Sind alle wichtigen Aspekte des zu entwickelnden Systems beschrieben?
→ wichtig dabei sind die Geschäftserfordernisse, totale Vollständigkeit ist nicht erstrebenswert und kann kaum erzielt werden

Widerspruchsfreiheit:

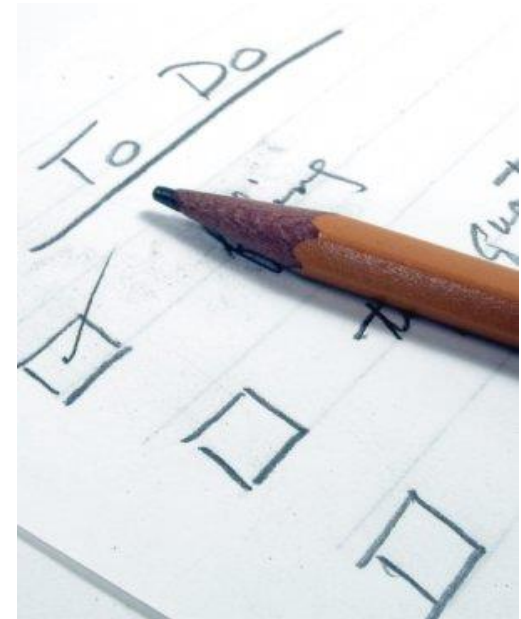
- Sind alle Widersprüche zwischen einzelnen Anforderungen ausgeräumt?

Requirements Engineering ist eine zentrale Aktivität im Produktentstehungsprozess



10 Ratschläge zur erfolgreichen Anforderungsspezifikation

1. Definiere eine Standard-Gliederung für das Dokument
2. Gestalte das Dokument änderungsfreundlich
3. Gib jeder Anforderung eine eindeutige Bezeichnung
4. Definiere eine Vorgehensweise bei **Änderung** der Anforderungen
5. Definiere Standard-Vorlagen für die Beschreibung der Anforderungen
6. Verwende eine einfache, konsistente und knappe Sprache
7. Organisiere formelle **Begutachtungen** der Anforderungen
8. Definiere Checklisten für die Begutachtung von Anforderungen
9. Benutze Checklisten zur Anforderungsanalyse
10. Rechne von vorneherein mit Konflikten und plane die Konfliktauflösung



[SoSa97]

Geht es nicht auch ohne Requirements Engineering?



- Kleines hochqualifiziertes Team mit überschaubarer Aufgabenstellung
- Kontinuierliche Einbindung des Kunden in die Entwicklung
- Keine Vorgaben an den Entwicklungsprozess durch Standards/Normen (also z.B. keine sicherheitskritischen Systeme)
- Massiver Einsatz von Prototyping
- Dennoch, auch hier Requirements Engineering „light“ etwa durch **Story Cards** und **Story Boards**
 - Ein **Story Board** ist eine konzeptuelle Beschreibung einer Systemfunktionalität für ein bestimmtes Szenario, inklusive der notwendigen Interaktionen zwischen dem Benutzer und dem System. Ein Story Board „erzählt eine Geschichte“.
 - Eine **User Story** berichtet, wie das System dazu beiträgt, das Problem eines Benutzers zu lösen oder einen Geschäftsprozess zu unterstützen.
 - Jede User Story wird auf eine **Story Card** geschrieben und stellt einen zusammengehörigen Funktionsblock aus Benutzersicht dar.

Beispiel einer Story Card

StoryTag: DocBook To HTML Release: Book Priority: 1
Author: Joanne on: 2/21/02 Accepted: 3/17/02

Description: Make the DocBook files readable and printable.

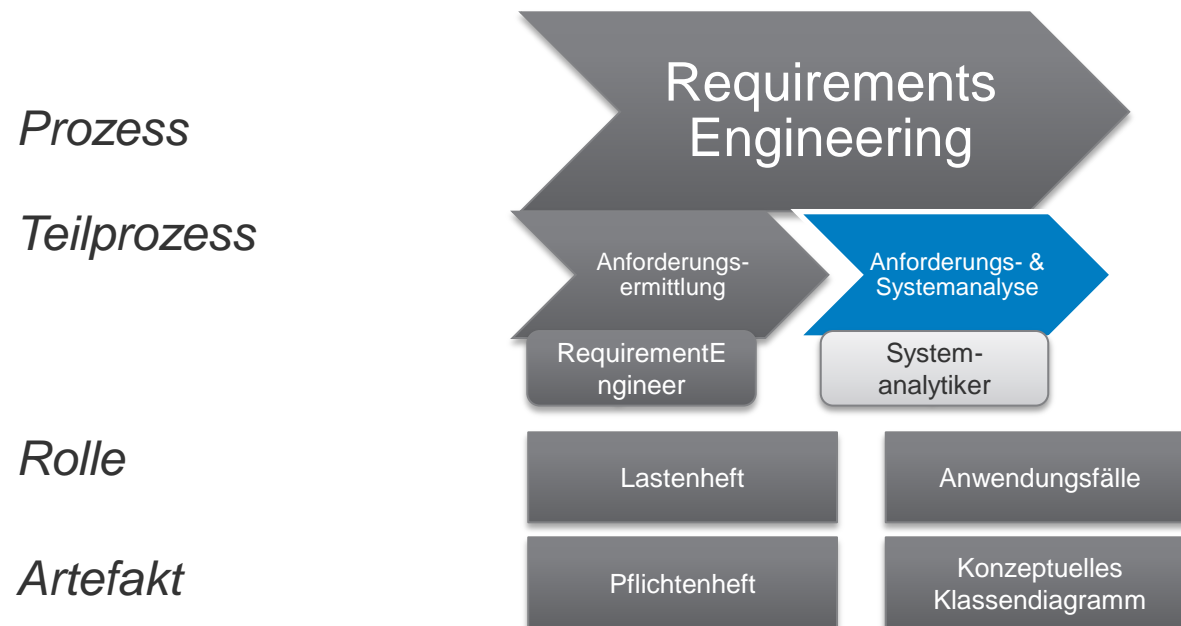
Considerations: HTML has some drawbacks: Estimate: 4.1
• Printed version is not production quality.
• Footnotes can't appear at end of page.

Who	Task	Est.	Done
Rob	Simple tags: <chapter>, <title>, <para>	1	2/24
Rob	Asymmetrical tags: <attribution>	1	3/3
Rob	Contextually related tags: <title>	1	3/11
Rob	Stateful output: <footnote>	1	3/14
Joanne	Acceptance Test: Print the first chapter	•1	3/17

User-Centered Design und Personas

2 Requirements Engineering

- Überblick und Motivation
- Anforderungsermittlung
- Anforderungs- und Systemanalyse mit UML
 - Einführung UML
 - Modellierung von Szenarien mit UML Anwendungsfalldiagrammen
 - Konzeptuelle Datenmodellierung mit UML Klassen- und Objektdiagrammen



- **Ziel:** Nutzungs- und Schnittstellenverhalten beschreiben
 - Die Interaktion zwischen systemexternen Akteuren und dem System ins Zentrum der Betrachtungen stellen
 - Jede Interaktionssequenz wird durch ein Szenario beschrieben; man erhält eine nutzerorientierte Spezifikation der Funktionalität des Systems

- **Szenario (*scenario*)**
 - Eine geordnete Menge von Interaktionen zwischen Partnern, in der Regel zwischen einem System und einer Menge systemexterner Akteure.
 - Eine Szenario kann entweder eine konkrete Interaktionsfolge (konkretes Szenario oder **Beispielszenario**) oder eine Menge möglicher Interaktionen (abstraktes Szenario oder **Typszenario**) sein.

▪ Anwendungsfall (*use case*)

- Ein Typszenario oder auch abstraktes Szenario
- Eine durch genau einen Akteur angestoßene Folge von Systemereignissen (Interaktionen zwischen System und externen Akteuren), welche für den Akteur ein Ergebnis produziert und an welchem weitere Akteure teilnehmen können
 - *Aus Jacobson et al. 1992; wird in UML so verwendet.*
 - *In diesem Kontext bezeichnet ein Szenario ausschließlich ein Beispielszenario.*

▪ Akteur (*actor*)

- Ein **Mensch**, der Ziele hat und zu deren Erreichung handelt oder ein Element des Anwendungsbereichs, das zur Erreichung bestimmter Ziele dient und hierzu handeln und/oder Informationen verarbeiten kann.
 - Max Müller
- Eine **Rolle**, welche ein externes System oder ein Mensch gegenüber dem System einnehmen kann.
 - Benutzer, Administrator, Redakteur, Dozent, ...

Anmerkung

- Häufig zur Beschreibung konkreter Beispielszenarien

Bewertung

- Flexibel und ausdrucksmächtig
- Von Anwendungsexperten lesbar und schreibbar
- Unpräzise
- Missverständnisse leicht möglich
- Fehler leicht übersehbar

Beispiel

Max Müller startet CQ.edit, öffnet das IEEE-Projekt und will die bestehende Konfigurationsdatei ieee-swa bearbeiten. Zu diesem Zweck öffnet er die Datei ieee-swa in Design-Sicht, fügt vordefinierte Module aus der ConQAT Bibliothek hinzu und speichert die Datei ab. Dabei wird die XML-Quelle der Konfigurationsdatei in der XML-Sicht im CQ.edit aktualisiert. Anschließend schließt Max Müller das Projekt und den Editor.

- **Name:** Config.-Datei Bearbeiten
- **Beschreibung:** Eine bereits vorhandene Konfigurationsdatei bearbeiten
- **Akteure:** ConQAT User
- **Auslöser:** Ein ConQAT User öffnet eine bereits angelegte Analysenkonfigurationsdatei eines bestehenden ConQAT-Projektes, die er im Design-Modus um neue Prozessoren/Prozessorblöcke erweitern möchte
- **Vorbedingung:** Config.-Datei ist angelegt
- **Nachbedingung:** Config.-Datei ist bearbeitet und gespeichert
- **Ablaufschritte:**
 1. Zu bearbeitende Config.-Datei in ConQAT.edit öffnen
 2. Datei durch Hinzufügen/Entfernen von Modulen bearbeiten
 3. In- und Output der Module überprüfen und bei Bedarf neu definieren
 4. Config.Datei speichern und schließen
- **Alternative Ablaufschritte:**
 - Speichern ist nicht möglich (Zugriff verweigert)
 - Config.-Datei umbenennen und speichern
 - ...

- **Akteur:** ConQAT User
- **Auslöser:** Ein ConQAT User öffnet eine bereits angelegte Analysenkonfigurationsdatei eines bestehenden ConQAT-Projektes, die er im Design-Modus um neue Prozessoren/Prozessorblöcke erweitern möchte
- **Normalverlauf:**
 1. Reiter für ConQAT Bibliothek anzeigen lassen
 2. Modul eines hinzuzufügenden Prozessors/Prozessorblocks auswählen
 3. Gewünschter Prozessor/Prozessorblock per drag and drop in die Analysenkonfigurationsdatei einfügen
 4. In- und Output für den eingefügten Prozessor/Prozessorblock definieren
 5. Wenn mehrere Prozessoren/Prozessorblöcke einzufügen sind, für die weiteren Prozessoren/Prozessorblöcke Schritte 2., 3. und 4. durchführen
 6. Projekt speichern und Vorgang abschließen
- **Alternative Abläufe:**
 - 3.1. Hinzugefügter Prozessor/Prozessorblock ist nicht erwünscht: Prozessor/Prozessorblock entfernen und mit Schritt 5. fortfahren
 - ...

Anmerkung

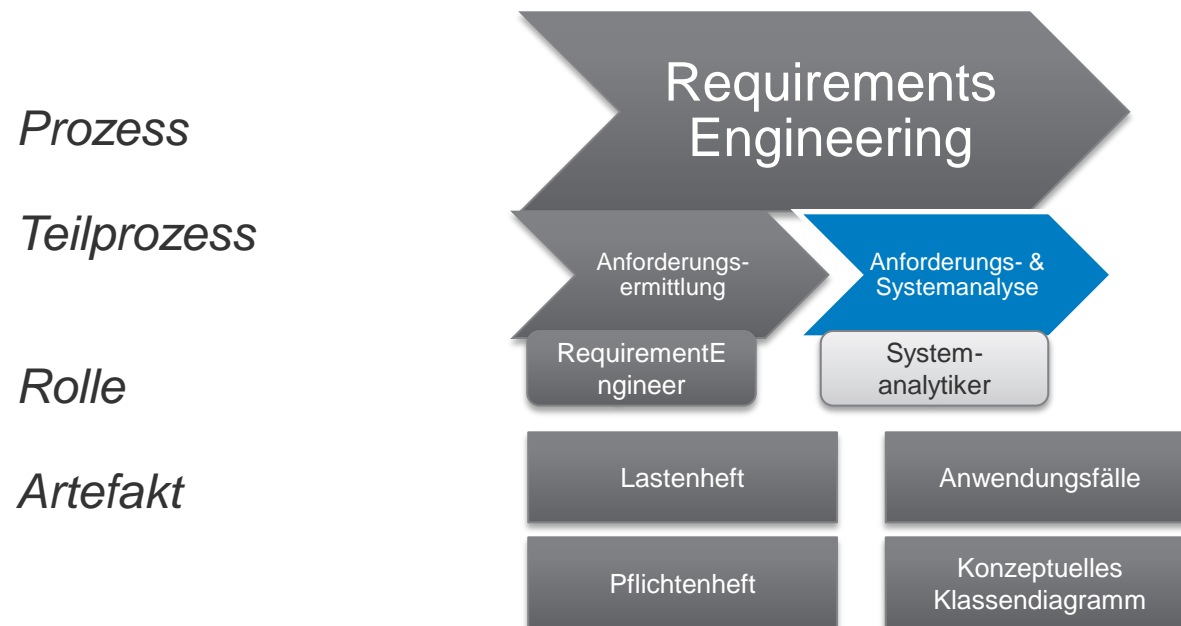
- Häufig zur Beschreibung von Anwendungsfällen
- Angabe des Hauptakteurs und des Auslösers für die Ausführung des Szenarien
- Ablauf der Schritte im Normalfall
- Kennzeichnung der Reihenfolge durch Nummerierung der Schritte oder Interaktionsfolgen
- Angabe möglicher Ausnahmefälle

Bewertung

- + Von Anwendungsexperten lesbar und schreibbar
- + Präziser als freier Text, weniger Auslassungen und Fehler
- Oft noch zu unpräzise und fehlerträchtig
- Zusammenhänge mit anderen Szenarien/Anwendungsfällen werden nicht erfasst

2 Requirements Engineering

- Überblick und Motivation
- Anforderungsermittlung
- Anforderungs- und Systemanalyse mit UML
 - Einführung UML
 - Modellierung von Szenarien mit UML Anwendungsfalldiagrammen
 - Konzeptuelle Datenmodellierung mit UML Klassen- und Objektdiagrammen



1997 wird UML 1.1 als Standard-Modellierungssprache der OMG (Object Management Group; ein Industriekonsortium) angenommen.

UML setzt sich rasch als Standardsprache für die grafische Modellierung von Anforderungen und Entwürfen durch.

- Gleichzeitig erfährt UML eine Reihe von Revisionen:
 - 1997: UML 1.0
 - 1997: UML 1.1
 - 1998: UML 1.2
 - 1999: UML 1.3
 - 2001: UML 1.4
 - 2003: UML 1.5
 - 2004: UML 2.0 mit teilweise erheblichen Änderungen
 - 2007: UML 2.1
 - 2010: UML 2.3
 - 2011: UML 2.4

Heute (2012) in Gebrauch: zunehmend 2.0+ teilweise noch 1.4

Die 3 UML “Amigos”:

Ivar Jacobson, James Rumbaugh, Grady Booch



Erfinder der
Anwendungsfall-,
Sequenz- und
Kollaborationsdiagramm
e zur Modellierung von
Schaltanlagen bei
Ericsson (1967)



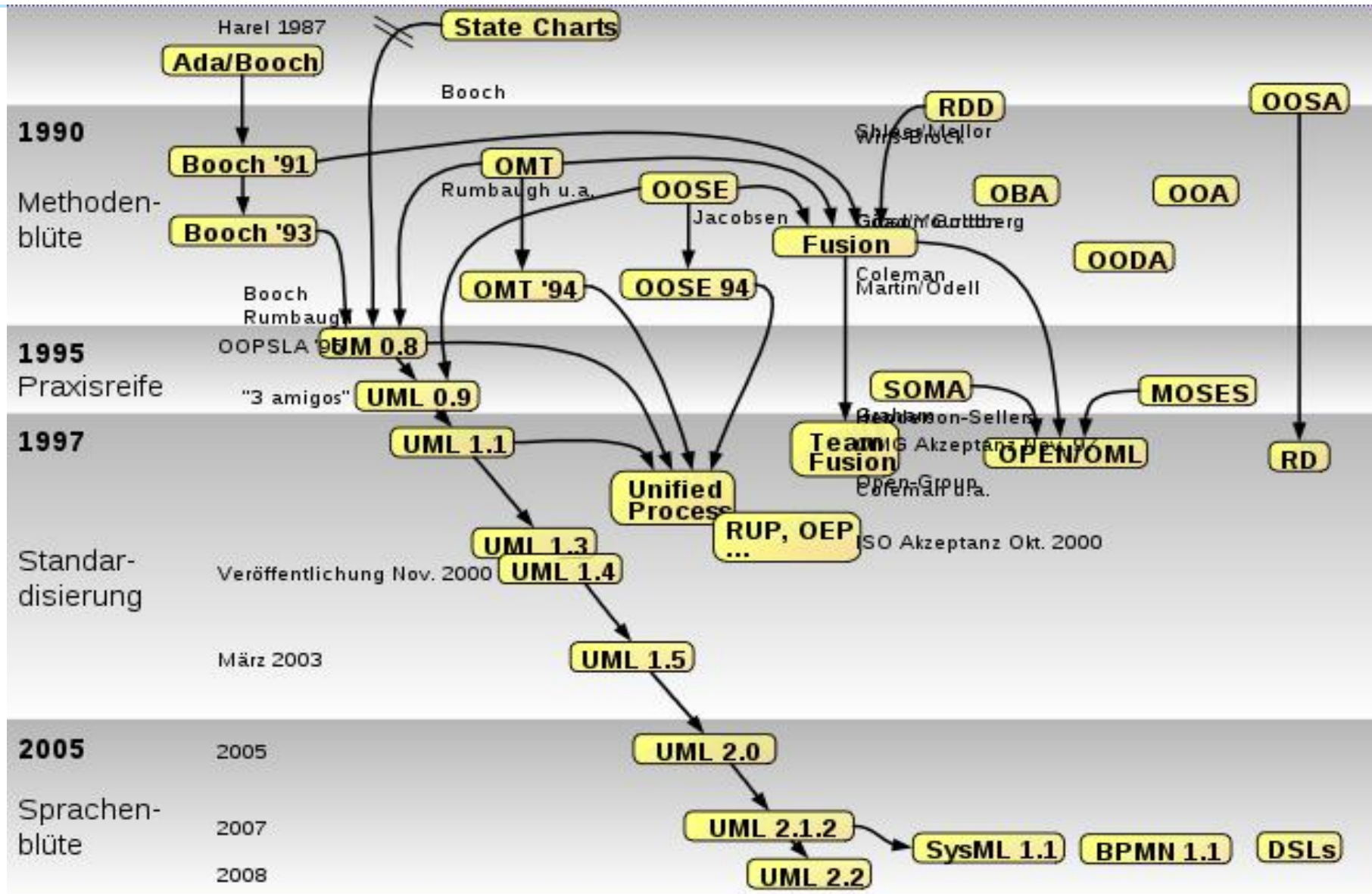
Erfinder der OMT
Notation (Object
Modeling Technique)
Führte Vererbung in
die E/R Modellierung
ein, 1991



Einer der ersten
“objektorientierten”
Modellierer
Entwickler der Booch-
Methode (Objekte als
Wolken, 1991)

[Wi09b]

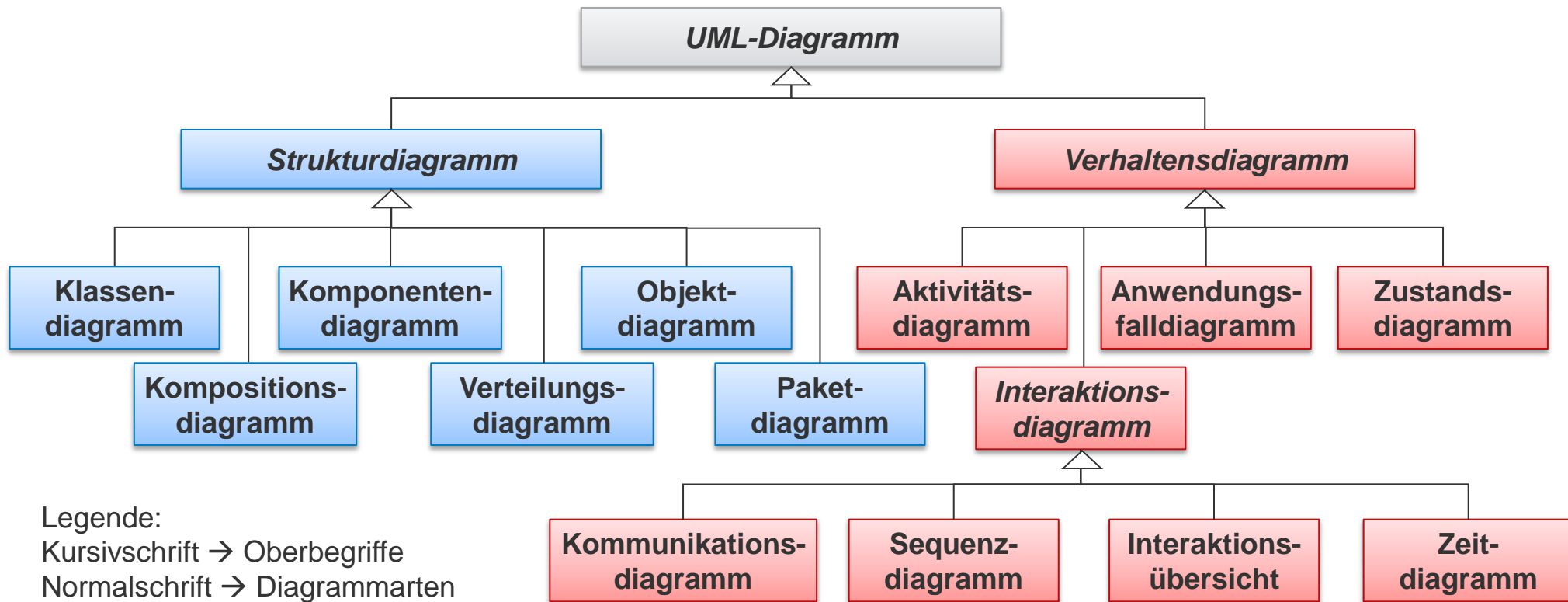
UML: Historie



- UML besteht aus einer Sammlung vorwiegend grafischer Sprachen zur Erstellung von Anforderungs- und Entwurfsmodellen aus verschiedenen Perspektiven
- Dementsprechend besteht eine UML-Spezifikation aus einer Sammlung sich ergänzender und teilweise überlappender Modelle
- Im Zentrum steht ein Klassenmodell, das den strukturellen Aufbau eines Systems spezifiziert
- Nach Bedarf beschreiben weitere Modelle zusätzliche Systemsichten

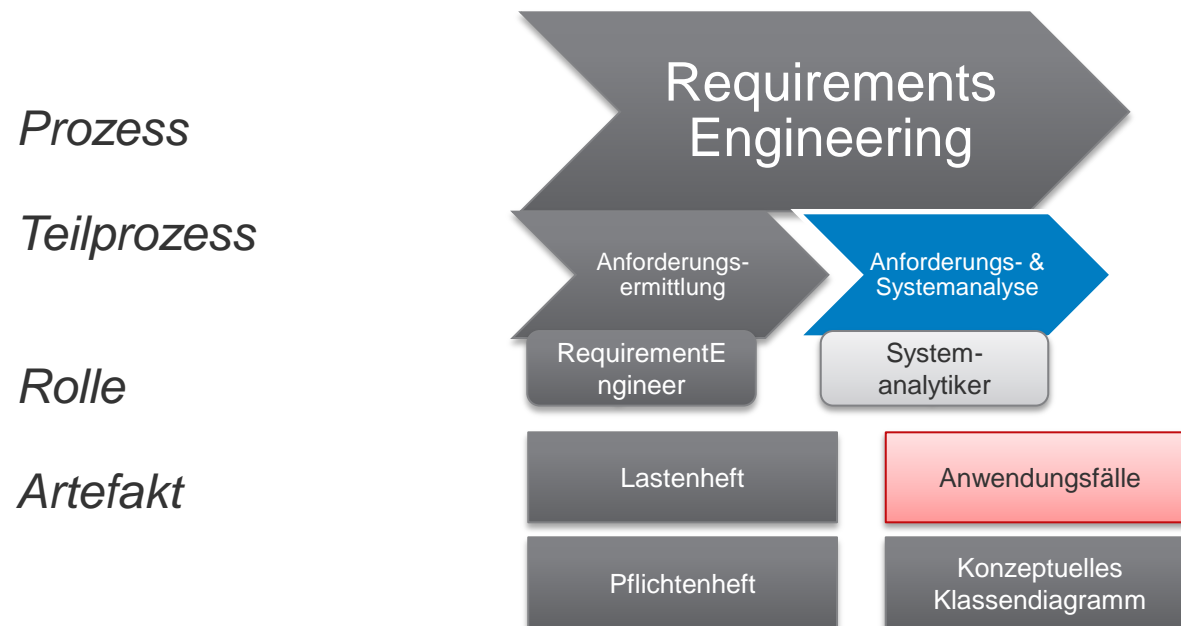
- Insgesamt acht verschiedene Sichten, gruppiert in zwei Kategorien
- **Struktursicht**
 - Statische Struktur (Daten-, Klassen- und Objektmodelle)
 - Struktur und Zusammenarbeit von Komponenten
 - Portionierung von UML-Modellen in Pakete und Subsysteme
 - Physische Struktur: Artefakte, Knoten, Verteilung
- **Verhaltenssicht**
 - Interaktion externer Akteure mit einem System
 - Zeitlich-dynamisches Verhalten von Systemen
 - Interaktion ausgewählter Objekte untereinander
 - Aktivitäten und deren Ablauf

Zur Darstellung der acht Sichten bietet UML 2.0 zwölf Diagrammarten an:



2 Requirements Engineering

- Überblick und Motivation
- Anforderungsermittlung
- Anforderungs- und Systemanalyse mit UML
 - Einführung UML
 - Modellierung von Szenarien mit UML Anwendungsfalldiagrammen
 - Konzeptuelle Datenmodellierung mit UML Klassen- und Objektdiagrammen



Anwendungsfall-Diagramme

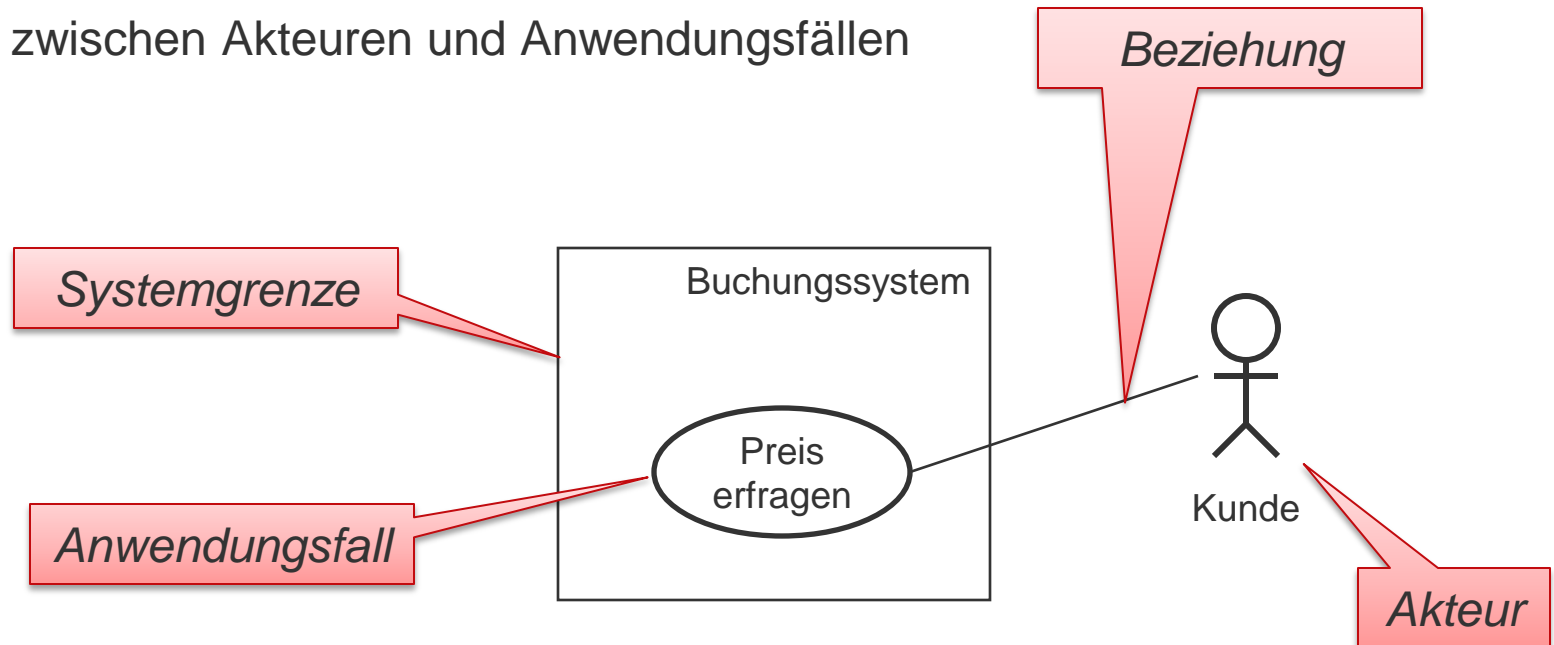
Ein Anwendungsfall-Diagramm zeigt die Beziehungen zwischen Akteuren und Anwendungsfällen in einem System

Englischer Begriff: ***use case diagram***

Ein Anwendungsfall-Diagramm für ein System besteht aus

- der Systemgrenze
- Akteuren
- Anwendungsfällen
- Beziehungen zwischen Akteuren und Anwendungsfällen

Notation:



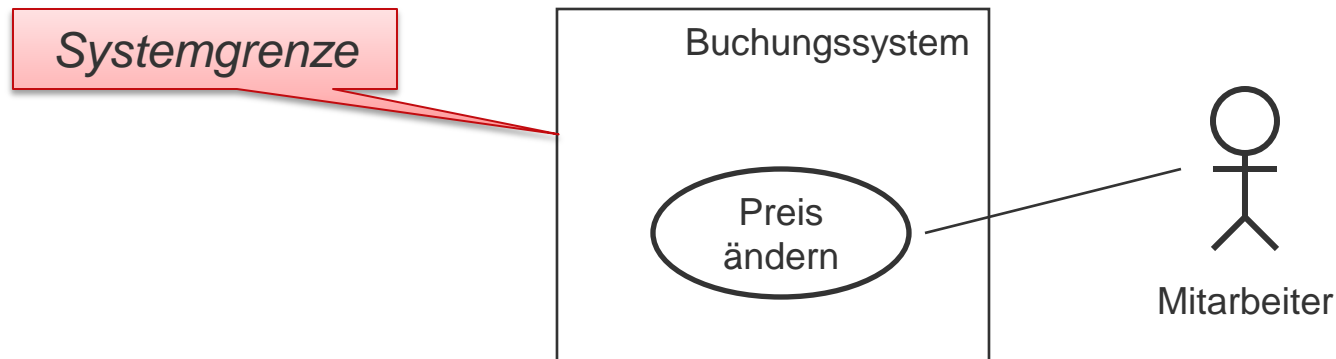
Die Anwendungsfallsicht beschreibt die Funktionalität eines Systems aus einer Perspektive von außerhalb, also aus Benutzersicht.

Es interessieren nur die Interaktionen zwischen Akteuren und dem beschriebenen System, nicht aber, was innerhalb des Systems geschieht.

Diese Grenze wird durch die Systemgrenze verdeutlicht.

Englischer Begriff: *system boundary*

Notation:



Beobachtung: In weiteren Anwendungsfällen kann ein System wieder als Akteur erscheinen, z.B. ein Buchungssystem (eines Reisebüros) in Bezug auf ein anderes System (Flugreservierungssystem).

Ein Akteur ist eine Abstraktion für ein Objekt außerhalb des Systems, das mit dem System interagiert, z.B. ein Benutzer. Möglich dabei sind:

- mehrere Rollen für einen Benutzer
- mehrere Benutzer mit der gleichen Rolle
- Systeme als Akteure

engl.: **actor**

Eigenschaften eines Akteurs sind:

- Name: Substantiv
- Beschreibung:
 - textuelle Beschreibung seiner Rolle, Tätigkeiten, Fähigkeiten, Protokolle, ...
 - alle Beschreibungen zusammen bilden den Akteur-Katalog

Notation:



Ein Anwendungsfall ist die Spezifikation einer Folge von Interaktionen zwischen einem Akteur und dem zu untersuchenden Computersystem

Englischer Begriff: ***use case***

Anwendungsfälle beschreiben eine für den Benutzer sichtbare Funktionalität

Eine Instanz eines Anwendungsfalls (Instanziierung) ist die Ausführung der beschriebenen Interaktionen

Ein Anwendungsfall definiert:

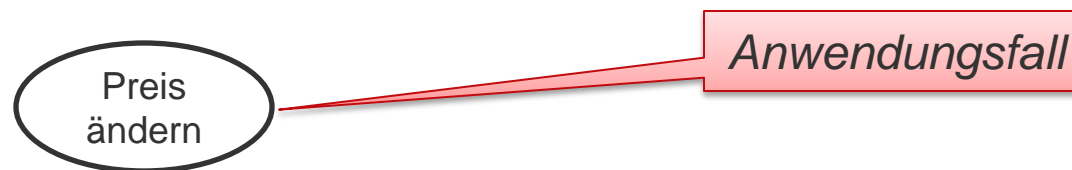
- Name: Verb (o. Substantiv), Name des Vorgangs
- Priorität: Priorität im Entwicklungsprozess
- Beschreibung:
 - textuelle Beschreibung der Interaktion zwischen Benutzer und System, unter Bezugnahme auf das Lastenheft (z.B. /LF70/)
 - Auflistung der einzelnen Schritte unter Nutzung der Begriffe des → Glossars
 - alle Beschreibungen zusammen bilden den Anwendungsfall-Katalog

Beschreibung eines Anwendungsfalls im Anwendungsfall-Katalog:

Beispiel: Anwendungsfall „Preis ändern“:

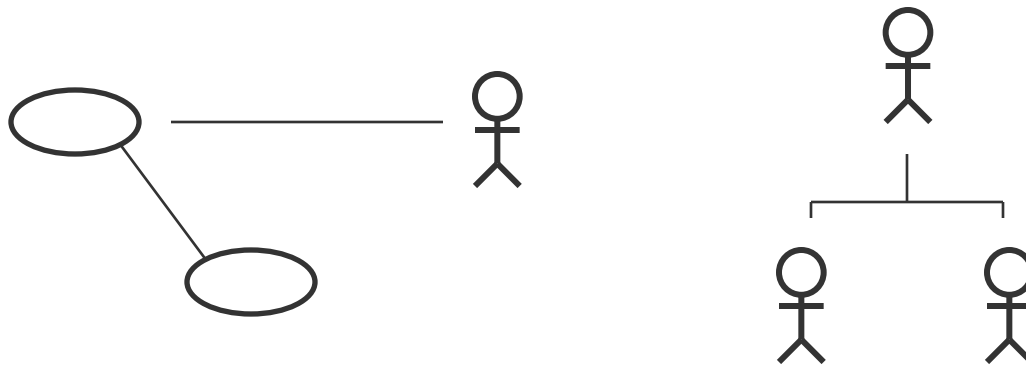
1. Der Mitarbeiter sucht das gewünschte Produkt durch Eingabe der Produktnummer
2. Das System zeigt alle Produktdaten an
3. Der Mitarbeiter trägt den neuen Gesamtpreis ein
4. Das System prüft, ob der Preis innerhalb der zulässigen Grenzen liegt
5. Das System prüft, ob aktuelle Aufträge betroffen sind und fragt, ob sie den alten oder neuen Preis erhalten sollen. Der Mitarbeiter kann die Änderung hier rückgängig machen
6. Der neue Preis wird übernommen

Notation:



Beziehungen zwischen Anwendungsfällen und Akteuren

- werden durch gerichtete oder ungerichtete Pfeile mit unterschiedlichen Pfeilspitzen notiert,
- können durch Stereotypen annotiert werden;
Schreibweise: « stereotyp »,
- können zwischen zwei Anwendungsfällen bestehen,
- können zwischen zwei Akteuren bestehen oder
- können zwischen einem Anwendungsfall und einem Akteur bestehen

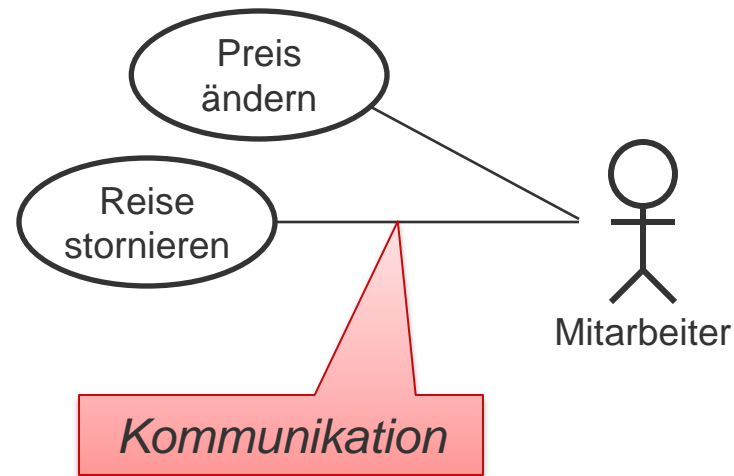


Die Kommunikation zwischen einem Akteur und dem System wird durch eine ungerichtete Kante zwischen Akteur und Anwendungsfall dargestellt.

Ein Akteur kann mit mehreren Anwendungsfällen assoziiert sein und umgekehrt.

Englischer Begriff: ***actor-use case communication***

Notation:



Mögliche Bedeutungen sind: „A ist beteiligt in/an B“, „A löst B aus“, „A kommuniziert mit B“, „A nutzt B“, ...

- Wähle einen **vertikalen Ausschnitt** aus dem System (ein Szenario)
 - Diskutiere mit dem Benutzer dessen bevorzugte Art mit dem System zu interagieren
- Wähle einen **horizontalen Ausschnitt** (viele Szenarien), um den Umfang der Systemfunktionalitäten zu beschreiben
 - Diskutiere den Umfang mit den Benutzern
- Verwende **graphische Prototypen** (*mock-ups*)
- Finde heraus, was die Benutzer tun
 - durch **Beobachtung** (gut)
 - durch Fragebögen (nicht so gut)

Schritte bei der Beschreibung eines Anwendungsfalls



1. Benenne den Anwendungsfall
Bsp.: ReportEmergency
2. Finde und generalisiere die Akteure
Bsp.: Field Officer (Bob & Alice), Dispatcher (John)
3. Beschreibe den Ablauf der Ereignisse im Anwendungsfall natürlichsprachlich

- dienen dazu die Komplexität des Anwendungsfalldiagramms zu reduzieren, indem sie
 - umfangreiche Anwendungsfälle in kleinere zerlegen,
 - alternative Abläufe voneinander trennen und
 - Spezialisierung von Anwendungsfällen erlauben.

- existieren in verschiedenen Arten:
 - Einschluss (*include*)
 - Erweiterung (*extends*)
 - Generalisierung

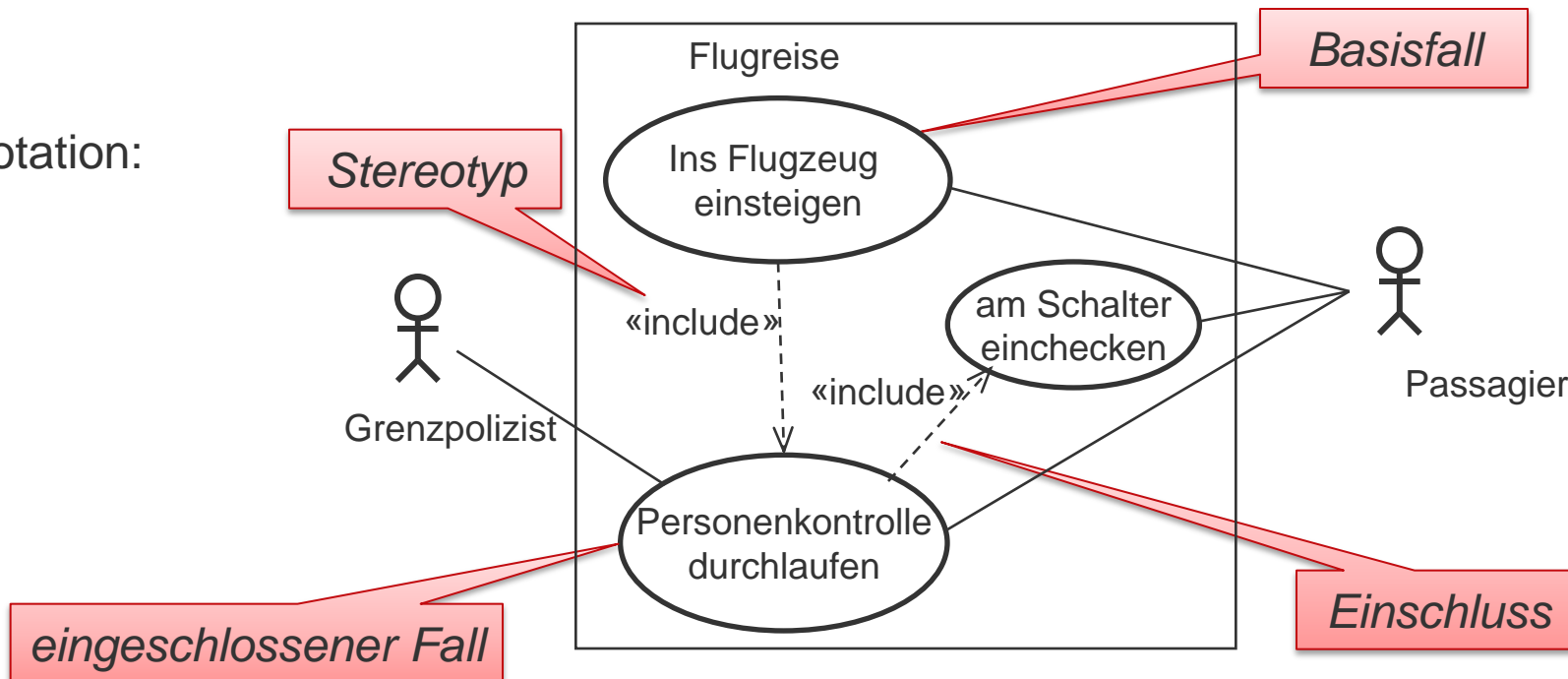
➔ Erweiterung und Generalisierung werden häufig von Kunden nicht verstanden, sind deshalb also mit Vorsicht zu verwenden.

Einschluss von Anwendungsfällen

Einschluss bedeutet das Aufnehmen bestimmter Aktionsfolgen in die eines Basisfalls
Der eingeschlossene Fall kann unabhängig vom Basisfall und von verschiedenen Basisfällen genutzt werden.

Der Einschluss ist nicht optional, d.h. der Basisfall kann nicht ohne den eingeschlossenen Fall genutzt werden

Notation:



- Die folgenden Fragen erweisen sich als hilfreich:
 - Welche Aufgaben soll das System ausführen?
 - Welche Daten werden von den Benutzern mit dem System erzeugt, gespeichert, verändert oder gelöscht?
 - Über welche externen Ereignisse muss das System informiert sein?
 - Über welche Ereignisse muss der Benutzer des Systems informiert werden?
- Die Fragen allein jedoch sind nicht ausreichend, Beobachtungen von Benutzern an bestehenden Altsystemen sind wichtig:
 - Befrage die Benutzer und nicht nur den Auftraggeber
 - Erwarte Widerstand und versuche diesen zu überwinden

- Jedes Typszenario / jeden Anwendungsfall beschreiben
 - Mit strukturiertem Text oder noch präziser mit Statecharts bzw. Aktivitätsdiagrammen
 - Dabei sowohl die Normalabläufe als auch die Ausnahmeabläufe beschreiben
 - Anwendungsfall-Diagramm als Übersicht modellieren

- Bei vielen, feingranular beschriebenen Typszenarien/Anwendungsfällen:
 - Übersicht schaffen durch Zusammenfassen zusammengehöriger Szenarien/Anwendungsfälle zu einem übergeordneten, vergrößerten Szenario/Anwendungsfall

Anwendungsfall: Geld Abheben am Automaten mit EC-Karte

Initiierender Akteur:

- Bankkunde

Vorbedingungen:

- Bankkunde besitzt ein Konto bei der Bank
- Bankkunde hat eine EC Karte und zugehörige PIN

Nachbedingungen:

- Bankkunde hat geforderten Betrag erhalten oder
- Bankkunde erhält Erläuterung von Geldautomaten, warum kein Geld ausgegeben wurde

Aktionen des Kunden

1. Der Kunde gibt die EC-Karte in den Automaten.
3. Der Kunde gibt seine PIN ein.
5. Der Kunde wählt das zu belastende Konto.
7. Der Kunde gibt den abzuhebenden Betrag ein.

Aktionen des Systems

2. Der Automat bittet um die Eingabe der PIN.
4. Der Automat bietet eine Auswahl der verschiedenen zum Kunden gehörigen Konten, falls mehrere existieren.
6. Der Automat prüft die Verfügbarkeit von Geld auf dem Konto.
8. Der Automat bucht den Betrag ab und gibt das Geld aus.

Aktionen des Kunden

1. Der Kunde gibt die EC-Karte in den Automaten. **[ungültige Karte]**
3. Der Kunde gibt seine PIN ein. **[ungültige PIN]**
5. Der Kunde wählt das zu belastende Konto.
7. Der Kunde gibt den abzuhebenden Betrag ein. **[Betrag über Limit]**

[ungültige Karte]: Der Geldautomat gibt die Karte wieder aus und bricht den Vorgang ab.

[ungültige PIN]: Der Geldautomat meldet einen Fehler und bietet eine Erneute Eingabe an. Nach zwei fehlerhaften Eingaben kündigt der Automat das Einbehalten der Karte bei einer weiteren Fehleingabe an. Bei der dritten fehlerhaften Eingabe zieht der Automat die Karte ein und bricht den Vorgang ab.

[Betrag über Limit]: Der Geldautomat meldet einen Fehler sowie das Limit und bietet eine erneute Betragseingabe sowie den Abbruch des Vorgangs an.

- Name
 - Verwendung eines Verbs im Namen des Anwendungsfalls.
 - Der Name sollte erklären, was der Benutzer zu erreichen sucht.
 - Beispiele: *Vereinbare Meeting*, *Schlage anderes Datum vor*
- Länge
 - Die Beschreibung eines Anwendungsfalls soll 1-2 Seiten nicht überschreiten. Falls dies nicht möglich ist, empfiehlt sich die Verwendung von Einschluss-Beziehungen.
 - Die Beschreibung sollte die komplette Folge der Interaktionen enthalten.
- Ablauf:
 - Verwendung des Aktivs. Die Sätze sollten stets mit *Der Nutzer* oder *Das System* beginnen.
 - Die kausale Abfolge der Schritte muss erkennbar sein.
 - Der komplette Ablauf sollte beschrieben werden, nicht nur der reguläre.
 - Die Systemgrenzen sowie externe Komponenten müssen erkennbar sein.
 - Wichtige Begriffe müssen im Glossar erläutert werden.

Beispiel für einen schlecht beschriebenen Anwendungsfall



Der Fahrer erreicht die Einfahrschranke, er erhält ein Ticket, die Schranke wird geöffnet und der Fahrer fährt hindurch.

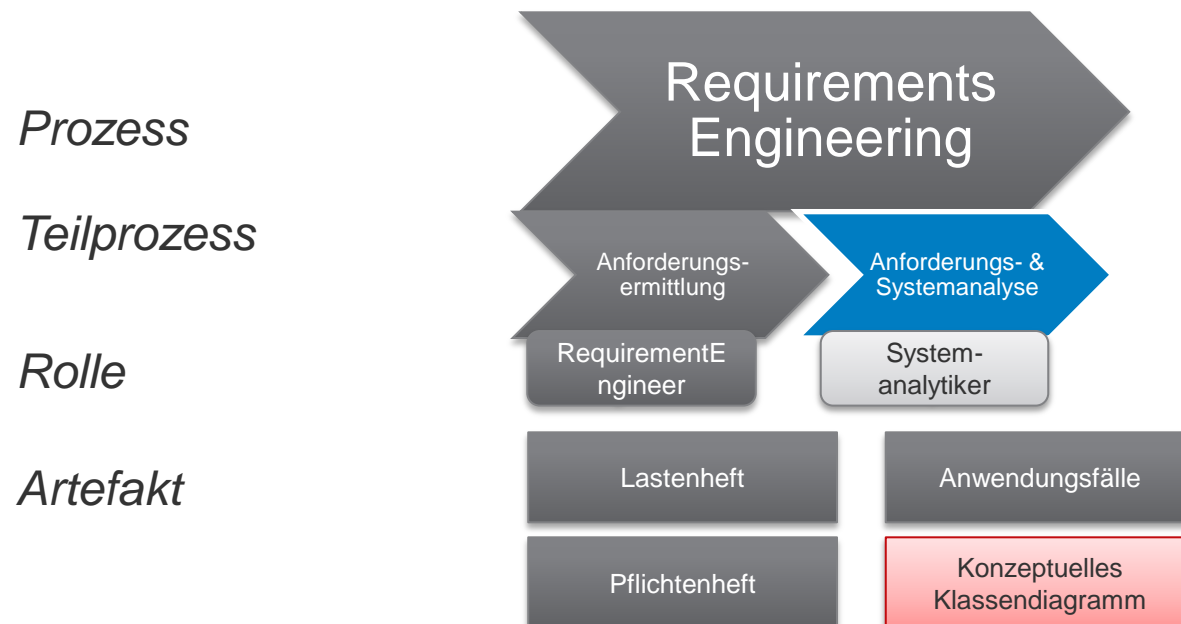
Oben stehender Anwendungsfall ist schlecht beschrieben, denn

- er enthält keine Akteure,
- lässt offen, wodurch die Aktion der Ticketausgabe ausgelöst wird,
- stellt nicht dar, wer die Schranke öffnet (Passiv) und
- ist nicht vollständig, da z.B. die Bezahlung ausgelassen wird.

- Szenarien
 - sind gut geeignet, um mit dem Kunden über Verhalten zu sprechen
 - können verschieden formuliert werden: As-Is, visionary, evaluation and training
- Anwendungsfälle
 - stellen eine Abstraktion konkreter Szenarien dar
 - unterstützen den Übergang von funktionalen Anforderungen zu Objekten

2 Requirements Engineering

- Überblick und Motivation
- Anforderungsermittlung
- Anforderungs- und Systemanalyse mit UML
 - Einführung UML
 - Modellierung von Szenarien mit UML Anwendungsfalldiagrammen
 - Konzeptuelle Datenmodellierung mit UML Klassen- und Objektdiagrammen



Ziel: Herausarbeiten der wichtigen und notwendigen Abstraktionen

Schritte bei der Objektmodellierung:

1. Identifikation der Klassen
2. Identifikation von Attributen
3. Identifikation von Methoden
4. Identifikation von Beziehungen zwischen Klassen
5. Festlegung von Rollen und Kardinalitäten
6. Einfügen von Vererbungsbeziehungen

Dabei ist die Reihenfolge der oben genannten Schritte eher zweitrangig und darf als Heuristik verstanden sein.

Was passiert, wenn wir die falschen Abstraktionen wählen?

➔ Wir müssen das Modell „noch einmal anfassen“ und überarbeiten.

- Ein **Objekt** stellt eine einzelne Instanz (ein Ding) dar
Diese Vorlesung beschäftigt sich mit Objektmodellierung.
- Eine **Klasse** beschreibt eine Gruppe von Objekten mit gleichartigen Eigenschaften
Vorlesungen haben ein Thema, einen Titel und einen Dozenten.

Für beide Arten von Konzepten gibt es geeignete und standardisierte graphische Notationen:

- Ein **Klassendiagramm** stellt eine Menge von Klassen sowie deren Beziehungen untereinander dar. Es eignet sich zur Beschreibung einer Taxonomie oder eines Schemas.
- Ein **Objektdiagramm** stellt eine Menge von konkreten Objekten sowie deren Beziehungen zueinander dar. Es eignet sich zur Diskussion von Szenarien und Beispielen.

Ziel

Konzeptuelle Perspektive

- repräsentiert die Konzepte, die zu den Klassen gehören
- bietet Sprachunabhängigkeit

Hilfsmittel

- Klassendiagramm
 - Klasse
 - Assoziation
 - ...
- Objektdiagramm
 - Objekt (als Instanz einer Klasse)
 - Referenz
 - ...

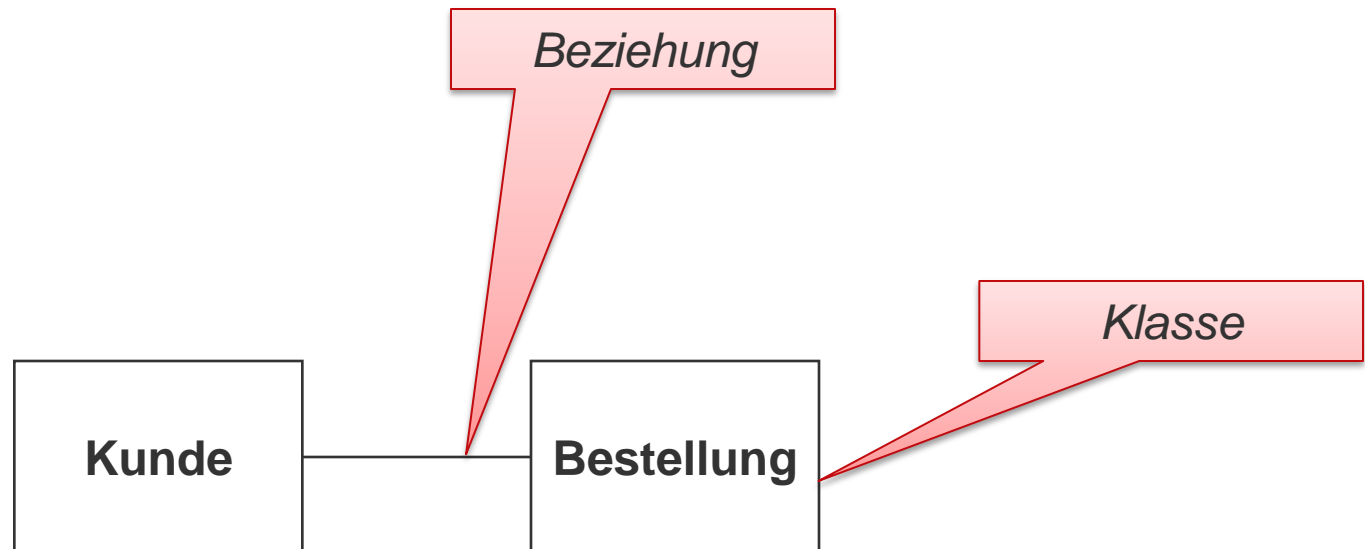
ACHTUNG: Es werden in diesem Kapitel nicht alle Konstrukte von UML Klassendiagrammen eingeführt!

Ein **Klassendiagramm** beschreibt einen strukturellen Aspekt des zu entwickelnden Systems in Form von Klassen, Interfaces und deren Beziehungen zueinander.

engl.: *class diagram*

Klassendiagramme können enthalten:

- Klassen
- Assoziationen
- ...



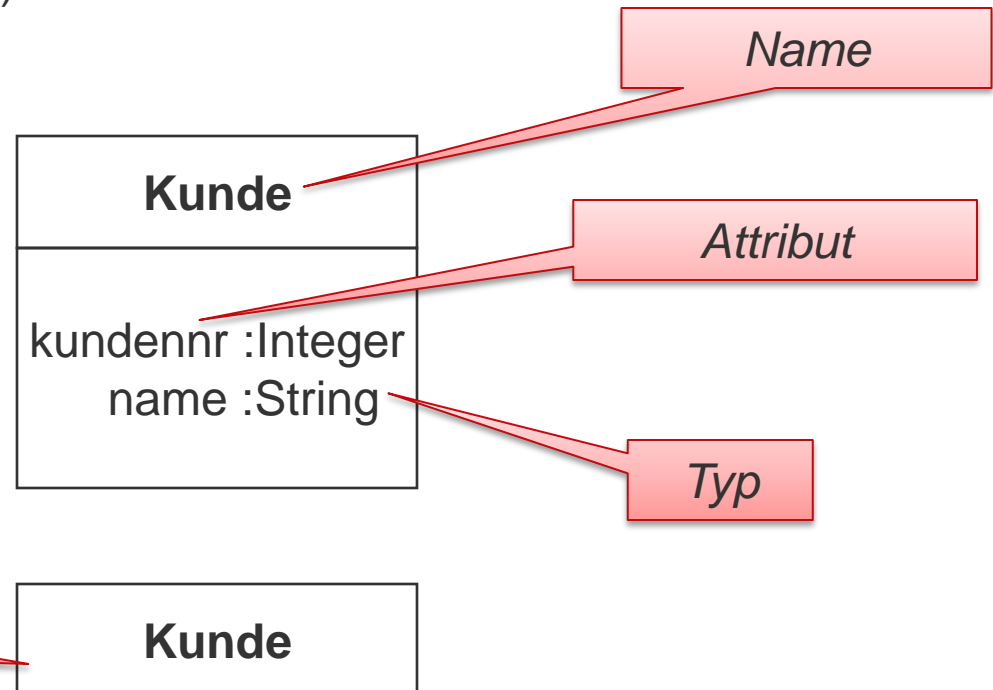
Klasse im konzeptuellen Klassendiagramm

Eine **Klasse** ist die Beschreibung einer Menge von Objekten, die dieselben Attribute, Operationen, Methoden, Beziehungen und dasselbe Verhalten haben.

Eine Klasse besitzt

- Name
- Beschreibung (Verantwortung, Aufgabe)
- Attribute
 - Name
 - Beschreibung, evtl. Typ
- (Methoden)

engl.: **class**

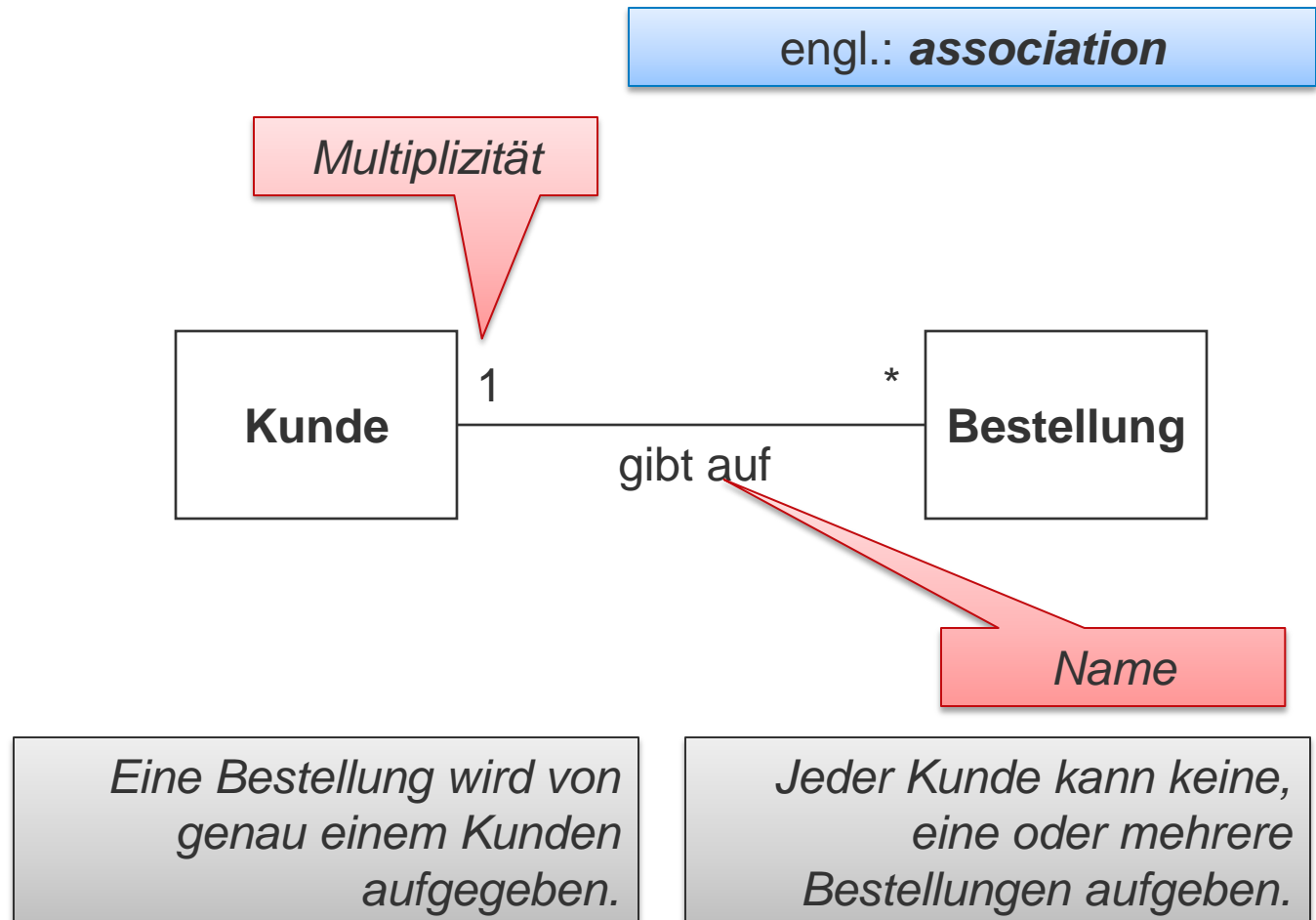


*alternative, kompakte
Darstellung ohne Details*

Eine **Assoziation** ist eine semantische Beziehung zwischen Klassen, die Verbindungen (z.B. Referenzen) zwischen ihren Instanzen betreffen.

Notation:

- Name
- Multiplizität
- Beschreibung (bspw. aus Anwendungsfällen extrahiert)



Die **Multiplizität** einer Assoziation definiert den zulässigen Wertebereich für die Anzahl der Objekte, die an der Assoziation teilnehmen (können). Die Multiplizität gibt auf Klassenebene an, wie hoch die Kardinalität auf Objektebene sein darf.

engl.: *multiplicity*

- Bsp.: Eine Beziehung mit einer Multiplizität von „0..4“ besagt, dass ein Objekt 0,1,2,3 oder 4 dieser Beziehungen aufbauen darf.

Notation:

- Nummer
- Nummer..Nummer
- Nummer..*
- keine Angabe := *
- *

Beispiel:

1	3,4	
0..1	1..2	0..1,4
1..*	2..*	
*	*	

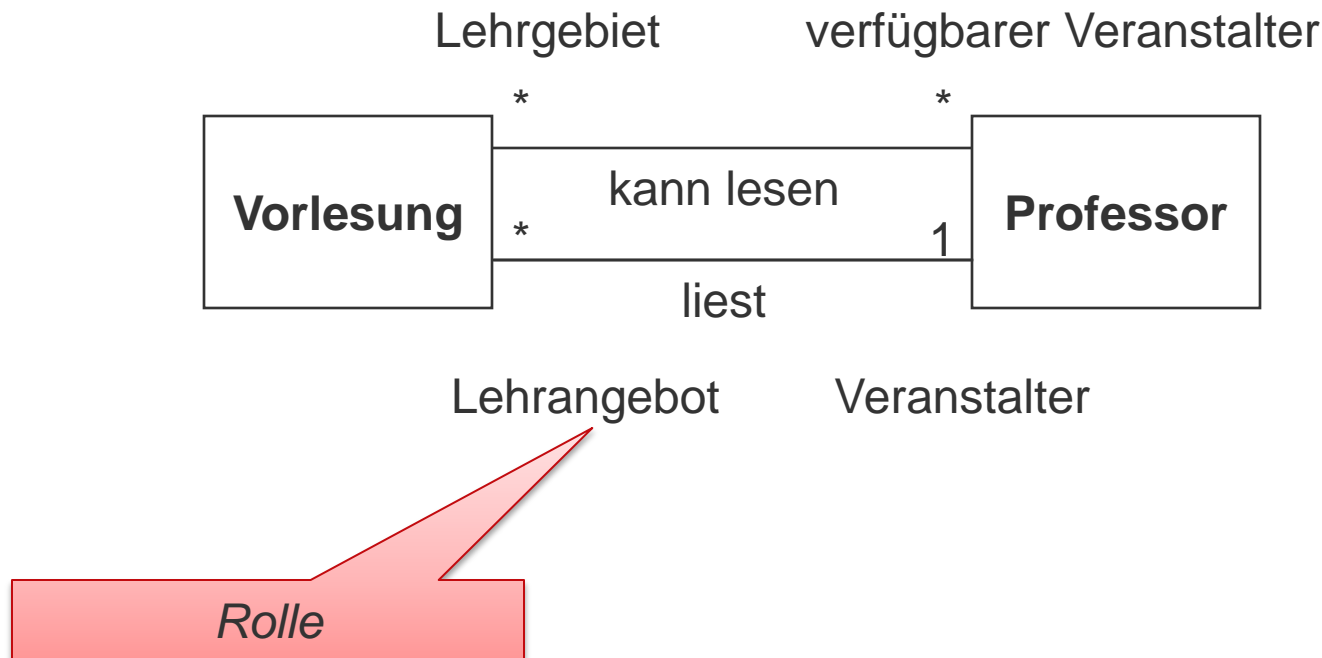
Äquivalent zu 0..*

Rolle

Eine **Rolle** in einer Assoziation ist ein Name, der die Beteiligung der Klasse an der Assoziation näher beschreibt.

engl.: **role**

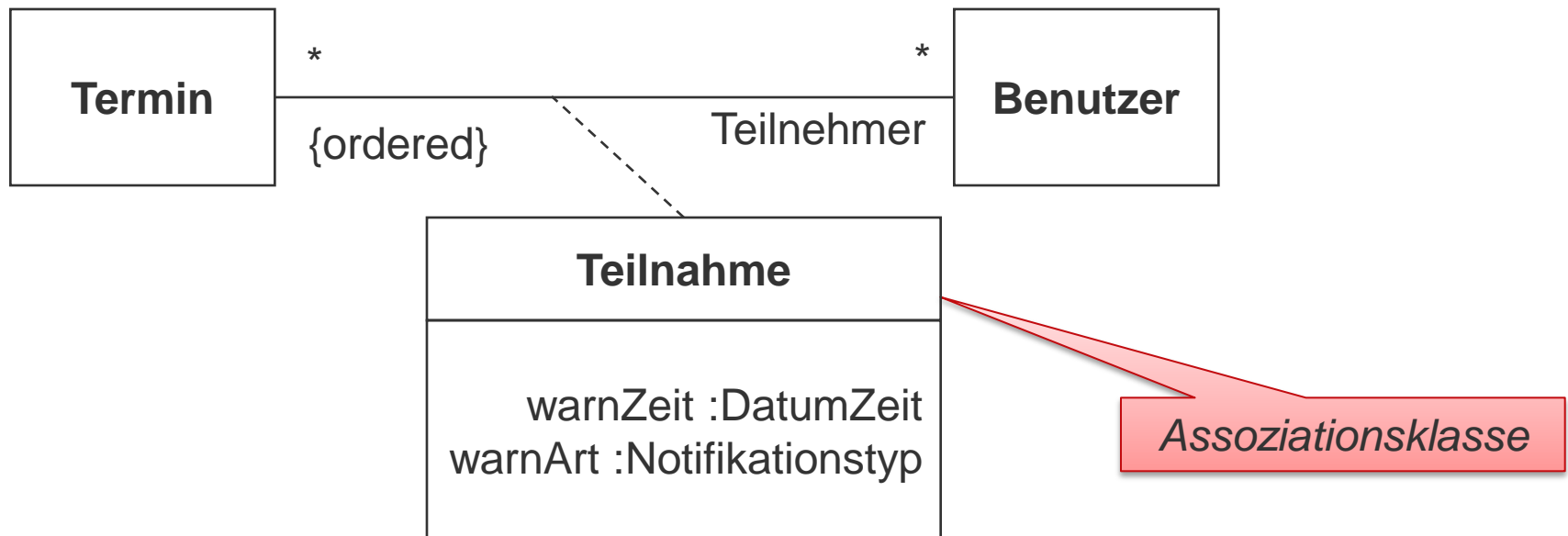
Notation:



Eine **Assoziationsklasse** ermöglicht es eine Assoziation mit eigenen Attributen auszustatten. Der Name der Assoziation stimmt dabei mit dem Namen der Assoziationsklasse überein.

engl.: **association class**

VORSICHT: Assoziationsklassen sind nicht mit schwachen Entitäten in Datenmodellen zu verwechseln! Eine Assoziation zwischen zwei Objekten ist standardmäßig eindeutig (*unique*) und die verbundene Instanz der Assoziationsklasse ist dieser und nur dieser zugeordnet.



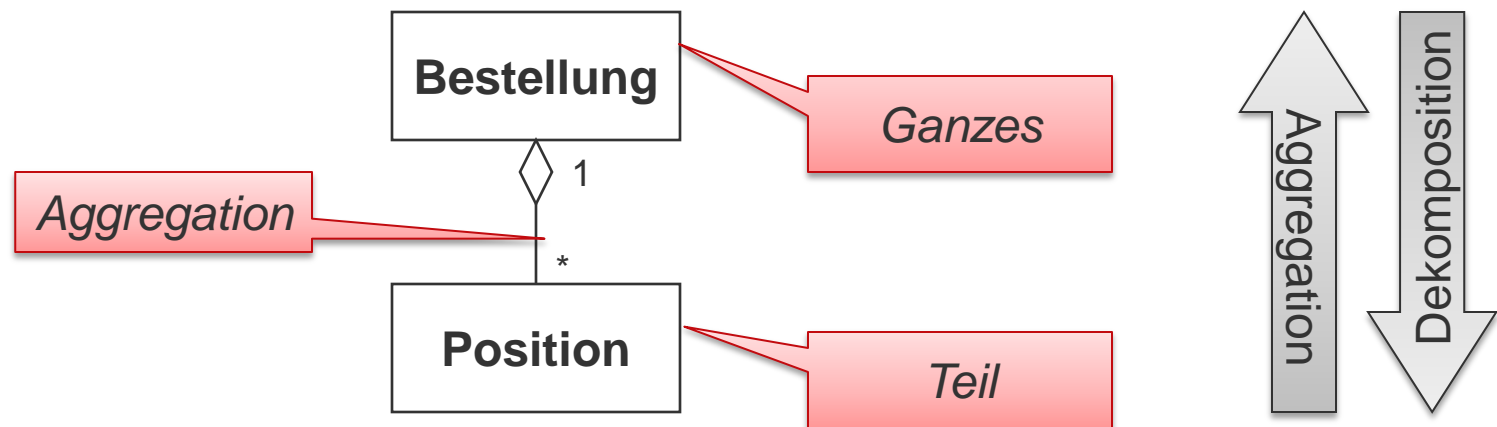
Die **Aggregation** bzw. **Komposition** ist eine semantische Beziehung zwischen einem *Ganzen* A und einem *Teil* B. "B ist ein Teil von A".

engl.: **aggregation**

Eine Aggregation kann

- eine gemeinsam genutzte Aggregation (engl.: shared aggregation) oder
- eine zusammengesetzte Aggregation / Komposition (engl.: composite aggregation) sein.

Notation:

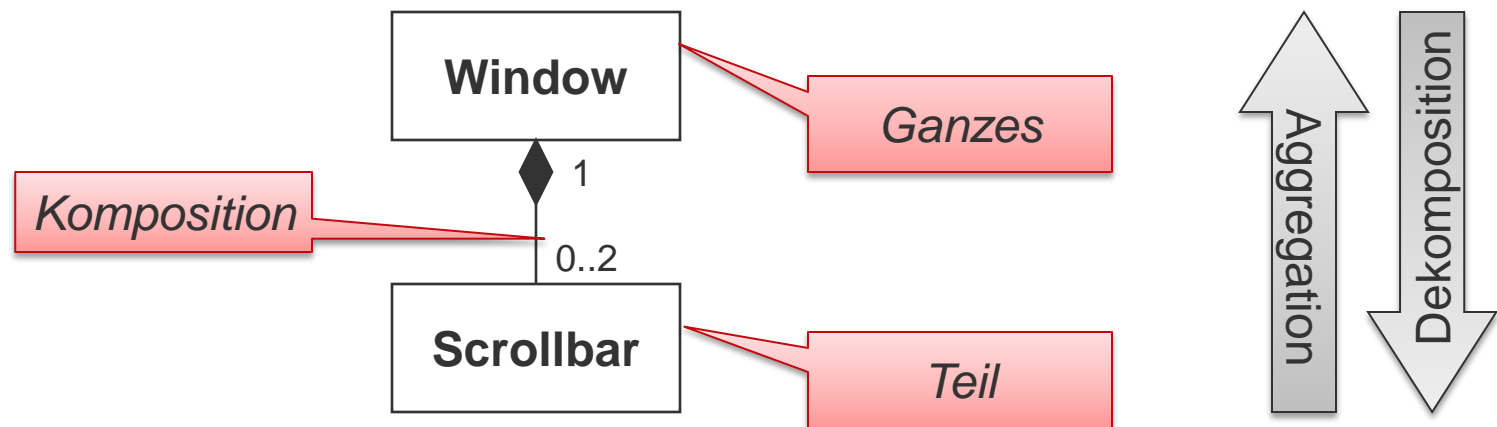


Die **Komposition** ist eine starke Aggregation.

Eigenschaften der Komposition:

- Abhängige Existenz der Komponente (*Teil* existiert nicht ohne *Ganzes* und stirbt mit ihm)
- Verbot der geteilten Aggregation (*Teil* kann nur Teil höchstens eines einzigen Aggregats sein)
- Transitivität der Komponentenbeziehung

Notation:



Die **Generalisierung** ist eine semantische Beziehung zwischen einem allgemeineren Konzept A (**Superklasse**) und einem spezielleren Konzept B (**Subklasse**).

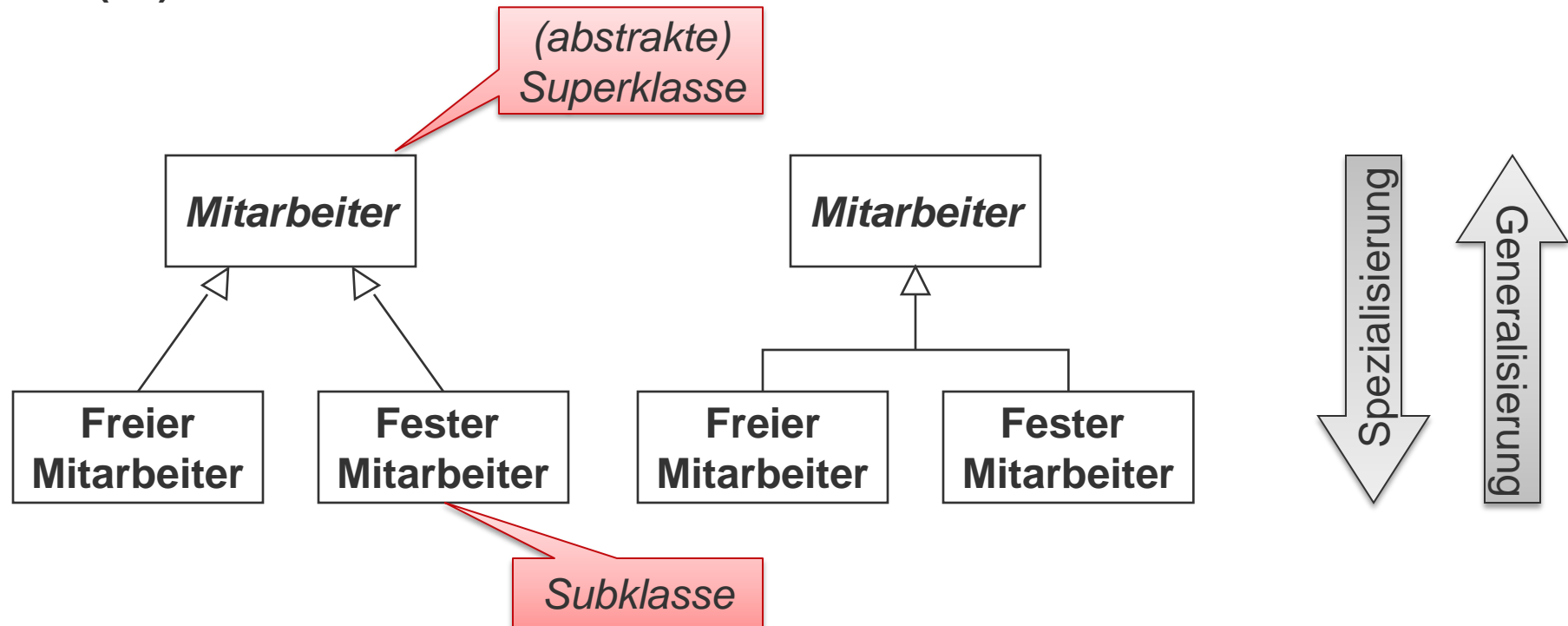
„A generalisiert B“, „jedes B ist ein A“

engl.: **generalization**

Für Klassen: **Vererbung**, Bilden einer **Taxonomie**.

[Mögliche Eigenschaften von Generalisierungen später]

Notation(en):



Eine **abstrakte Klasse** kann nicht instantiiert werden. Die Eigenschaft *isAbstract* einer Klasse ist standardmäßig auf *false* gesetzt.

Abstrakte Klassen werden häufig bei Generalisierungen und in Mustern eingesetzt.

engl.: ***abstract class***

Die Eigenschaft *isAbstract* gibt es u.a. auch für Anwendungsfälle.

Notation(en):

Mitarbeiter
{abstract}

Mitarbeiter

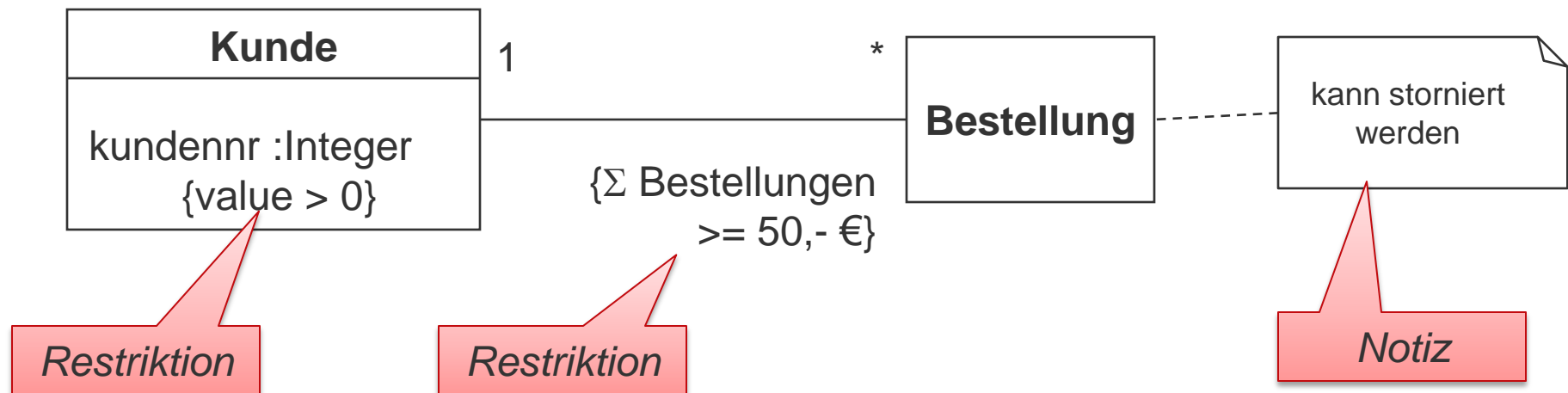
kursive Schreibweise

Restriktionen und **Notizen** können u.a. an Assoziationen, an Attributen und an Klassen annotiert werden. Restriktionen sind semantische Einschränkungen, die als Ausdruck notiert werden.

Restriktionen können in OCL (Object Constraint Language), Java oder eigener Syntax erstellt werden

engl.: **constraint**

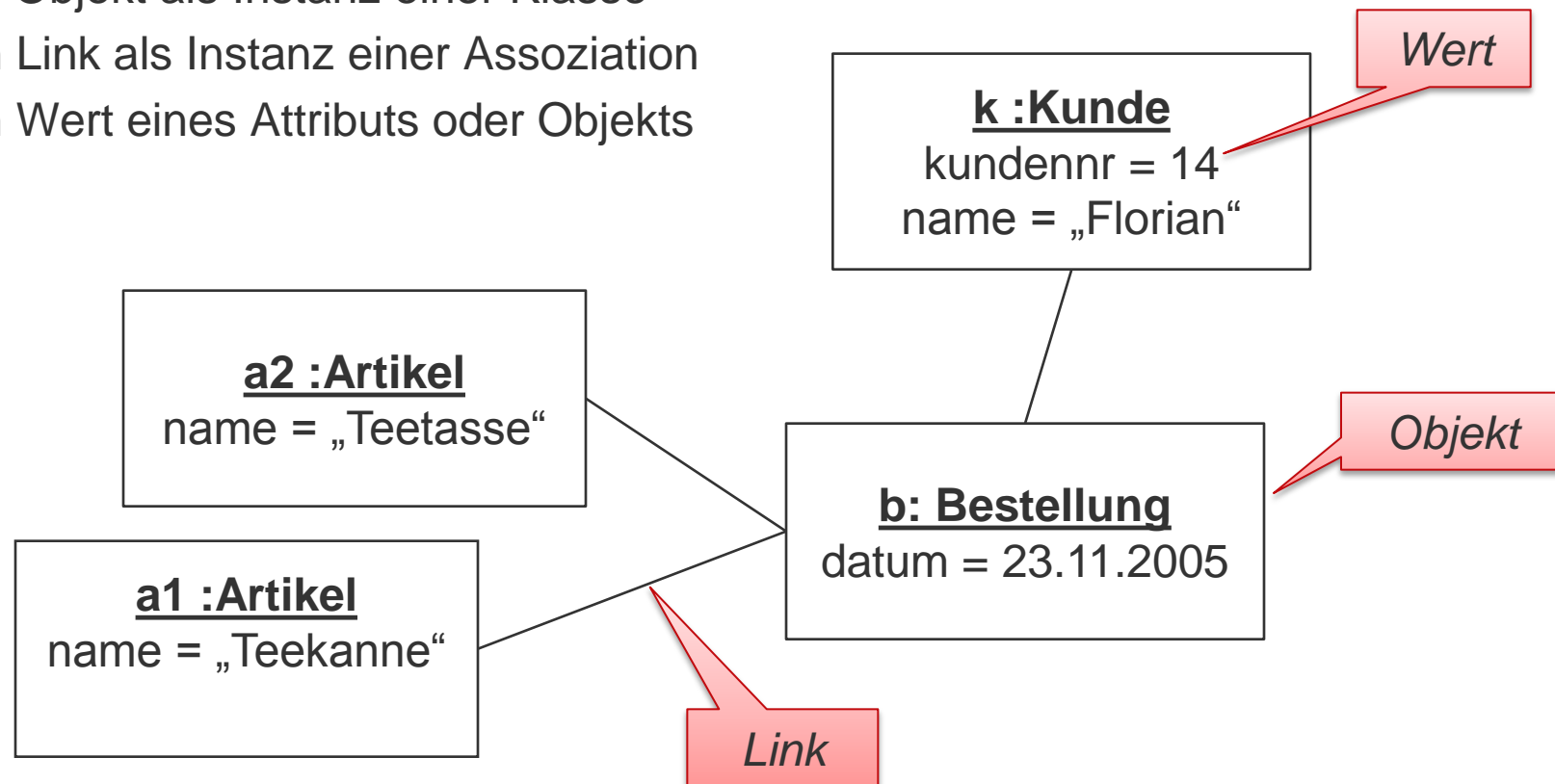
engl.: **note**



Auch Notizen können Restriktionen enthalten.

- Ein Objektdiagramm stellt das System zu genau einem Zeitpunkt (Momentaufnahme) während der Ausführung dar. Um Objektdiagramme von Klassendiagrammen zu unterscheiden, werden hier die Namen und Typangaben unterstrichen.
- Es kennt im Wesentlichen drei Arten von Instanzen
- Das Objekt als Instanz einer Klasse
- Den Link als Instanz einer Assoziation
- Den Wert eines Attributs oder Objekts

engl.: **object diagram**



Objektdiagramm vs. Klassendiagramm

	Klassendiagramm	Objektdiagramm
Assoziationen	Abstrahierte Form	Konkrete Beziehungsinstanz (engl.: <i>link</i>)
Generalisierungen	Zulässig	Nicht zulässig
Attributwerte	Standardwert	Zulässig
Multiplizitäten	Zulässig	Konkrete Werte
Typ eines Modellelements	Zulässig	Konkrete Instanz

- Problem-zentrierter Ansatz
Sammlung relevanter Abstraktionen aus der Befragung von Experten der Problemdomäne (Anwender)
- Syntaktischer Ansatz
Syntaktische Analyse der Anwendungsfälle sowie ihrer Beschreibungen
- Musterbasierter Ansatz
Verwendung wiederverwendbarer Entwurfsmuster zur Strukturierung der Problemdomäne
- Lösungs-zentrierter Ansatz
Identifikation von Klassen in bestehenden Lösungskomponenten (Alt-Anwendungen)

Die Identifikation von Klassen stellt eine nicht triviale Aufgabe dar



Problem 1: **Festlegung der Systemgrenze**

- Welche Abstraktionen liegen im, welche außerhalb des System?
- Akteure sind kein Systembestandteil
- Klassen und Objekte liegen innerhalb des Systems

Problem 2: **Blickwinkel auf das System**

- Zu einem System gibt es nicht **den** einen Satz an Klassen.
- Die Abstraktion ist stark vom geplanten Einsatzbereich und –zweck abhängig.

Entity Objects

- repräsentieren die persistenten vom System verwalteten Informationen
- stellen Konzepte aus der Anwendungsdomäne dar
- sind stets zu erfassen und modellieren
- ändern sich kaum in der Systemevolution

Boundary Objects

- stellen die Interaktionen zwischen dem System und dem Benutzer dar
- werden häufig zur Erfassung des äußeren Systemverhaltens modelliert
- ändern sich sehr häufig in der Systemevolution

Control Objects

- repräsentieren Überwachungsaufgaben des System
- werden häufig nicht explizit modelliert
- ändern sich häufiger als die Entity Objects in der Systemevolution

Syntaktischer Ansatz zur Identifikation von Klassen (Abbott's Technique)



1. Auswahl eines Anwendungsfalls sowie der zugehörigen Beschreibung
2. Analyse des Beschreibungstextes (Nomen-Verb-Analyse)
 - Nomen sind Kandidaten für Klassen und Objekte
 - Verben sind Kandidaten für Methoden
3. Trennung von Klassen und Objekten
4. Typisierung der gefundenen Klassen
 - Konzepte, die vom System gespeichert werden, sind Kandidaten für Entity Objects
 - Artefakte, welche die Systemschnittstelle zur Umwelt beschreiben, sind Kandidaten für Boundary Objects
 - Vorgänge, die vom System beobachtet werden müssen, sind Kandidaten für Control Objects

Ablauf eines Anwendungsfalls:

1. Ein Kunde betritt ein Geschäft, um ein Spielzeug zu erwerben.
2. Dieses Spielzeug muss seiner Tochter gefallen und muss weniger als 50€ kosten.
3. Der Kunde probiert ein Videospiel mit einem Datenhandschuh und einer VR-Brille aus. Es gefällt ihm.
4. Ein Mitarbeiter des Geschäfts hilft dem Kunden.
5. Die Eignung eines Spielzeugs für ein Kind hängt vom Alter des Kindes ab.
6. Die Tochter des Kunden ist aktuell 3 Jahre alt.
7. Der Mitarbeiter des Geschäfts empfiehlt ein ungefährlicheres Spielzeug, nämlich eine Stoffpuppe der „Teletubbies“.

Abbott's Technique am Beispiel (2)

Wort	Wortart	Konzept im Objektmodell
„Teletubbies“	Substantiv, Eigename	Objekt
Spielzeug	Substantiv, Gattungsname	Klasse
erwerben	Verb	Methode
ist (ein)	Identifizierendes Verb	Vererbungsbeziehung
muss	Modales Verb	Einschränkung
ungefährlich	Adjektiv	Attribut
hängt ab von	Intransitives Verb	Einschränkung, Beziehung
...		
hat (ein)	Besitzanzeigendes Verb	Aggregation

Wer liest Klassendiagramme / Wer liest sie nicht?



Ein Klassendiagramm beschreibt statische Eigenschaften eines Systems.

Es wird hauptsächlich genutzt vom:

- **Domänenexperten** des Anwendungsdomäne
 - um die Anwendungsdomäne abzubilden
 - als Hilfsmittel bei der Anforderungsanalyse
- **Entwickler**
 - als Hilfsmittel beim Entwurf des Systems
 - als Vorlage für die Implementierung des Systems

Nicht am Klassendiagramm interessiert sind der

- **Kunde**, welcher sich mehr mit Projektverlauf beschäftigt
- **Endbenutzer**, welcher sich für Funktionalitäten interessiert

Der Blickwinkel des Entwicklers auf ein Objektmodell wechselt



Entsprechend der momentanen Tätigkeit im Entwicklungsprozess, nimmt der Entwickler verschiedene Rollen ein:

- Analyst
- System / Objekt Designer
- Implementierer (Entwickler)

Mit jeder dieser Rollen ist ein anderer Blickwinkel auf das Objektmodell und auf dessen Klassendiagramm verbunden.

Der Analyst interessiert sich für

- **Klassen der Anwendungsdomäne**: die Assoziationen zwischen den Klassen stellen dabei Beziehungen zwischen Abstraktionen in der Anwendungsdomäne dar
- **Methoden** und **Attribute** von Klassen der Anwendungsdomäne

Der Analyst verwendet das Konzept der **Vererbung**, um eine **Taxonomie** der Anwendungsdomäne zu erstellen.

Der Analyst ist nicht interessiert

- an der **Signatur** der Methoden
- Klassen aus der **Lösungsdomäne**

Das Klassendiagramm des Analysten wird **konzeptuelles Klassendiagramm** genannt.

Der Designer operiert in der Lösungsdomäne.

Die Assoziationen zwischen den Klassen werden durch Referenzen zwischen den Klassen realisiert.

Eine wichtige Aufgabe im Design ist die Spezifikation von Schnittstellen:

- Der Designer beschreibt Schnittstellen von Klassen und Subsystemen
- Subsysteme stellen eine Sammlung von Klassen (einen sog. Modul) mit einer wohldefinierten Schnittstelle dar
- Der Designer bemüht sich um die Wiederverwendung von Klassen bestehender System (Bibliotheken, Frameworks).
- Der Designer optimiert seine Klassen auf Wiederverwendung in zukünftigen Systemen.

Das Klassendiagramm des Designers heißt **Entwurfsdiagramm**.

Der „Klassenimplementierer“

- setzt die Schnittstelle einer Klasse in einer Programmiersprache um
- interessiert sich für Datenstrukturen und Algorithmen

Der „Klassenerweiterer“

- ist an der Frage interessiert, wie sich eine bestehende Klasse zur Lösung eines neuen Problems erweitern oder anpassen lässt

Der „Klassenverwender“

- interessiert sich für die Signaturen der Methoden sowie die Vorbedingungen, unter welchen die Methoden ordnungsgemäß arbeiten
- abstrahiert von der Implementierung der Methoden durch die Klasse

Wozu ist die Unterscheidung der verschiedenen Blickwinkel gut?



Die Blickwinkel verwenden dieselbe Notation und werden deswegen nicht selten vermischt. Typische Probleme in diesem Zusammenhang sind:

- Fehlende Unterscheidung zwischen Anwendungs- und Lösungsdomäne
 - Grund: UML erlaubt es, beide Arten von Klassen in einem Modell zu verwenden
 - Lösung: Strikte Vermeidung von Lösungsklassen im Analysemodell

- Fehlende Unterscheidung zwischen Spezifikation und Implementierung einer Klasse
 - Grund: Objektorientierte Programmiersprachen erlauben es gleichzeitig eine Klasse zu spezifizieren und zu implementieren
 - Lösung: Trennung zwischen Analyse- und Designmodell – ersteres enthält keine Implementierungsinformationen

Das Analysemodell wird in der Analysephase geschaffen

- Dient zur Kommunikation zwischen den Stakeholdern Endbenutzer, Kunde und Analyst.
- Das Klassendiagramm beschränkt sich auf Klassen aus der Anwendungsdomäne.

Das Designmodell wird in der Designphase geschaffen

- Dient zur Kommunikation zwischen den Stakeholdern Designer, „Klassenimplementierer“, „Klassenverwender“ und „Klassenerweiterer“.
- Das Klassendiagramm enthält Klassen aus Anwendungs- und Lösungsdomäne.

Szenarien modellieren die Benutzersicht; ignorieren Zusammenhänge, Daten und die zur Erzeugung der Resultate notwendigen Operationen weitgehend

Objekt- und Klassenmodelle modellieren Zusammenhänge, Daten, und die zur Erzeugung der Resultate notwendigen Operationen; ignorieren aber die Benutzersicht weitgehend

Wichtig:

- Die Betrachtung der einzelnen Sichten bei der Entwicklung von Software-Systemen ist notwendig
- Die Sichten müssen miteinander verknüpft werden