Einführung in die Informationsverarbeitung

Øyvind Eide

Woche 13
Modellierung
Planung

oeide@uni-koeln.de
http://idh.uni-koeln.de

II 0. UML 2.0 (2003 / 04 ff.)

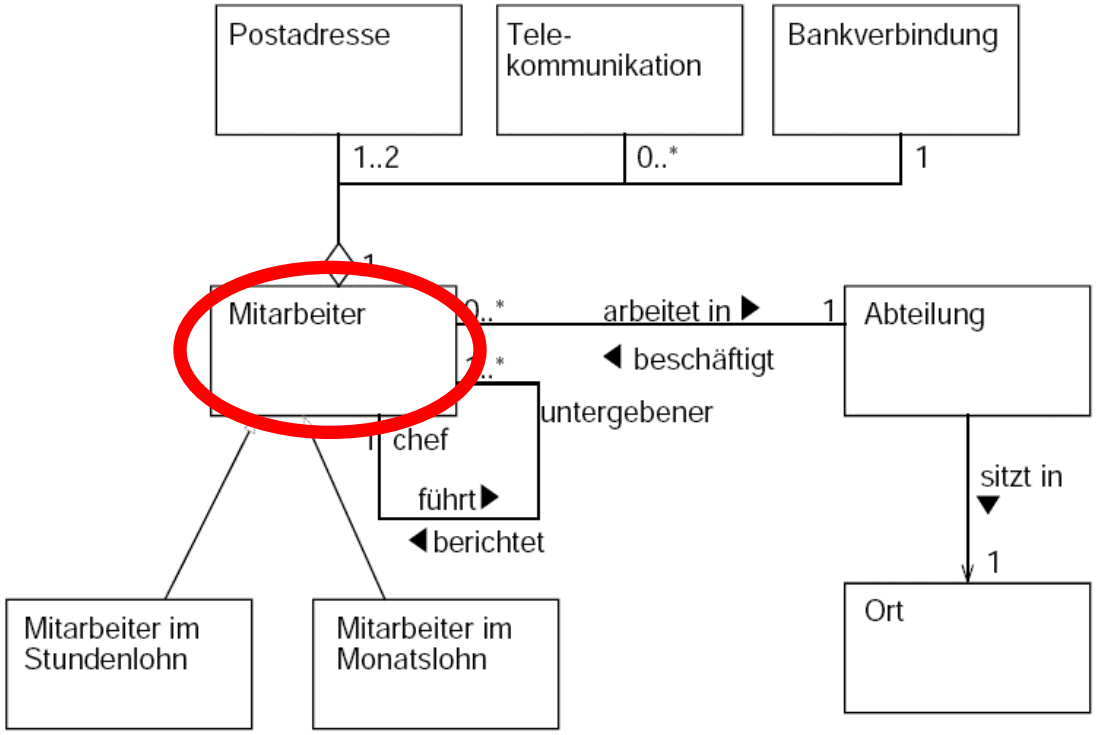
UML ist eine Sammlung von "*graphischen Sprachen*", d.h. Regelsystemen für die Konstruktion graphischer Schemata, die:

- ❖unterschiedliche Perspektiven von Anforderungen an Systeme und Entwürfen von Systemteilen, sowie deren Zusammenwirken darstellen,
- ❖einander dabei überlappen können und
- ❖unabhängig voneinander verwendet werden können.

Am wichtigsten:

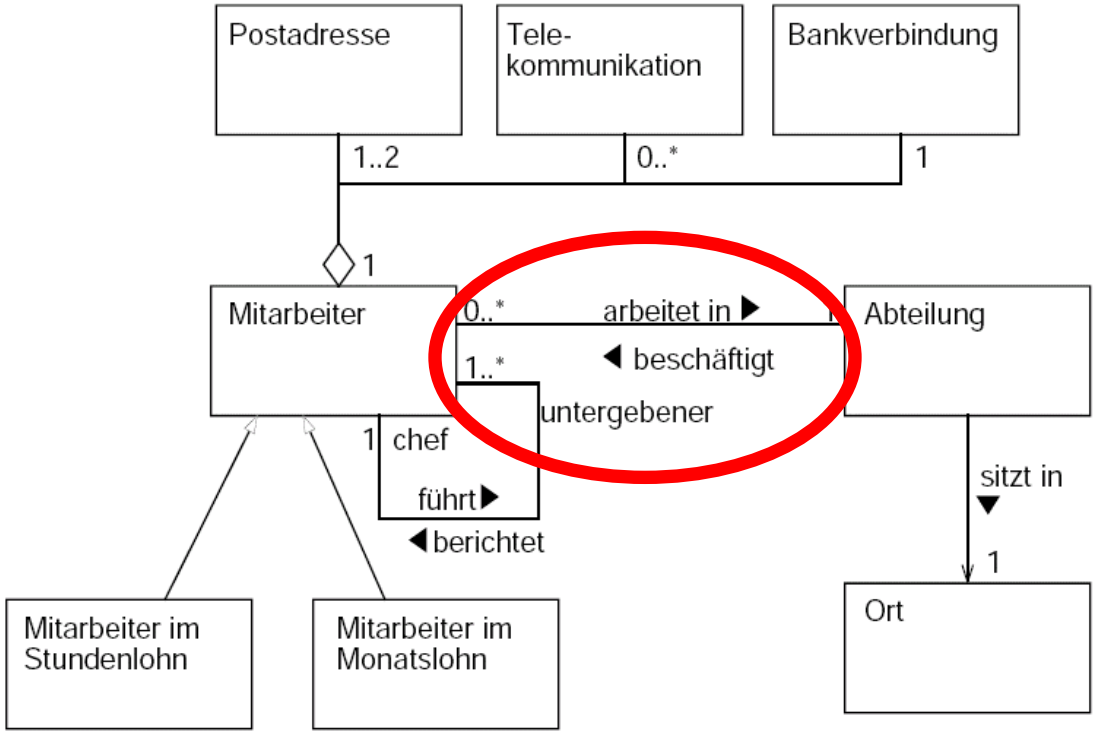
- ❖Klassenmodelle beschreiben den strukturellen Aufbau eines Systems,
- ❖Anwendungsfallmodelle (Use Cases) beschreiben die Interaktion mit dem System aus Benutzersicht.

II 1. Klassendiagramme



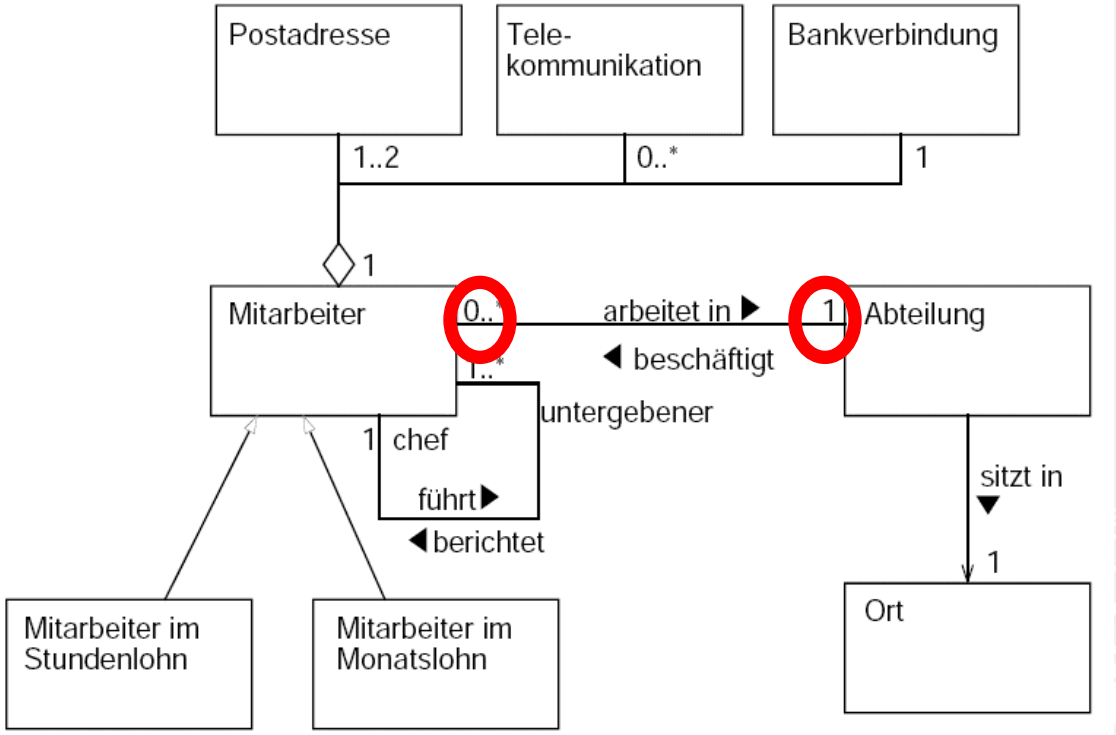
Objekt „Mitarbeiter“
(kann Attribute und Methoden haben) ➔ Programmierung

II 1. Klassendiagramme



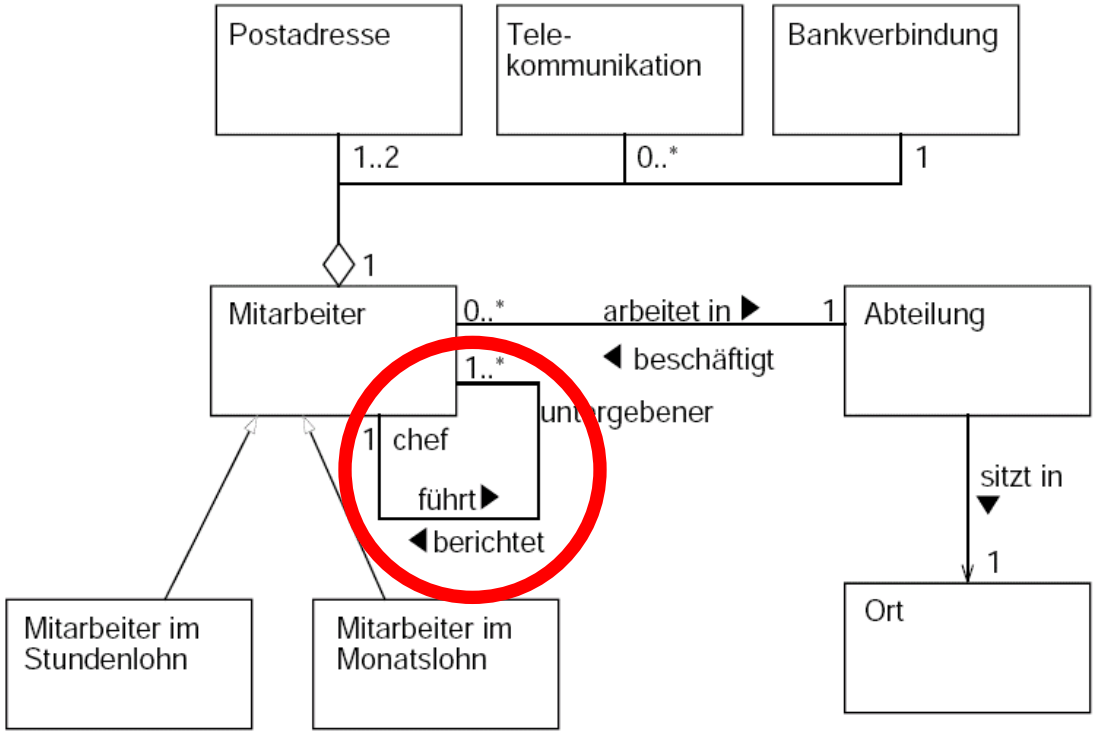
Binäre Assoziation beschreibt die Beziehungen zwischen Klassen

II 1. Klassendiagramme



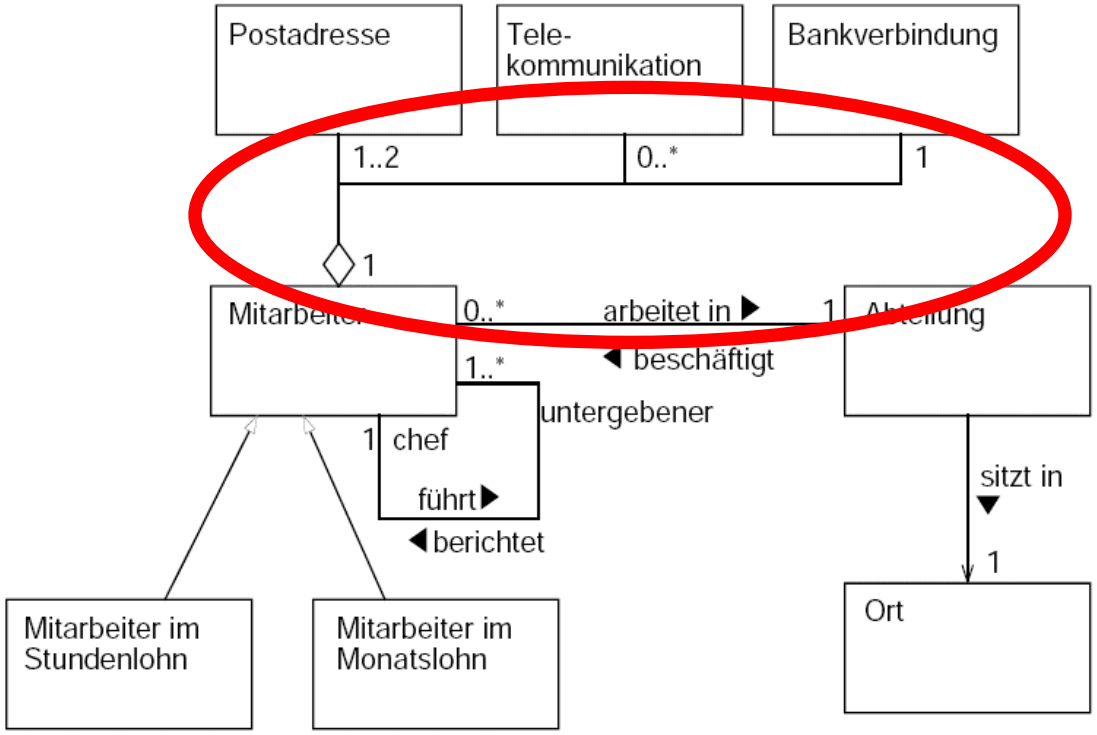
Multiplizität gibt an, wie viele Objekte an einer Assoziation beteiligt sein können.

II 1. Klassendiagramme



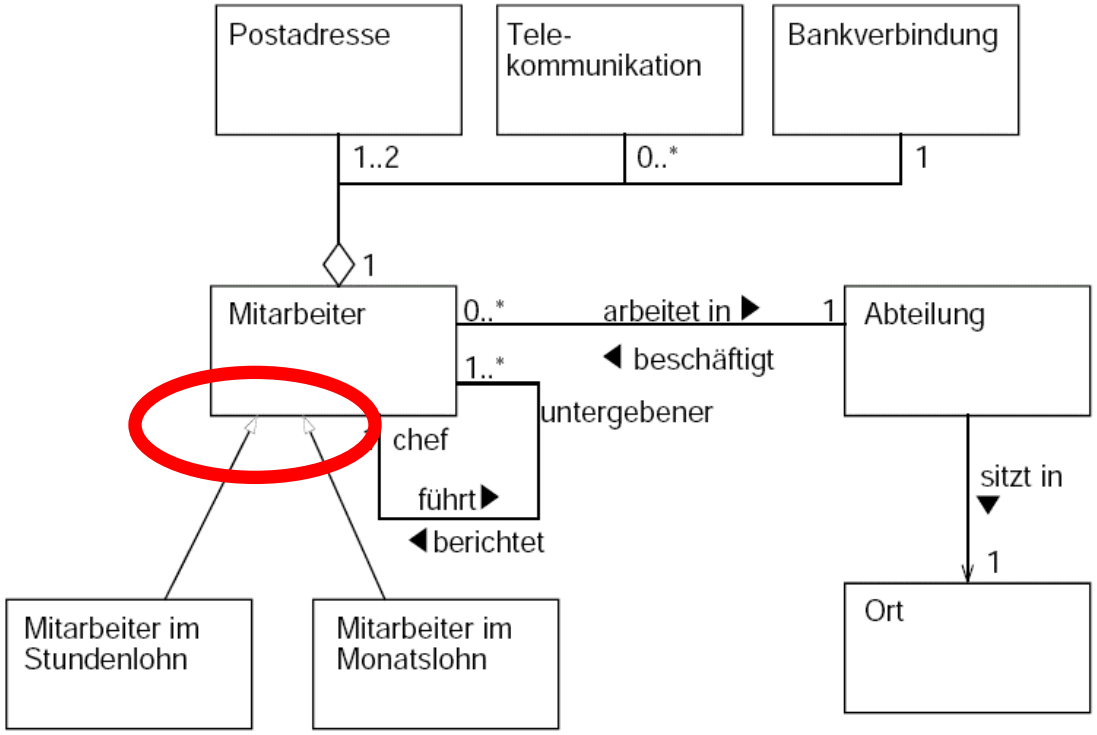
Reflexive Assoziation verbindet Objekte einer Klasse miteinander.

II 1. Klassendiagramme



Aggregation verbindet beliebig viele Klassen zu einer übergeordneten.

II 1. Klassendiagramme



Generalisierungsbeziehung zwischen *Superklasse* und *Subklasse*.

II 2. Anwendungsfalldiagramme

Das Verhalten eines Systems kann als Sammlung von *Anwendungsfällen (= use cases)* beschrieben werden.

Ein Anwendungsfall beschreibt eine Klasse möglicher Interaktionen.

Konkrete Anwendungsfälle heißen auch *Szenarien*.
(➔scenario based design.)

Anwendungsfälle werden in strukturiertem Text beschrieben.

Alle möglichen Anwendungsfälle - oder ein für ein bestimmtes Teilsystem relevanter Teil - werden als *Anwendungsfalldiagramm* realisiert.

II 2. Anwendungsfalldiagramme

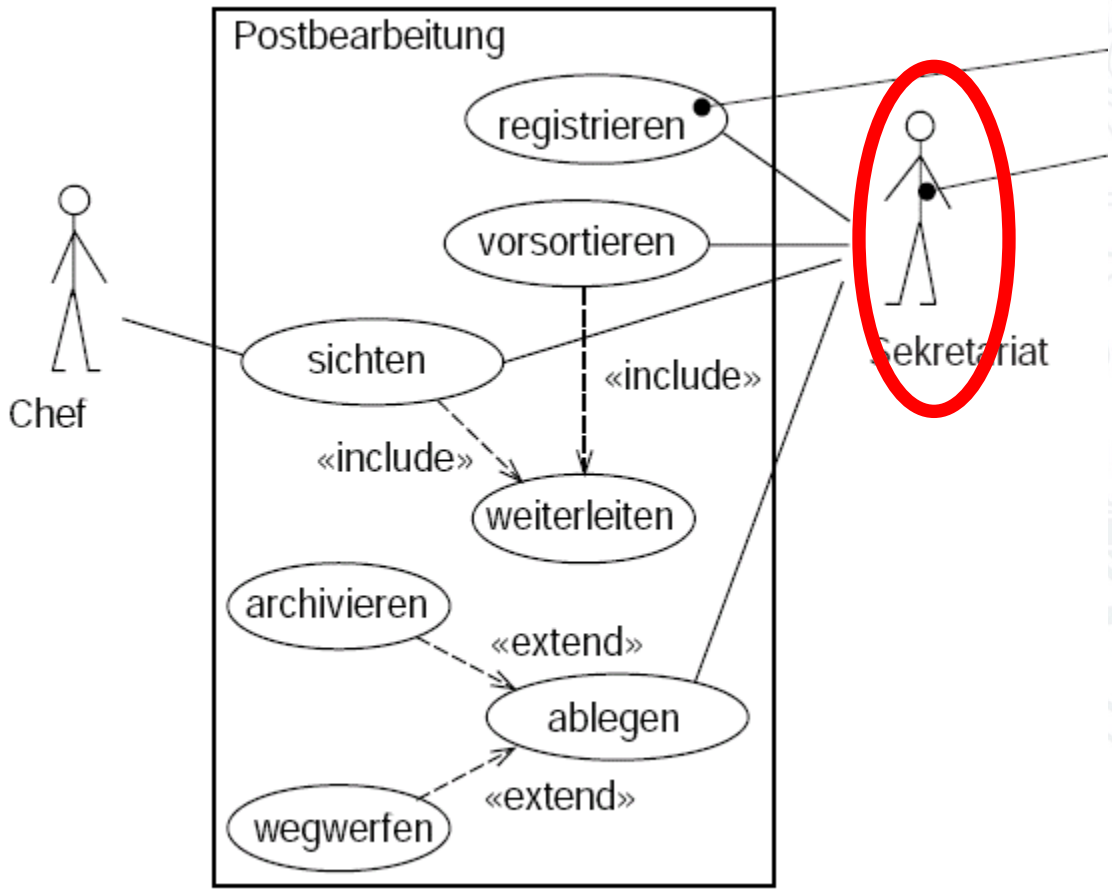
Anwendungsfall als strukturierter Text (auch als Aktivitäts – oder Zustandsdiagramme)

Beispiel: "Buch an einem Selbstausleiheautomaten ausleihen"

Normallfall:
1.BenutzerIn liest Ausweis in System ein; System validiert Ausweis.
2. BenutzerIn wählt "Ausleihen"; System aktiviert Ausleihfunktion.
3.BenutzerIn liest Buchcode ein; System identifiziert das Buch, registriert Ausleihe, deaktiviert das Diebstahletikett.

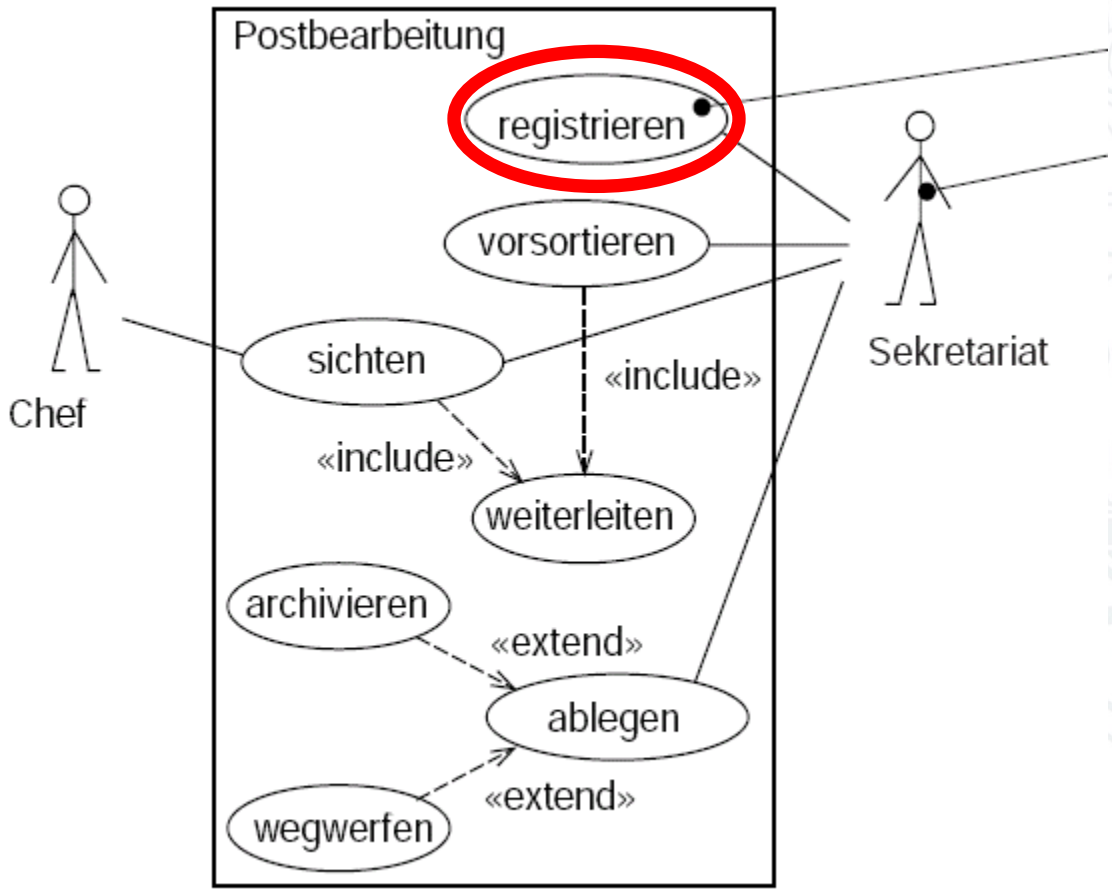
Auch Sonderfälle

II 2. Anwendungsfalldiagramme



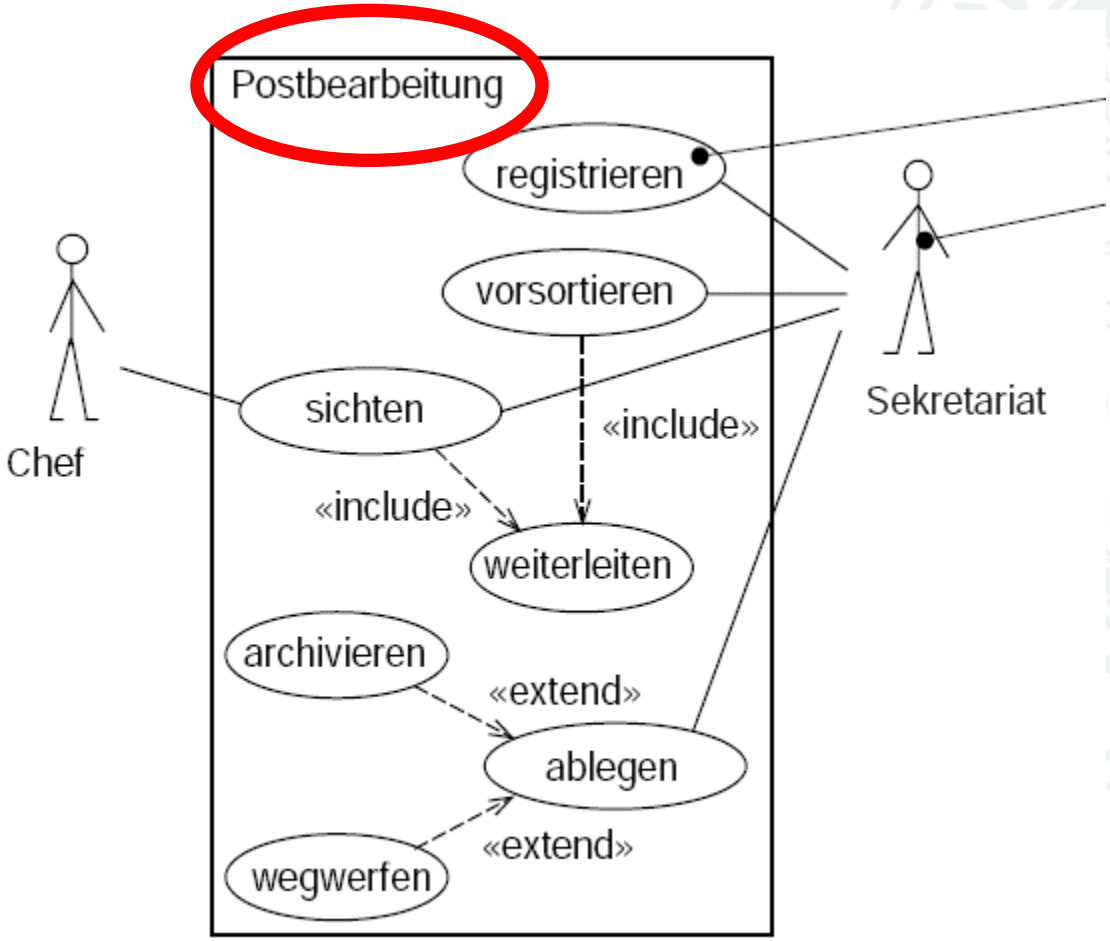
Akteur

II 2. Anwendungsfalldiagramme



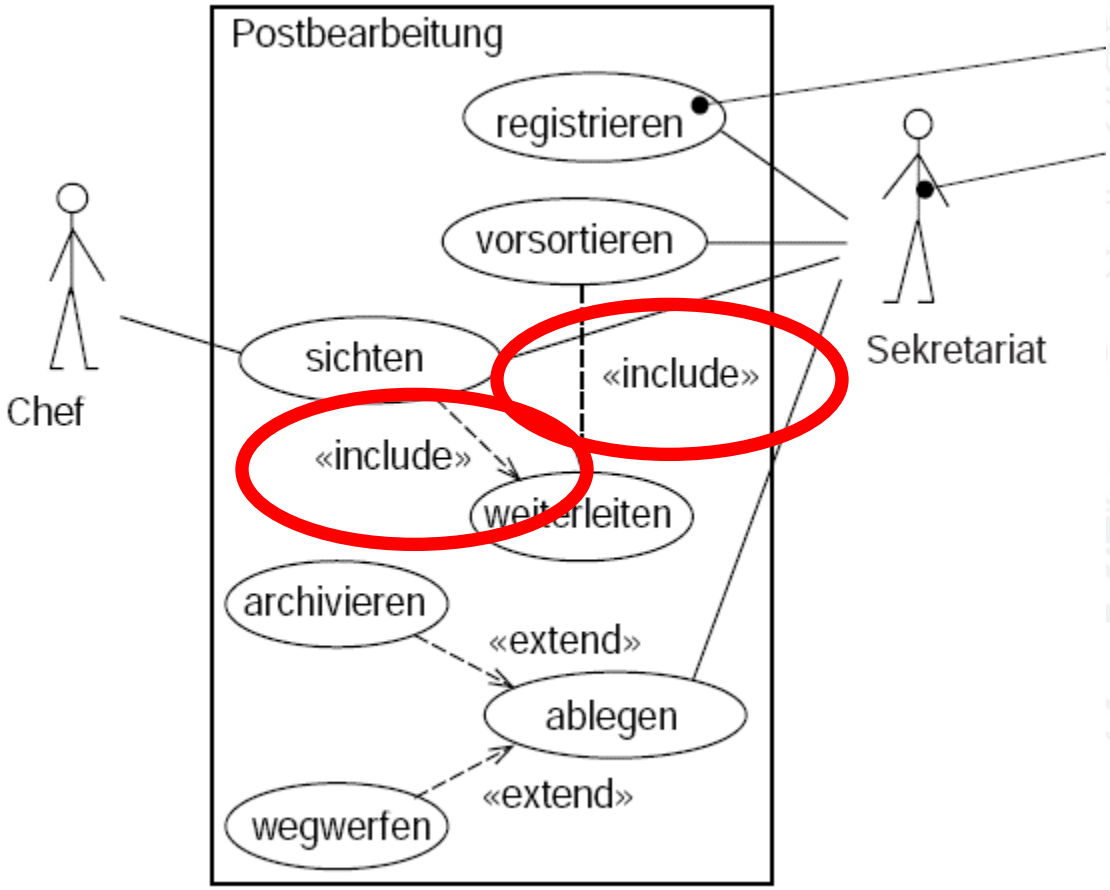
Anwendungsfall

II 2. Anwendungsfalldiagramme



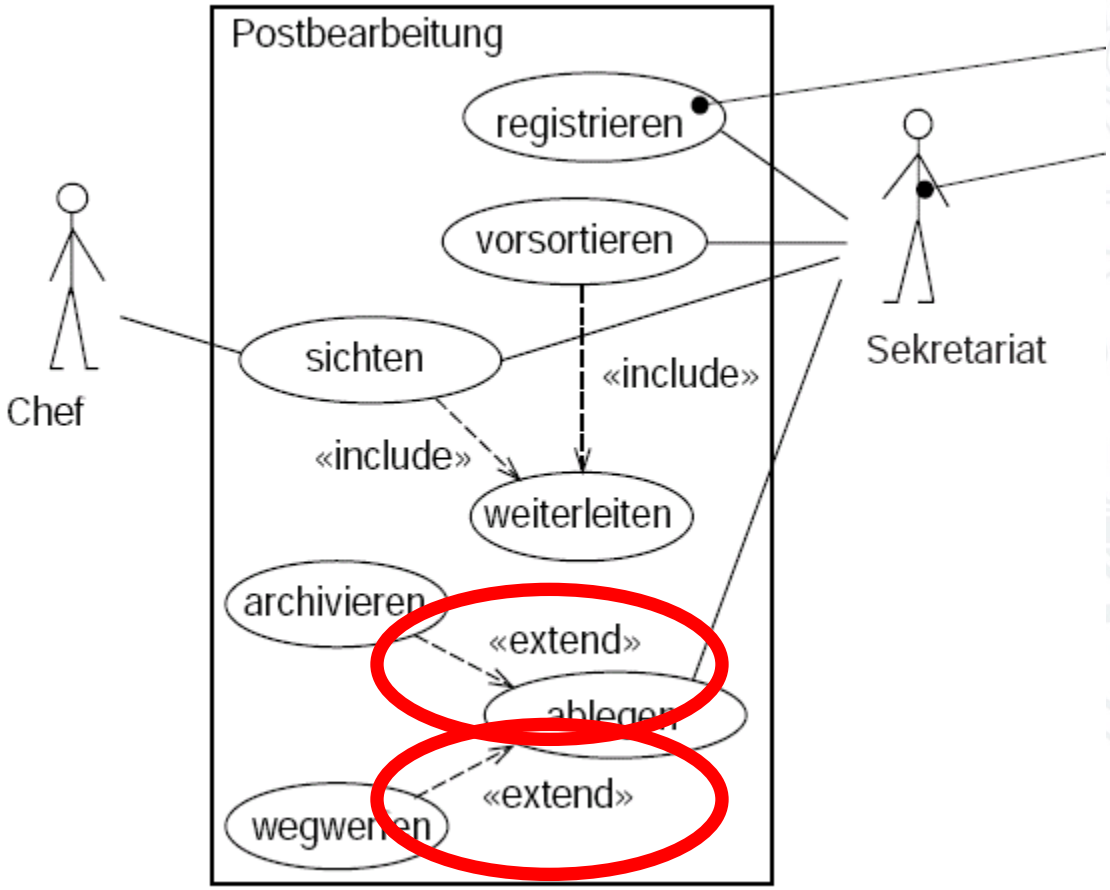
Anwendungsfall-diagramm

II 2. Anwendungsfalldiagramme



Include:
Bindet anderen Anwendungsfall ein, der an mehreren Stellen genutzt werden kann.

II 2. Anwendungsfalldiagramme



Extend:
Modelliert Varianten, die einen Basisanwendungsfall abwandeln.

II 3. Zustandsdiagramme

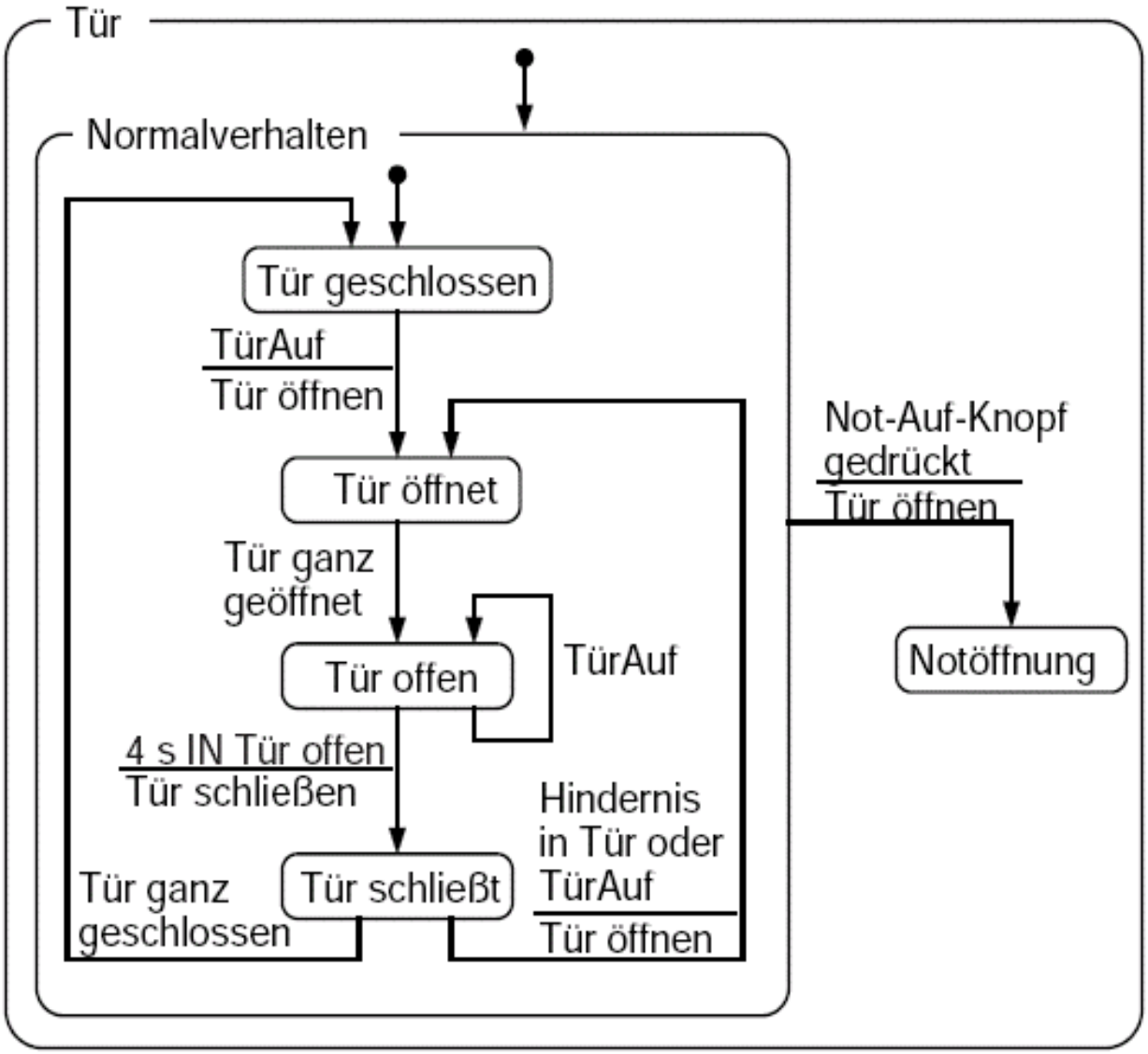
Zustandsdiagramme modellieren das dynamische zeitliche Verhalten eines Systems.

Auch *state machine* ➔ *state diagram*

Mögliche Zustände der Objekte einer Klasse oder eines Teilsystems.

Dynamik des Systemverhaltens: Reaktionen auf äußere Ereignisse.

II 3. Zustandsdiagramme



Thesaurus

- Ein kontrolliertes Vokabular
- Begriffe sind miteinander verbunden
  - durch Relationen
- ein Themengebiet
  - zu beschreiben
  - zu repräsentieren

Beispiel: TGN

- Getty Thesaurus of Geographic Names
- Eine Datenbank
  - ~1.000.000 Bezeichnungen
  - ~900.000 Orte
  - vom Getty Research Institute herausgegeben
- Verschiedene Hierarchien
  - aktuellen und historischen
  - geophysischen als auch geopolitischen

Beispiel: TGN

ID: 1003171 **Record Type: administrative**

Köln (national district)

Coordinates:
Lat: 50 56 00 N *degrees minutes* Lat: 50.9333 *decimal degrees*
Long: 006 40 00 E *degrees minutes* Long: 6.6667 *decimal degrees*

Names:
Köln (**preferred**,C,V)
Köln district (C,O,display)

Hierarchical Position:
World (facet)
.... Europe (continent) (P)
..... Germany (nation) (P)
..... North Rhine-Westphalia (state) (P)
..... Köln (national district) (P)

Place Types:
national district (**preferred**, H)
second level subdivision (H)

Ontologien

- Informatik
  - (und Philosophie)
- Darstellungen von Begrifflichkeiten
  - formal geordnete
  - Beziehungen
  - Gegenstandsbereich
- Enthalten
  - Inferenzregeln
  - Integritätsregeln

Beispiel: Semantic MediaWiki

- freie Open-Source-Softwareerweitung
- von der Wikipedia genutzt
- Informationen expliziter zu machen:
  - typisierte Verweise (Relationen)
  - (Seiten-)Attribute
- Ermöglich Erstellung Ontologien

Semantic MediaWiki Graph

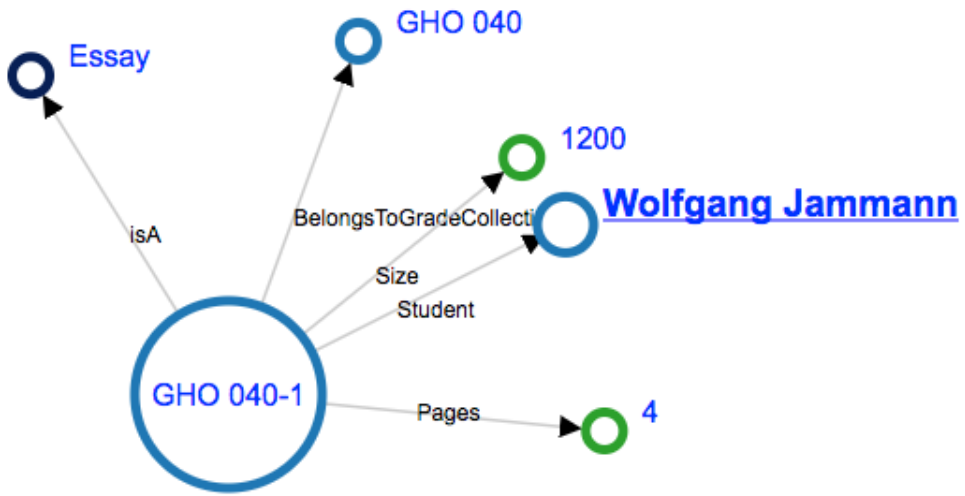
Wiki Article*

GHO 040-1

Submit

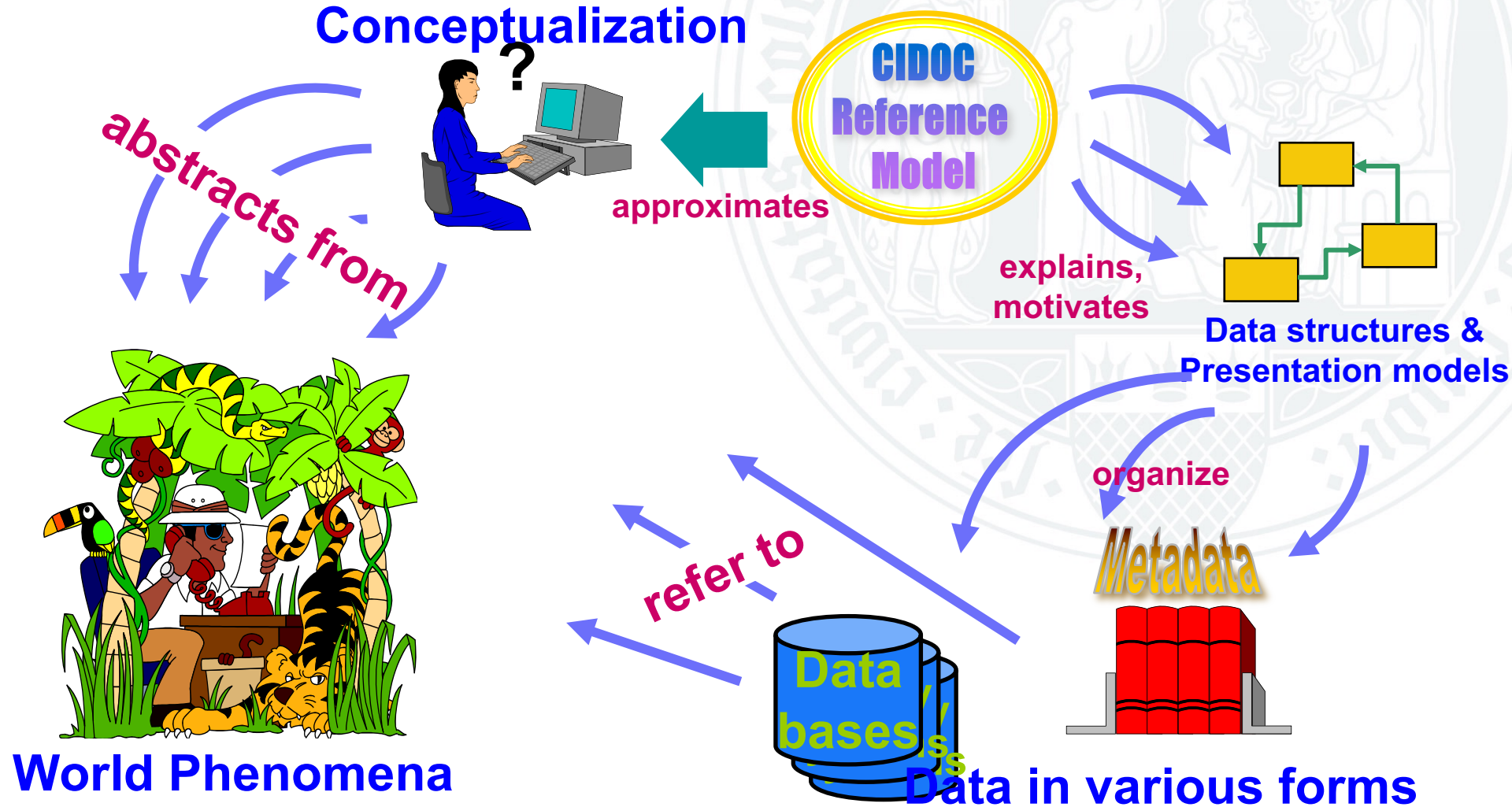
Color Keys for Data Types...

- Category
- Internal Link
- Number
- Text

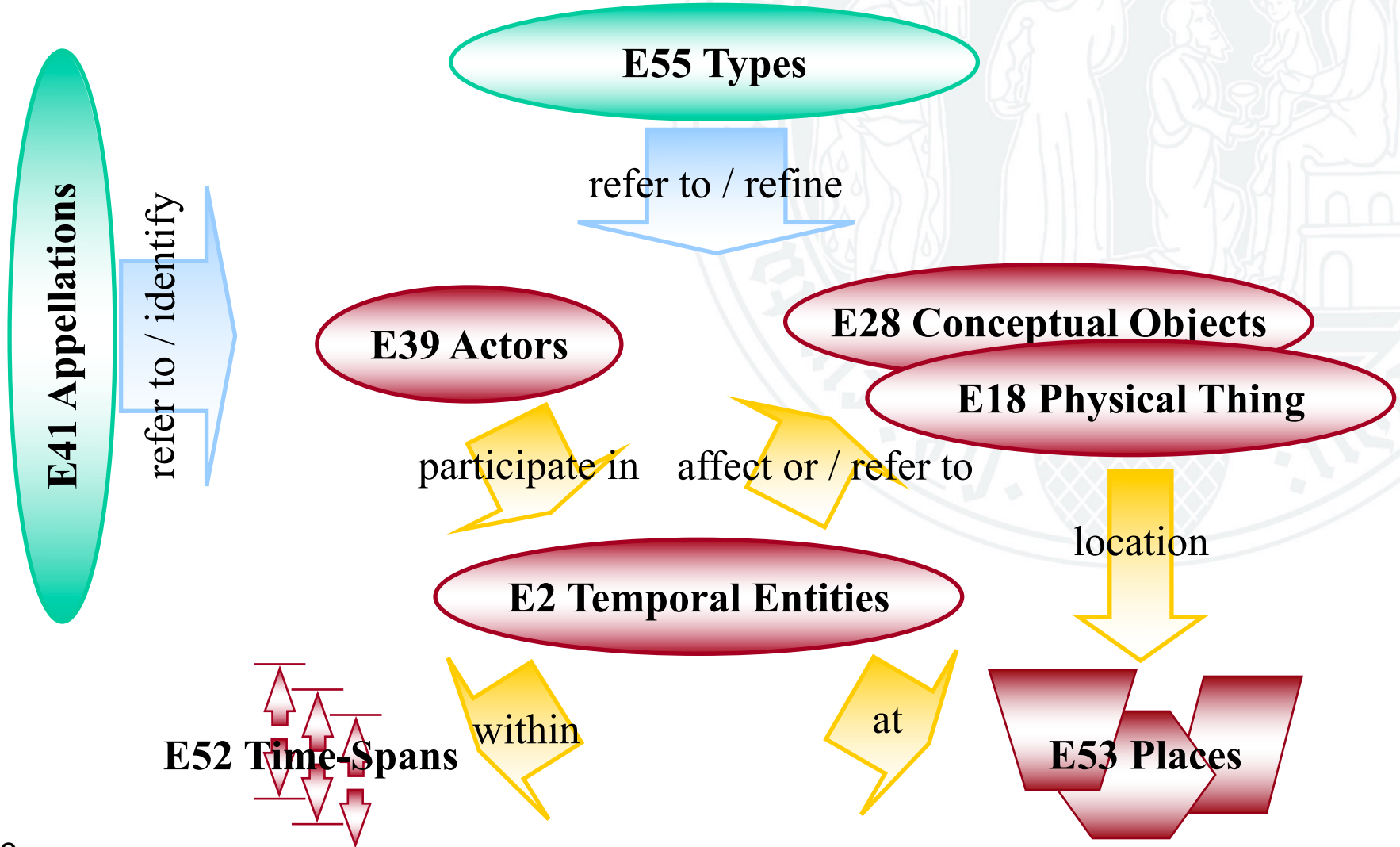


Beispiel: CIDOC-CRM

The Intellectual Role of the CRM



Top-level classes useful for integration



Rekapitulation

Der Gang der Argumentation:

1. Der Rohstoff: Information
2. Darstellung der Information auf dem Rechner: Datenstrukturen
3. Verarbeitung der Information auf dem Rechner: Algorithmen
4. Abstrakte Lösungen für den Entwurf von Systemen: Graphen und andere Modelle

Systemdesign / Systemplanung

1. Entsteht Software, entstehen Informationssysteme als Ergebnis eines künstlerischen Prozesses?

2. Oder sind sie planbar?

Ein ernstes Problem …

Erfolg von IT Projekten laut Umfragen:

- 45,2 % aller Softwareprojekte erfolgreich.
- 19,4 % Zeit- und Kostenüberschreitungen.
- 35,4 % Fehlschläge.

Ein ernstes Problem …

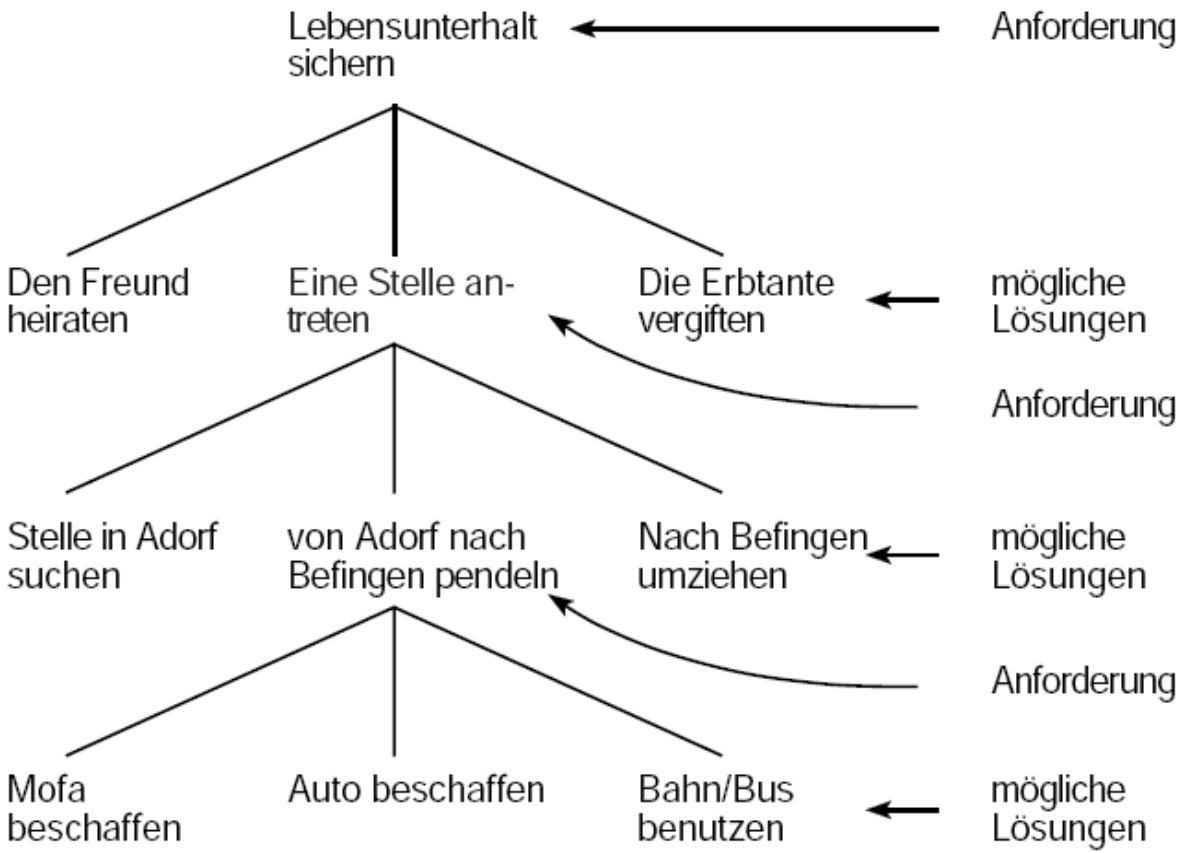
In Abhängigkeit von der Teamgröße.
Erfolgreiche Projekte bei einer Teamgröße von:

- Bis 4 Personen
  60 %
- 4 – 8 Personen
  38 %
- 8 – 20 Personen
  32 %
- Mehr als 20 Personen
  18 %

Was heißt Planung?

Hierarchische Verzahnung von Problem und Lösung

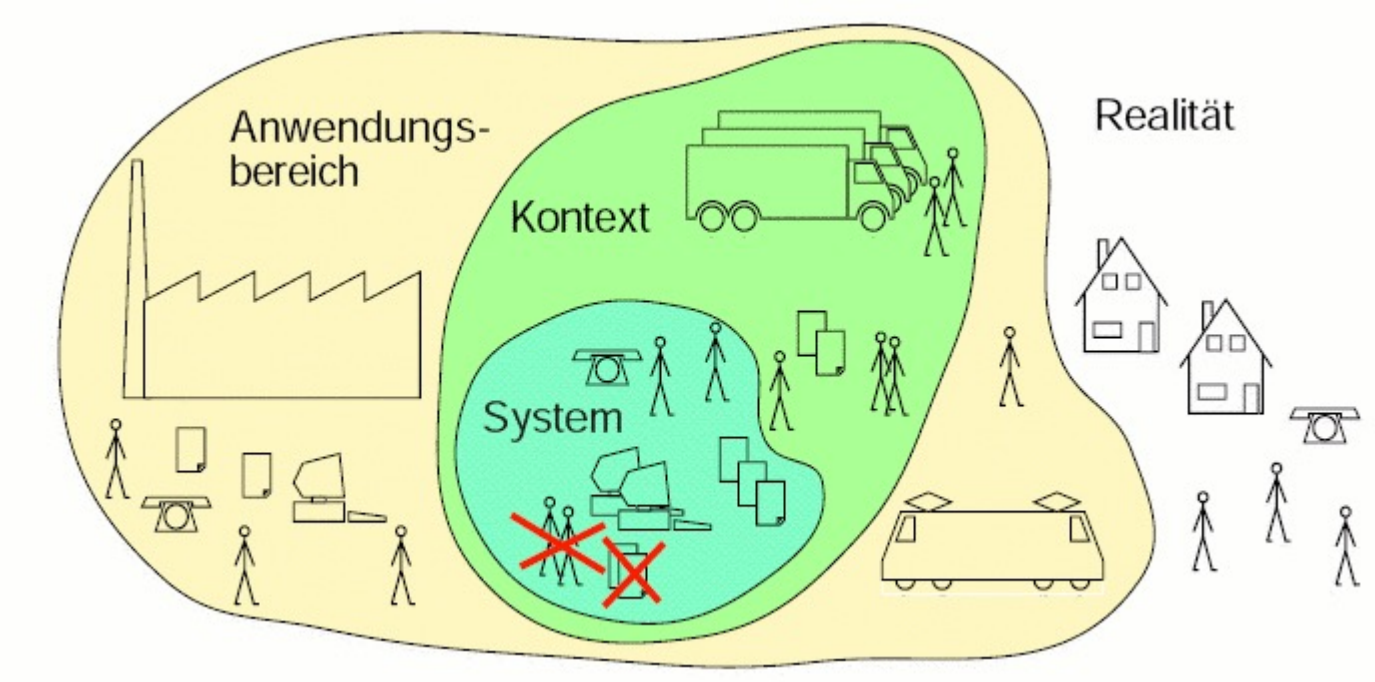
Problem: Sonja Müller hat ihr Studium abgeschlossen und erhält keine Unterstützung von ihren Eltern mehr. Sie ist daher mit der Anforderung konfrontiert, ihren Lebensunterhalt zu sichern. Sie wohnt in Adorf und hat ein Stellenangebot bei einer Firma in Befingen. Ferner hat sie einen reichen Freund und eine ebenso reiche Erbtante.



Was heißt Planung?

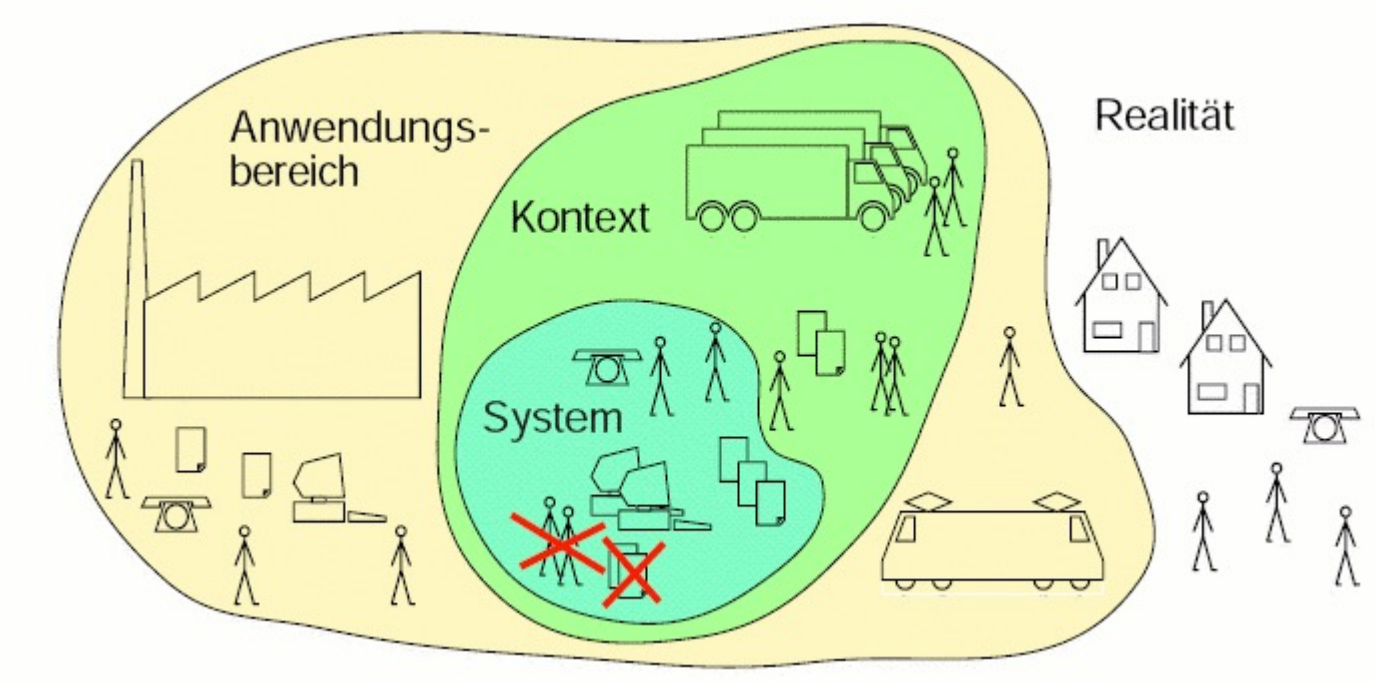
Der eben beschriebene Vorgang, angewendet auf informationstechnische Probleme: Requirements Engineering.

Requirements Engineering



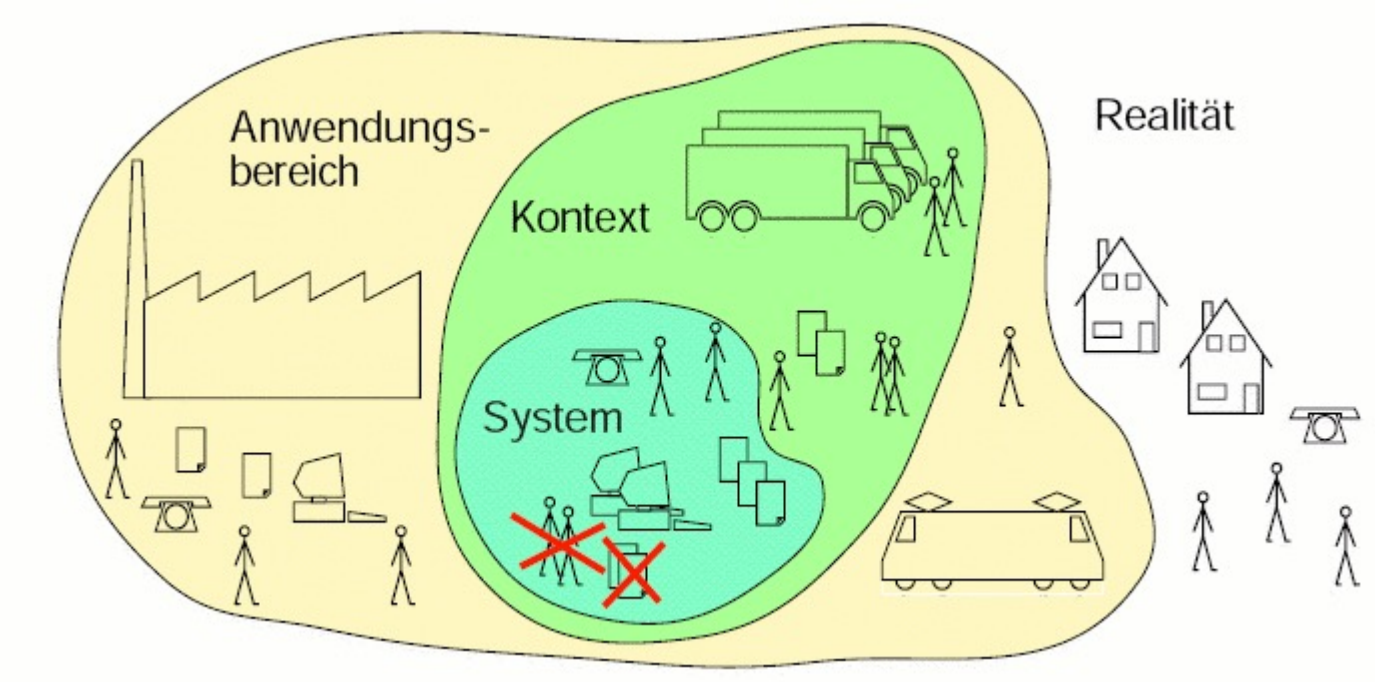
Requirements Engineering bildet Modelle eines Ausschnitts der *Realität*.

Requirements Engineering



Systeme sind daher immer in einen *Kontext* eingebettet, der den direkt für den Entwurf des Systems relevanten Bestandteil der Realität beschreibt.

Requirements Engineering



Requirements Engineering legt die Grenzen des *Systems* gegenüber dem *Kontext* fest.

Requirements Engineering

Unterschiedliche, aus einander abgeleitete, Betrachtungsebenen

Anforderung aus der *Realität*:
Auf dem bestehenden Schiennenetz sollen mehr Leute transportiert werden.

Daraus abgeleitete Anforderung an das *System*:
Die Minimaldistanz zwischen zwei Zügen ist immer größer als der maximale Bremsweg des nachfolgenden Zuges.

Daraus abgeleitete Anforderung an das umzusetzende Informationssystem ("*die Software*"):
Der maximale Bremsweg muss alle 100 ms neu berechnet werden.

Was heißt Planung?

Wie kann man die formalisierte Beschreibung der Anforderungen in einen Gesamtprozess eingliedern, der ein Projekt zur Erzeugung eines Informationssystems *insgesamt* beschreibt?

Konzept des *Systems Designs* / *Software Engineering.*

Wasserfall-modell

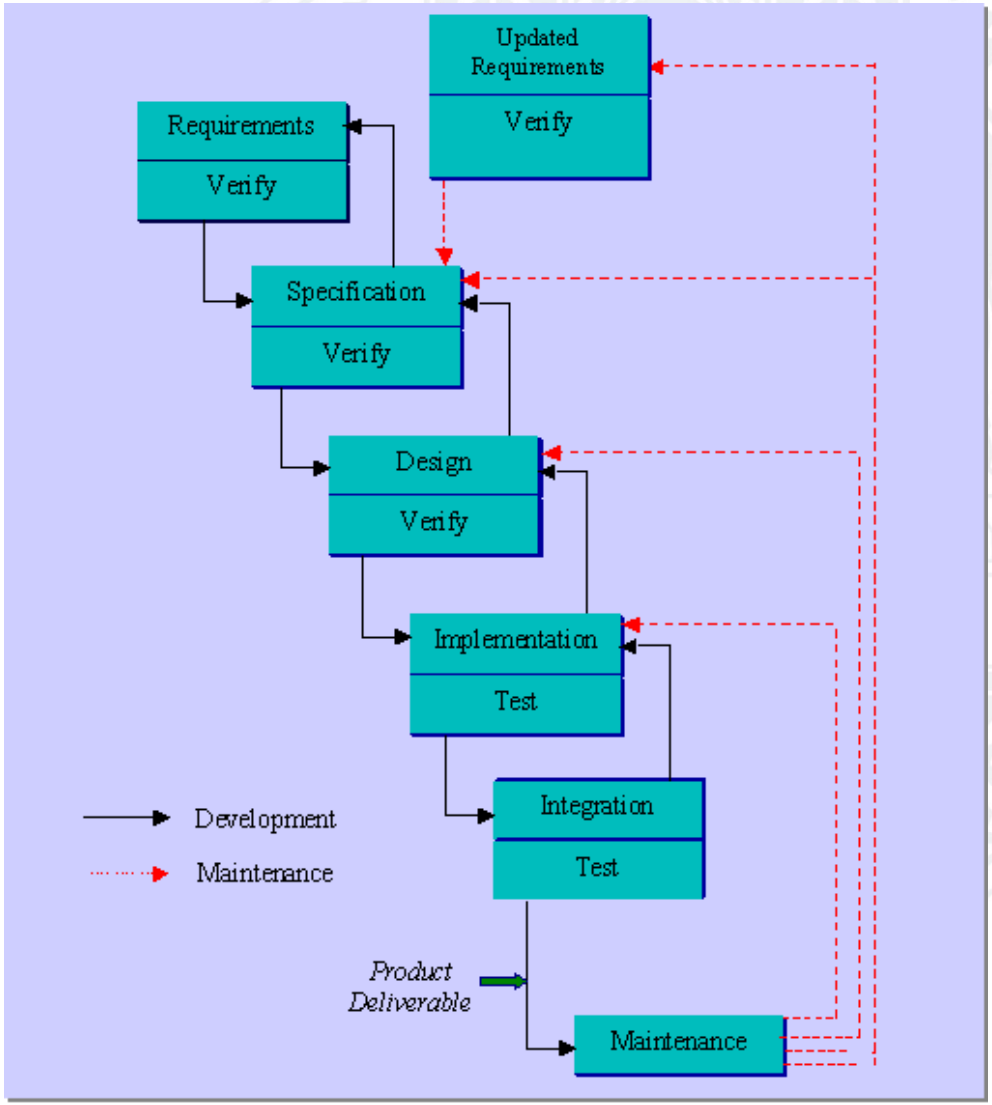


Fig. 1.2 - Schematic illustrating the Waterfall Model

One consequence

4.2 Operator communication

4.2.1 SINTRAN commands

A CC program has to assign the operator specified files on the prescribed I/O-units before initiating the execution of the ORBCOR program:

	Terminal	IN/OUT	for off-line (PRDEV) execution
1:	OPERATOR-file	IN	
2:	TWISS-file	IN	
3:	HCORRECTOR-file	IN	
4:	VCORRECTOR-file	IN	
5:	PRINT-file	OUT	
6:	ORBIT-file	IN	
7:	CALCORB-file	OUT	for CALC only
8:	GRAPHICS-file	OUT	on demand (not implemented yet)
9:	CORSTR-file	OUT	for GLBL & LOCL only
10:	RESULTS-file	OUT	

Eva Bozoki. ORBCOR. The new orbit correction system of the PS-complex. Cern, 1985. https://cds.cern.ch/record/162842/files/CM-P00059663.pdf

Agile Entwicklung

- Das Manifest
  - http://agilemanifesto.org/iso/de/manifesto.html
- 12 Prinzipien
- Viele spezifische Methodologien
  - an den Prinzipien basiert
  - die Prinzipien ausdehnen
- Die Ziele verstehen
  - dann spezifische Versionen praktisch anwenden

Manifest für Agile Softwareentwicklung

Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen.

Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

Manifest für Agile Softwareentwicklung

- Individuen und Interaktionen
  - mehr als Prozesse und Werkzeuge
- Funktionierende Software
  - mehr als umfassende Dokumentation
- Zusammenarbeit mit dem Kunden
  - mehr als Vertragsverhandlung
- Reagieren auf Veränderung
  - mehr als das Befolgen eines Plans

Prinzipien hinter dem Agilen Manifest

Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.

Prinzipien hinter dem Agilen Manifest

Heisse Anforderungsänderungen selbst spät in der Entwicklung willkommen.

Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.

Prinzipien hinter dem Agilen Manifest

Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.

Prinzipien hinter dem Agilen Manifest

Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.

Prinzipien hinter dem Agilen Manifest

Errichte Projekte rund um motivierte Individuen.

Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.

Prinzipien hinter dem Agilen Manifest

Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.

Prinzipien hinter dem Agilen Manifest

Funktionierende Software ist das wichtigste Fortschrittsmaß.

Prinzipien hinter dem Agilen Manifest

Agile Prozesse fördern nachhaltige Entwicklung.

Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.

Prinzipien hinter dem Agilen Manifest

Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.

Prinzipien hinter dem Agilen Manifest

Einfachheit – die Kunst, die Menge nicht getaner Arbeit zu maximieren – ist essenziell.

Prinzipien hinter dem Agilen Manifest

Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.

Prinzipien hinter dem Agilen Manifest

In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

Scrum

- Leichtgewichtiges Rahmenwerk
  - keine umfangsreiche Methodologie
- Zielgruppen
  - Menschen
  - Teams
  - Organisationen
- Ziel
  - Wert durch adaptive Lösungen für komplexe Probleme zu generieren

Basis für Scrum

- Empirie
  - Wissen aus Erfahrung gewonnen wird
  - Entscheidungen auf der Grundlage von Beobachtungen getroffen werden
- Lean Thinking
  - reduziert Verschwendung
  - fokussiert auf das Wesentliche
- Ansatz
  - iterativ
  - inkrementell

Scrum-Werte

- Commitment
- Fokus
- Offenheit
- Respekt
- Mut

Kommunikation zentral

Scrum-Elemente

- Scrum Team
  - DeveloperInnen
  - Product Owner:in
  - Scrum Master:in
- Scrum Events
  - Der Sprint
  - Sprint Planning
  - Daily Scrum
  - Sprint Review
  - Sprint Retrospective
- Definition of Done

Scrum

https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-German.pdf

https://scrumguides.org

Danke für dieses Semester!