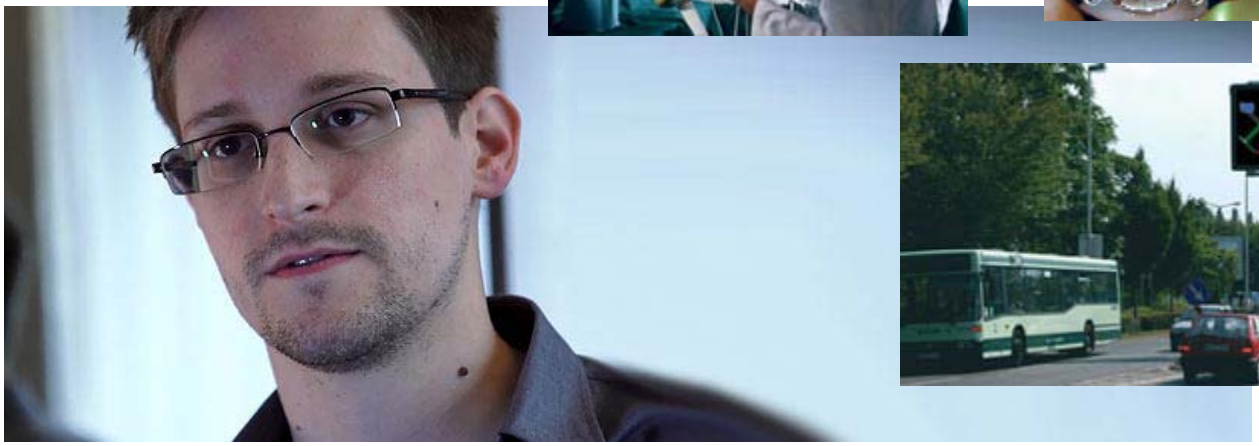
Einführung in die praktische Informatik
WS13/14

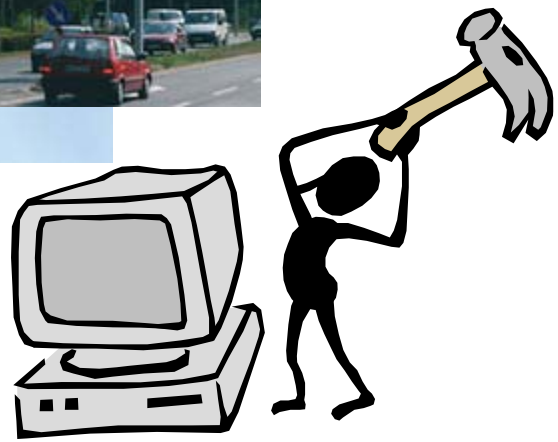
Barbara Paech, Tom-Michael Hesse

Institute of Computer Science
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
paech@informatik.uni-heidelberg.de

Bilder der Informatik?

http://www.informatikjahr.de





Mein Hintergrund

- **Software Engineering**
- Entwicklung
  - Qualitativ hochwertiger
  - Großer Programmsysteme
  - Mit vielen Beteiligten
  - Unter Kosten- und Zeiteinschränkungen
  - Nach ingenieurmäßigen Prinzipien
    - Planung
    - Arbeitsteilung
    - Methoden und Werkzeuge (Standardisierung, Qualität)
    - Modell

- Werdegang: Habilitation TU München, Abteilungsleitung FH IESE (Industrie), seit 10 Jahren Uni HD

Kommunikation

- Übungsorganisation: Tom-Michael Hesse, Gebäude 326, Raum 224
- TutorInnen: Studierende aus höheren Semestern
- Fragen / Anregungen sehr erwünscht !

- Meine Sprechstunde DO 16.00 -16.45 (nach der Vorlesung), Gebäude 326, Raum 208

- Vorlesung Dienstag, Donnerstag: Neue Konzepte
- Die Vorkenntnisse der HörerInnen sind sehr unterschiedlich => Nehmen Sie Rücksicht auf andere. Dinge, die Ihnen trivial erscheinen, sind es für viele andere nicht.
- Sie dürfen gerne Fragen stellen: Es gibt keine dummen Fragen.
- Üben Sie! Eine notwendige Voraussetzung für das Lernen der Programmierung ist, dass man es tut.

Übungsbetrieb

- Übung: Vertiefung der Vorlesung und Besprechung der Hausaufgaben
  - **Übungsbeginn: Montag 21. Oktober**
  - **DIESE WOCHE Rechnerübungen!**
- **Anmeldung zur Übung (Gruppeneinteilung) über E-Learning-System Moodle**
  - **http://elearning2.uni-heidelberg.de/course/view.php?id=3251**
  - **URZ-Account über CampusCard**
  - **ALLE anmelden**
  - Jetzt oder im Lauf des Semesters für die Klausur anmelden
  - Falls Sie die Klausurzulassung schon haben: registrieren, aber nicht notwendigerweise Übungsgruppe auswählen

Materialien

- Vorlesungsfolien und Übungsblätter verfügbar in Moodle
- Forum für Information, Diskussion
- Untergruppen für Übungsgruppen

- Bücher siehe Webseite

- Skript Bastian 2011
  - Wesentliche Konzepte gleich
  - Hier mehr Konzepte aus dem Software Engineering
  - Dort mehr Mathematisches, mehr Erklärung

Übungsaufgaben

- Übungsblätter (und zugehörige Vorlesungsfolien) verfügbar jeweils donnerstags ab 16 Uhr in Moodle
- Abgabe der Lösungen bis Freitag der nächsten Woche um 12.00
  - elektronisch in Moodle
  - Abgabe in 2-3er Gruppen möglich
    - ABER wer nur Übungsschein benötigt (keine Klausur), muss Übungsblätter allein bearbeiten und das bei uns ankündigen
- Achtung: 1.Blatt schon diese Woche (Abgabe am 25.10)
  - Nur halbe Menge der Aufgaben

Programmierung

- Programmierung ist wichtiger Bestandteil der Vorlesung, aber nicht einziger!

- Programmierkurs zusätzlich:
  - Pflicht für die Bachelor Angewandte Informatik
  - Für andere freiwillig zur Vertiefung möglich (wenn Platz ist)

- Programmierberatung
  - Zeiten ???

Leistungsnachweis

- Erfolgreiche Übung
  - Mind. 15% der Punkte bei JEDEM Übungsblatt
  - Mind. 50% der Punkte insgesamt
  - Mind. 2 Präsentationen in den Übungen (ansonsten keine Anwesenheitskontrolle)
- Klausur: Februar
  - Erfolgreiche Übung ist Voraussetzung
  - Klausurzulassung aus Vorjahren wird anerkannt. Schriftliche Bestätigung bei Herrn Hesse abgeben.

Klausurregeln

- Pro Jahr (also pro Vorlesung IPI) 1 Prüfungsversuch mit 2 Klausuren. Dabei gilt
  - 1. Klausur am Ende der Vorlesungszeit
  - 2. Klausur mind 5. Wochen nach der ersten und spätestens vor Beginn der nächsten Vorlesungszeit
  - Evtl. 3. Klausur: mindestens 1 Woche nach der zweiten Klausur und spätestens 2 Wochen nach Beginn der nächsten Vorlesungszeit
- An der zweiten Klausur kann nur teilnehmen, wer in der ersten Klausur keinen Erfolg hatte (also krank oder nicht bestanden)
- An der dritten Klausur kann nur teilnehmen, wer keinen Erfolg hatte und für eine der beiden ersten Klausuren mindestens ein Attest hat.
- Wer bei der dritten Klausur immer noch krank ist, hat den Prüfungsversuch verloren (kann das aber dann wenn nötig bei Ausnahmeantrag geltend machen).
- Die Note ergibt sich bei Bestehen aus der bestandenen Klausur (also kein Malus für mehrfache Teilnahme)

Netzwerk

- Die Informatik trägt wesentlich zur rasanten Entwicklung unserer Gesellschaft und ihrer Technologien bei, muss sich daher ständig an ihrem Nutzen messen und sehr bedacht ihre eigene Entwicklung fördern.
- Sie werden ein aktiver Bestandteil dieser Entwicklung sein.
- Gemeinsam mit Anderen als Teil von Arbeitsgruppen, Teams, Projekten,.. – gleich ob in Kombination mit Wirtschaftswissenschaften, Biologie, Medien,… – sei es im ersten Semester, später in der Praxis oder in der Forschung.
- Besser, gleich mit netzwerken anfangen…
- Gesellschaft für Informatik
- www.wir-sind-informatik.de

Ziele der Lehrveranstaltung

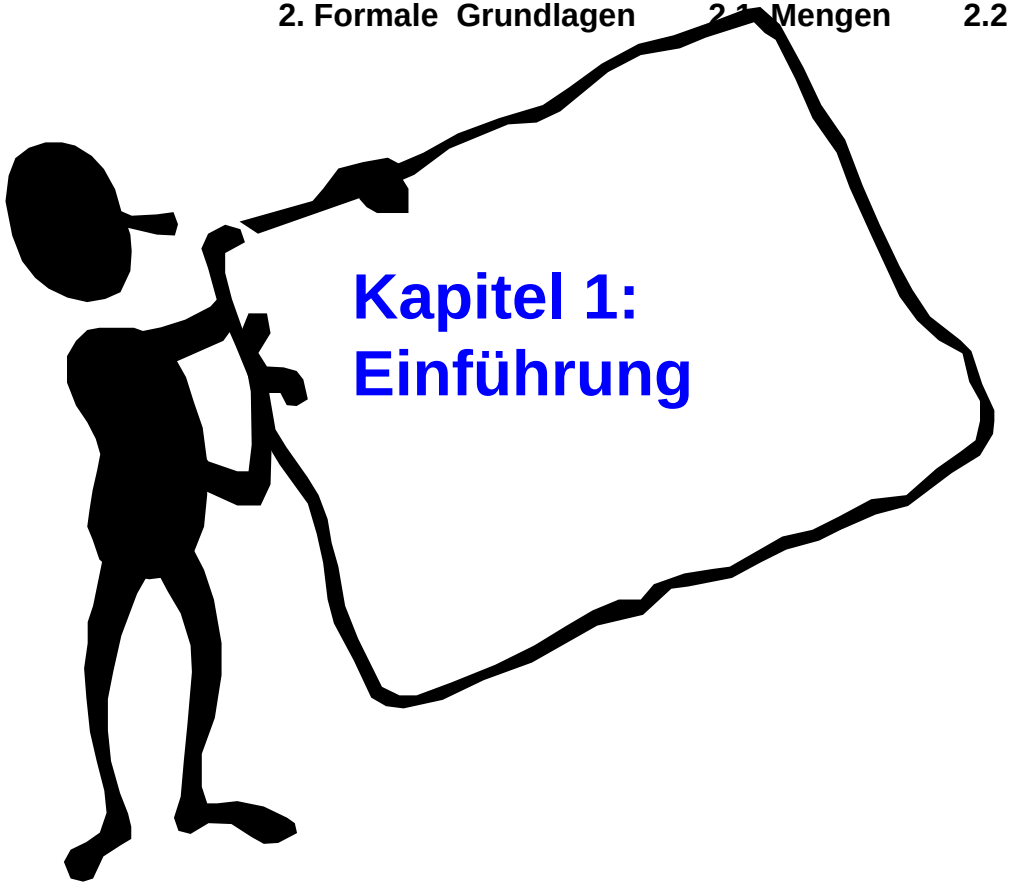
- Grundlagen lernen
  - Begriffe, Denkweisen
  - Algorithmen-Entwicklung
  - Programmierung im Kleinen und Qualitätssicherung dafür

- Spaß haben
  - Teamarbeit (sehr wichtig im Studium!)
  - Computer „beherrschen“
  - „Erholung“ von Mathematik

Überblick über diese Vorlesung

1. Einführung
2. Grundlagen der Formalisierung
3. Funktionale Programmierung
4. Imperative Programmierung
5. Komplexe Datenstrukturen
6. Objektorientierte Programmierung
7. Beschreibungstechniken
8. Qualitätssicherung
9. Grundlegende Algorithmen

- Themen werden nicht immer sequentiell durchlaufen



Kapitel 1: Einführung

1. Überblick Einführung

- Lernziel: Einführung in und Überblick über die Softwareentwicklung (Programmierung) im Kleinen

- **1.1. Was ist Informatik?**
- **1.2. Was sind die Aufgaben von InformatikerInnen?**
- **1.3. Was sind die Grundkonzepte der Informatik?**
- 1.4. Überblick über die Teilgebiete der Informatik

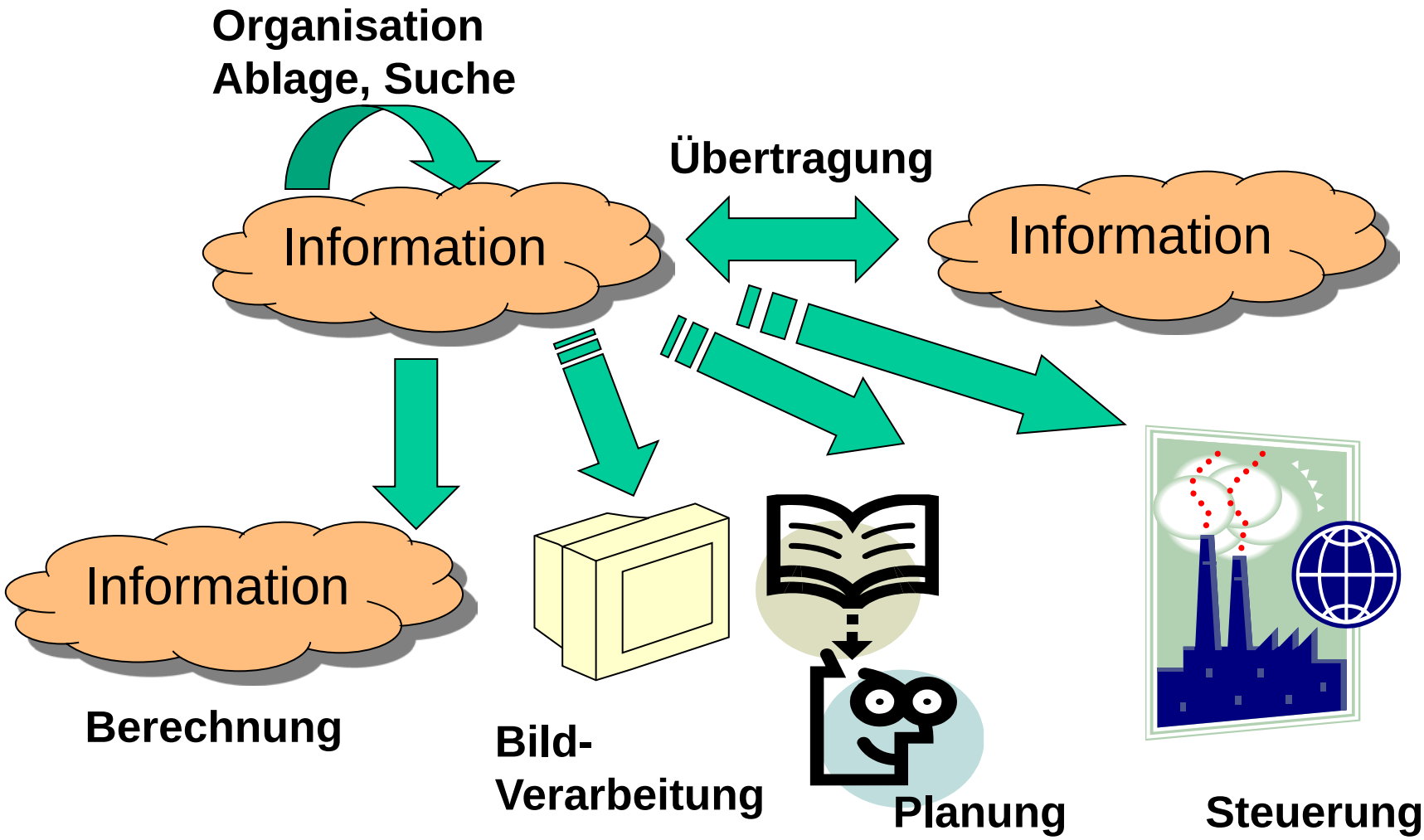
1.1. Was ist Information?

- Daten vs. Information vs. Wissen
  - Teil der Kommunikation
  - Daten beschreiben Fakten
    - Heute ist der 15. Oktober 2013
  - Information sind die Daten zusammen mit einer Bedeutung
    - Bedeutung für mich: Heute halte ich die erste Vorlesung IPI im WS13/14
  - Wissen ist über die Zeit angehäufte und geordnete Information
    - Mein Wissen: was möchte ich Ihnen im Lauf des Semesters beibringen

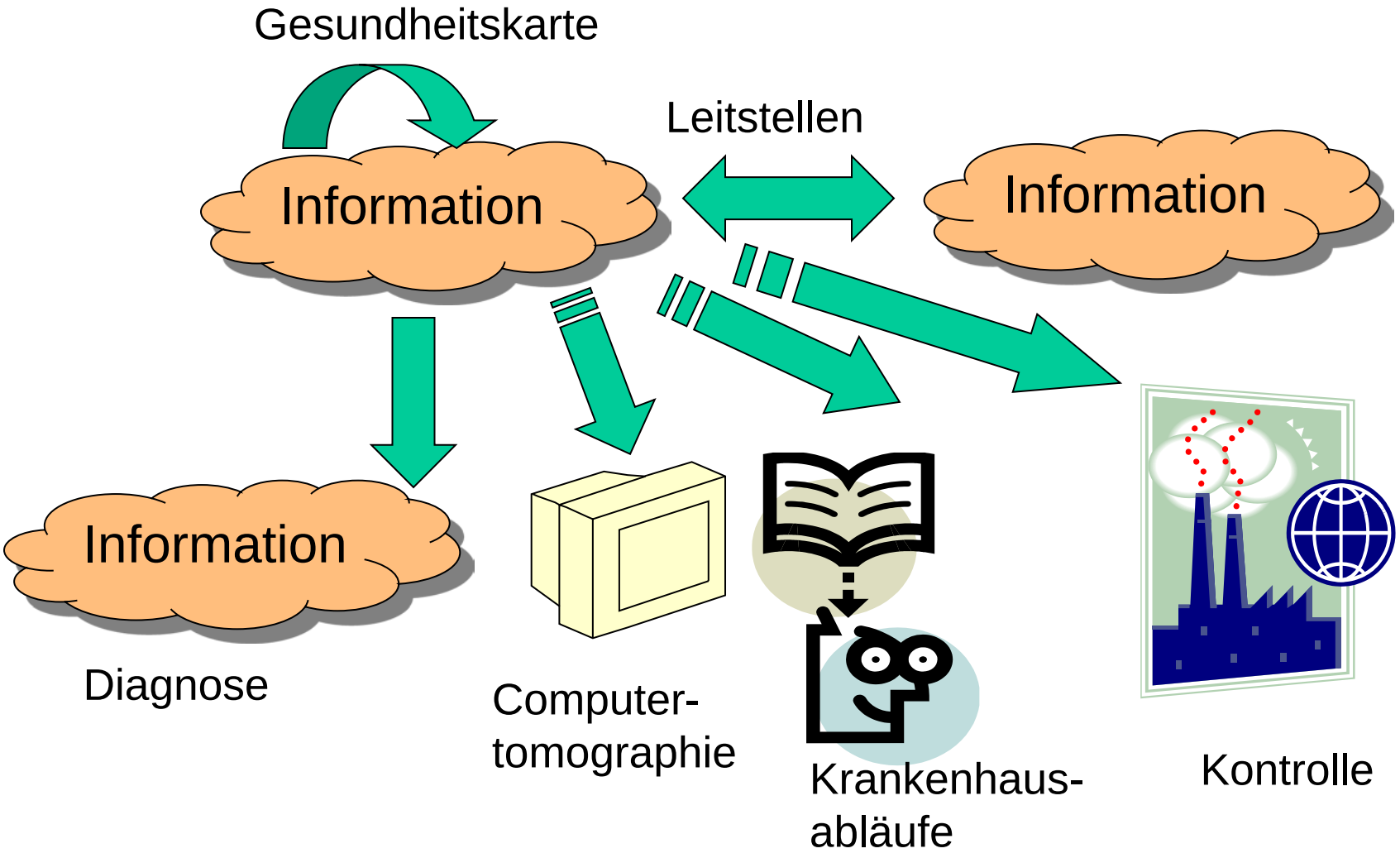
1.1. Was kann man mit Information machen?

- Informatik (engl. Computer Science)
  - Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von **Informationen**, besonders der automatischen Verarbeitung mithilfe von Computern [Schülerduden]
  - Auch Technik, Anwendung von …. [Broy]

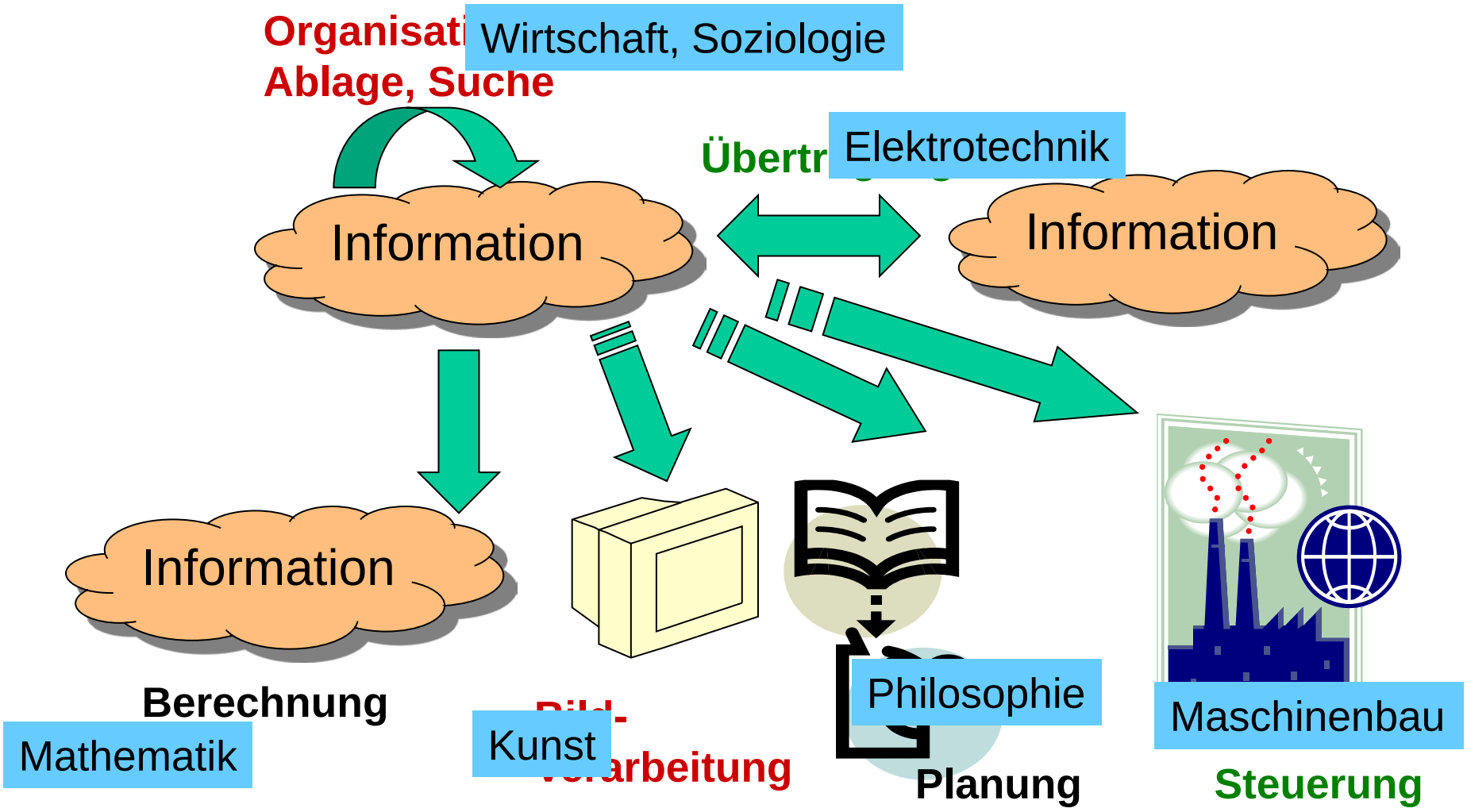
1.1. Phänomene der Informatik



1.1. Beispiel Gesundheit



1.1. Berühren fast alle anderen Wissenschaften



1.1. Was ist neu an der Informatik?

- Jeanette Wing: Denken in (Automatisierung) von Abläufen
  - Computational Thinking, Abstraction of Automation
    - Schulranzenpacken: Caching
    - Suche nach verlegten Dingen: Backtracking
    - Entscheidung zwischen Miete oder Kauf von Skis: Online Algorithmus
    - Entscheidung zwischen Schlangen an der Kasse (Performanzmodellierung in Multiserver-Systemen)

1.1. Was ist Gegenstand der Informatik? (1)

- System
  - Menge von Elementen, zwischen denen bestimmte Beziehungen bestehen [Duden 1974]

- Wissenschaften, die sich mit Systemen beschäftigen:
  - Naturwissenschaften: natürliche Systeme (z.B. Mensch, Bienenstock, Kosmos)
  - Geistes- und Sozialwissenschaften: soziale Systeme (z.B. Nation, Hochbegabte, Unternehmen)
  - Ingenieurwissenschaften: künstliche Systeme (z.B. Auto, Brücke, Informatiksystem)

1.1. Was ist Gegenstand der Informatik? (2)

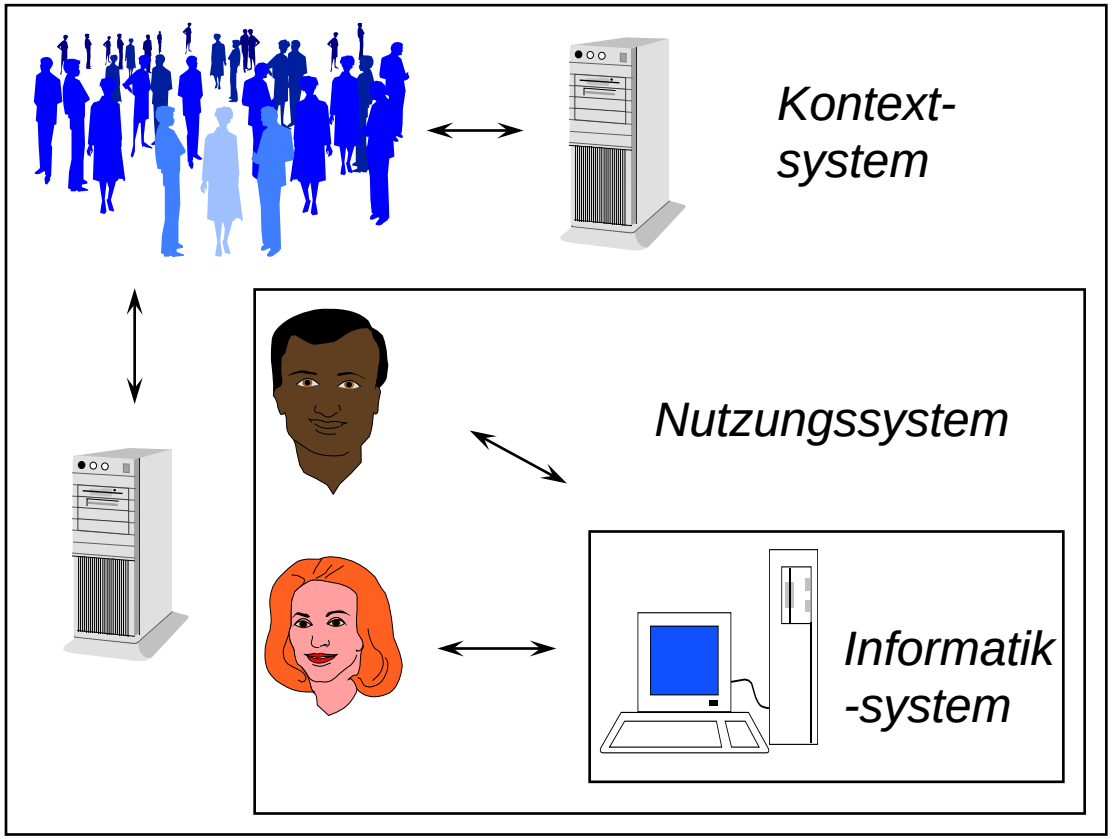
- Informatiksystem
  - Elemente heißen Komponenten und bestehen aus Hardware und Software
- Typische Informatiksysteme
  - Berechnungssysteme (z.B. Wettervorhersage, Informatik: Programmübersetzung)
  - Interaktive Systeme (Daten/ Informations/ Wissensaufnahme und –bereitstellung) (z.B. SAP, online-Banking)
  - Prozessüberwachungssysteme (z.B. Verkehrsflusskontrolle, chem. Prozesse; Informatik: Betriebssystem)
  - Eingebettete Systeme (z.B. Waschmaschinensteuerung, Roboter)
  - Kommunikationssysteme (z.B. Telefonvermittlung)

1.1. Informatik als Wissenschaft

	Gegenstand	Ziel	Beispiele
Geisteswis-senschaft	Kulturell geistige Schöpfung	Erkennen, Handeln	Kunst, Religion, Recht
Humanwis-senschaft	Menschen	Erkennen, Umsetzung	Soziologie, Medizin
Naturwissen-schaft	Natur	Erkennen	Physik, Chemie, Biologie
Ingenieur-wissenschaft	Technische Produkte	Umsetzung	Maschinenbau Elektrotechnik
Strukturwis-senschaft	Methoden zur Erforschung	Erkennen Umsetzung	Mathematik
Informatik	Information und Informatik-Systeme und Methoden	Erkennen, Umsetzung	

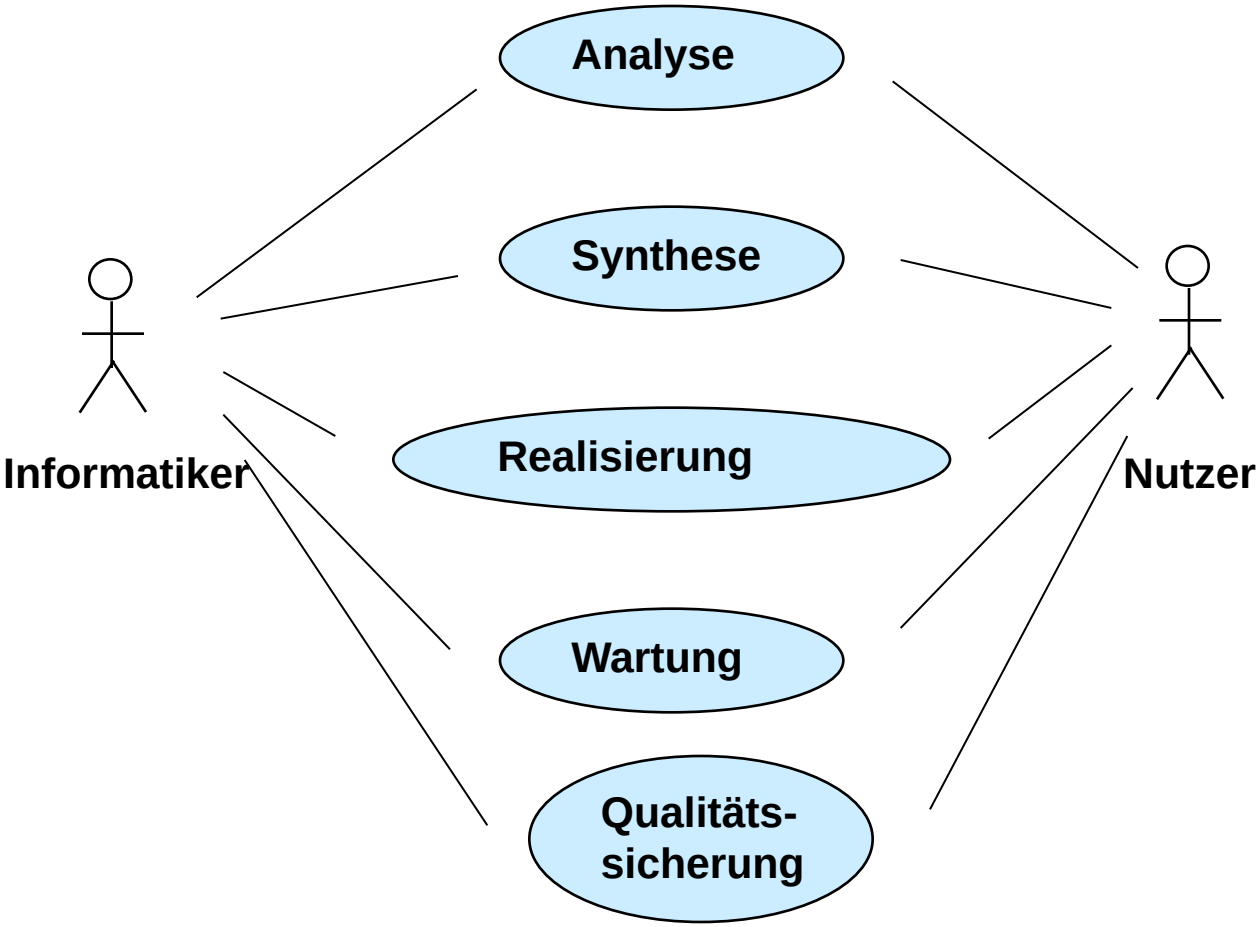
[Paech2000]

1.1. Gestaltungsbereiche der Informatik



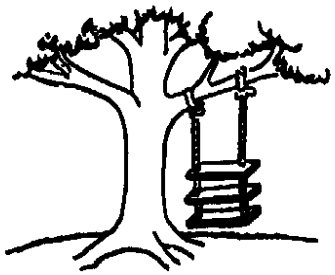
ALLE 3 Systeme werden durch Informa-tiker-Innen gestaltet

1.2. Was sind die Aufgaben von InformatikerInnen?

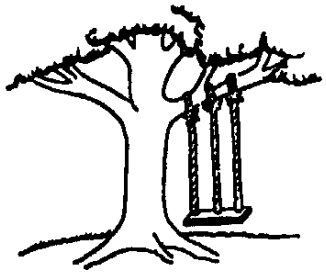


- Das Verständnis von Problemen aus der Realität
- Realität ist dabei ein
  - soziales System
  - natürliches System
  - künstliches System
- Achtung: InformatikerInnen bearbeiten alle Systeme, haben aber immer künstliches System als Lösung im Kopf (Hammer/Nagel)
- Verwendung von Modellen
  - In der Vorlesung: Actor – Task – Diagramm, Entity – Relationship – Diagramm
- Ergebnis: Anforderungsspezifikation

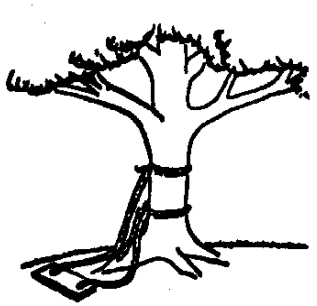
1.2. Analyse: Gemeinsames Verständnis nötig!



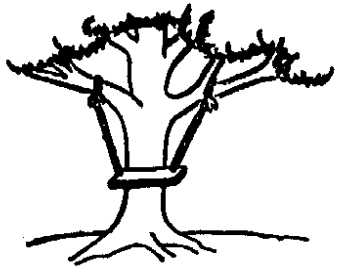
As proposed by the project sponsor



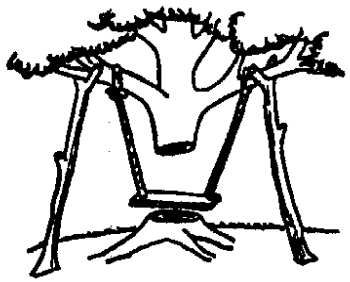
As specified in the project request



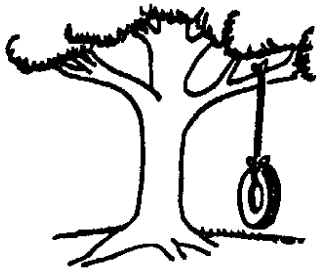
As designed by the senior analyst



As produced by the programmers



As installed at the user's site



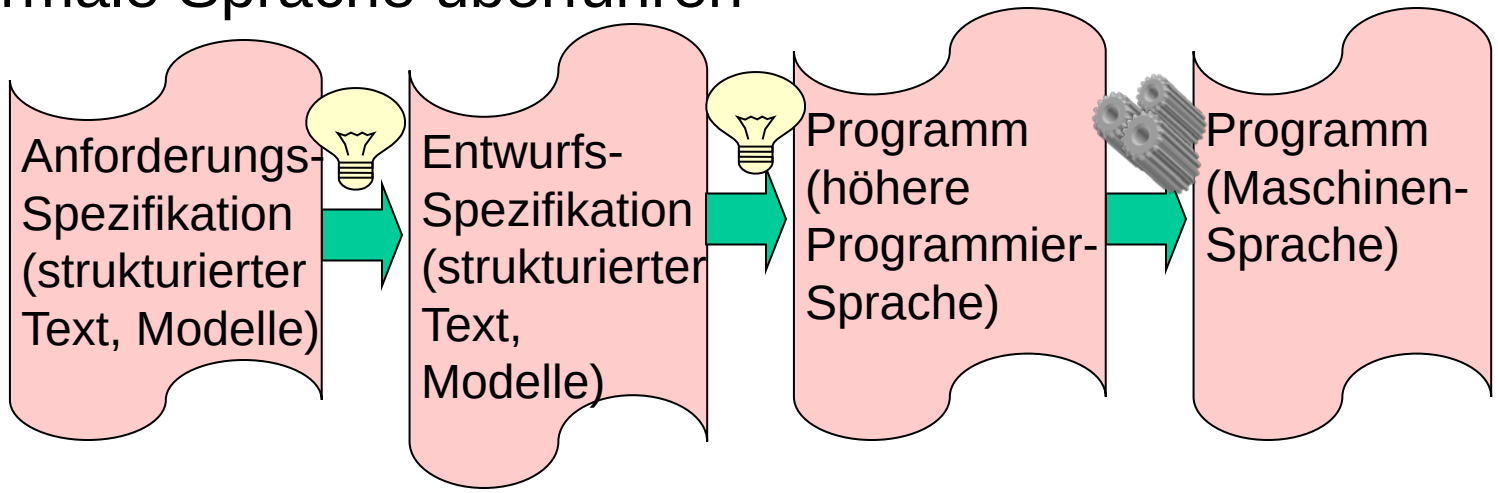
What the user wanted

1.2. Synthese (Design)

- Die systematische Entwicklung einer Lösung des Problems (umgesetzt durch ein Informatiksystem)
  - Ggf. Verwendung von (existierenden / neuen) Hardware-und Softwarebausteinen
- Wahl der Technologie
- Komponenten: Benutzungsschnittstelle, Datenbasis, Anmeldungs-, Auswertungskomponente
- Feinentwurf
  - Schnittstellen der Komponenten
  - Funktionen und Datenstrukturen, z.B. Sortieralgorithmen, Listen oder Bäume
  - Verwendet Modelle (in der Vorlesung: Klassendiagram und Sequenzdiagram)
- Ergebnis: Entwurfsspezifikation

1.2. Formalisierung

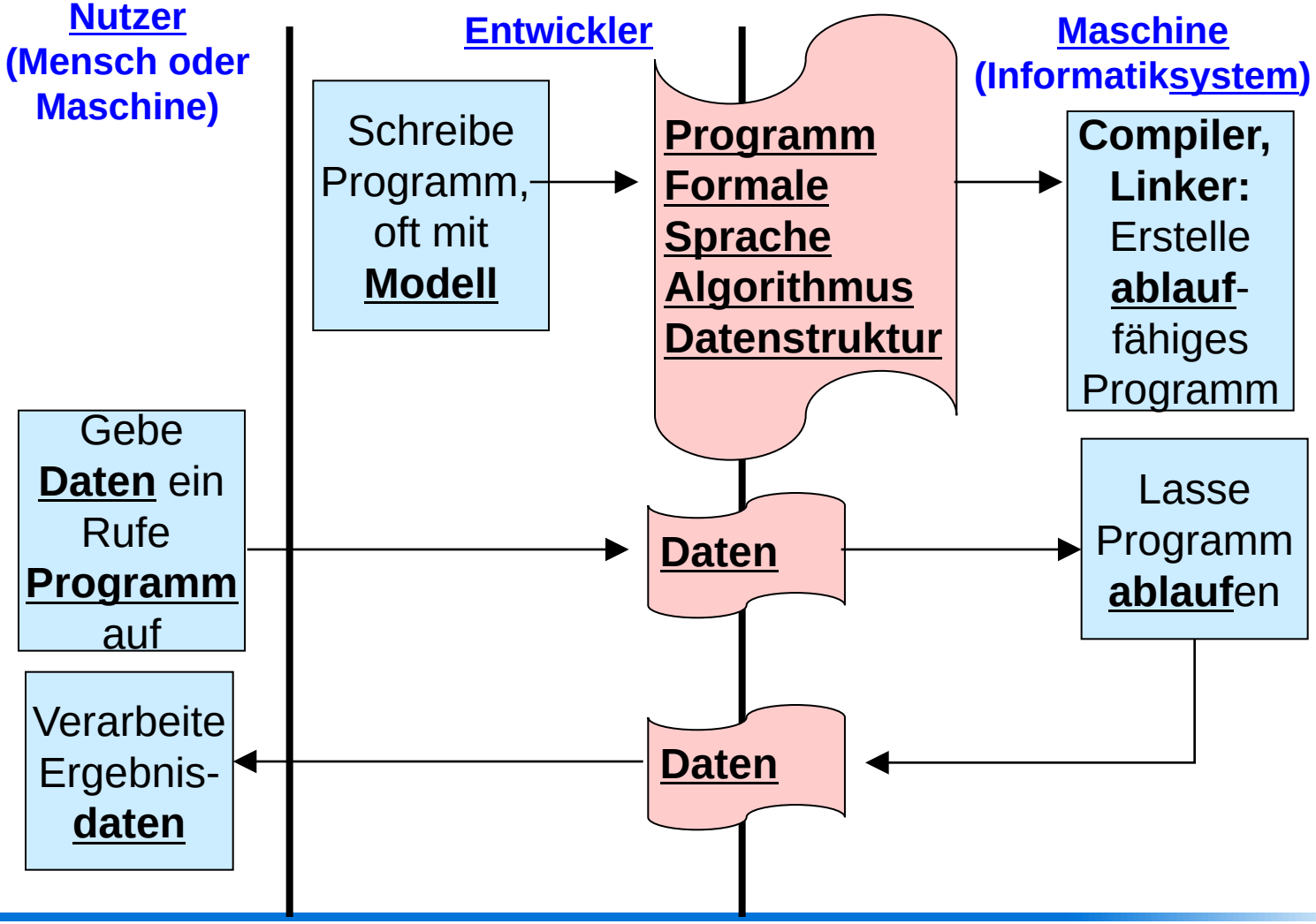
- Formalisierung ist ein wichtiger Bestandteil der Entwicklung
- Maschine (Computer) „versteht nur“ künstliche Sprache (Maschinensprache, binär)
- EntwicklerInnen müssen ihr Lösungsverständnis in eine formale Sprache überführen



1.2. Realisierung

- Die Konstruktion der Lösung
  - In Form eines Informatiksystems (Veränderung der Realität!)
- Wahl der Programmiersprache
- Programmieren: Editieren – Compilieren / Linken – Ausführen - Fehlerbehebung
- Ergebnis: Software (Code, Daten und Dokumentation)
- Entwicklung: Der Weg von der Analyse über die Synthese und Realisierung zum Informatiksystem

1.2. Programmierung



1.2. Werkzeuge zur Programmierung

- Editor: Erstellung des Programmtextes (Quelltext)
- Compiler: übersetzt menschenlesbares Programm in Maschinensprache
- Linker: Bindet Programmteile zusammen
- Weitere Möglichkeiten: Interpreter, virtuelle Maschine

- Integrierte Entwicklungsumgebung (Integrated Developer Environment, IDE)
  - Integriert Editor, Compiler und Linker und Ausführung
  - Verwaltet Menge von Programmen
  - Editor: kann einfache Syntaxfehler erkennen

[Brügge03]

1.2. Wartung und Qualitätssicherung

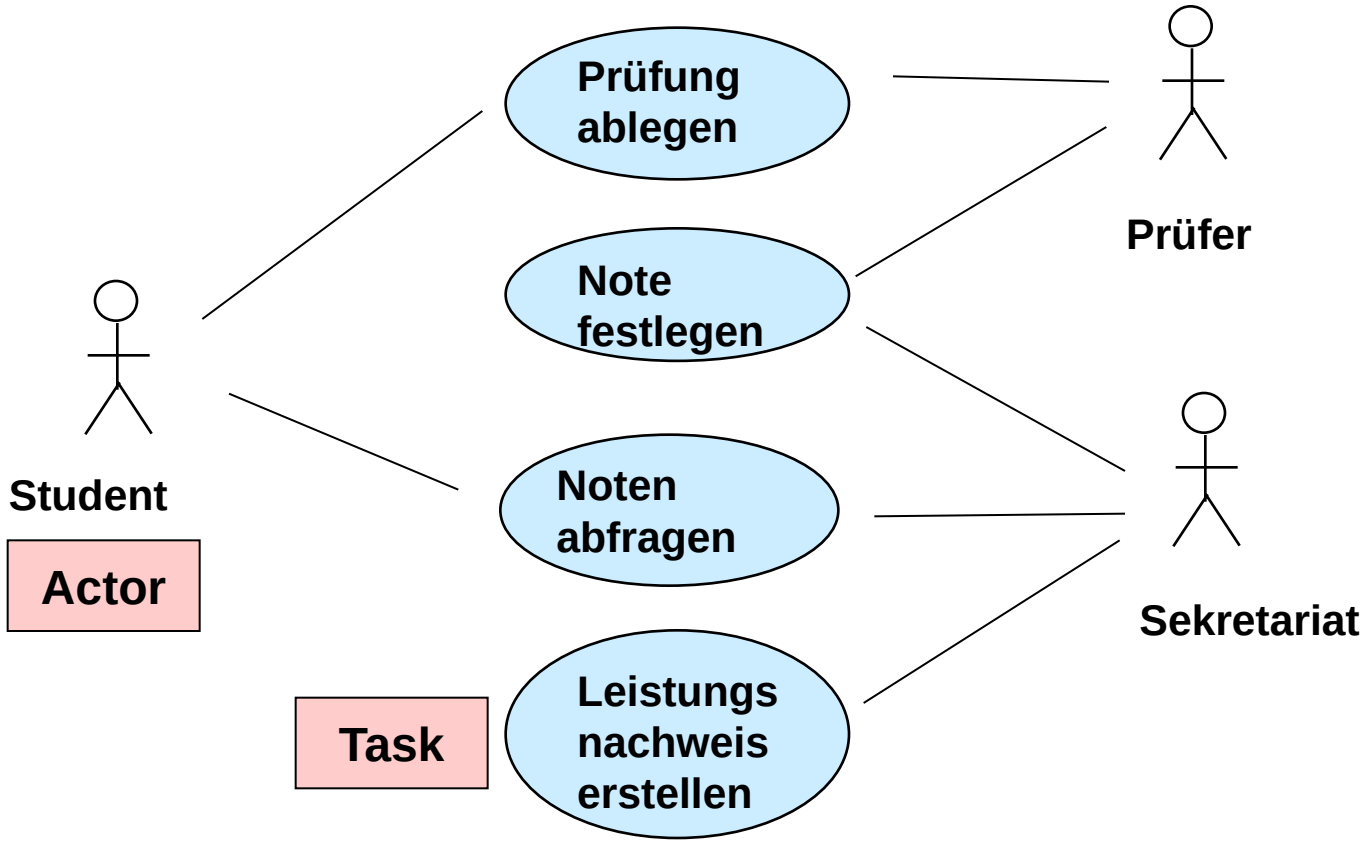
- Wartung: Die Bereitstellung und Betreuung des Informatiksystems über die gesamte Lebenszeit
  - Anpassung an neue Technologie
  - Anpassung an neue organisatorische Prozesse (z.B. Vorgaben aus dem Ministerium)

- Qualitätssicherung: Sicherstellung der Qualität während Entwicklung und Wartung
  - Validierung: Rückkopplung mit dem Benutzer
  - Verifikation: Überprüfung der Software gegen die Anforderungsspezifikation
    - Statische Analyse: Inspektion, Metriken, Beweis
    - Dynamische Analyse: Testen

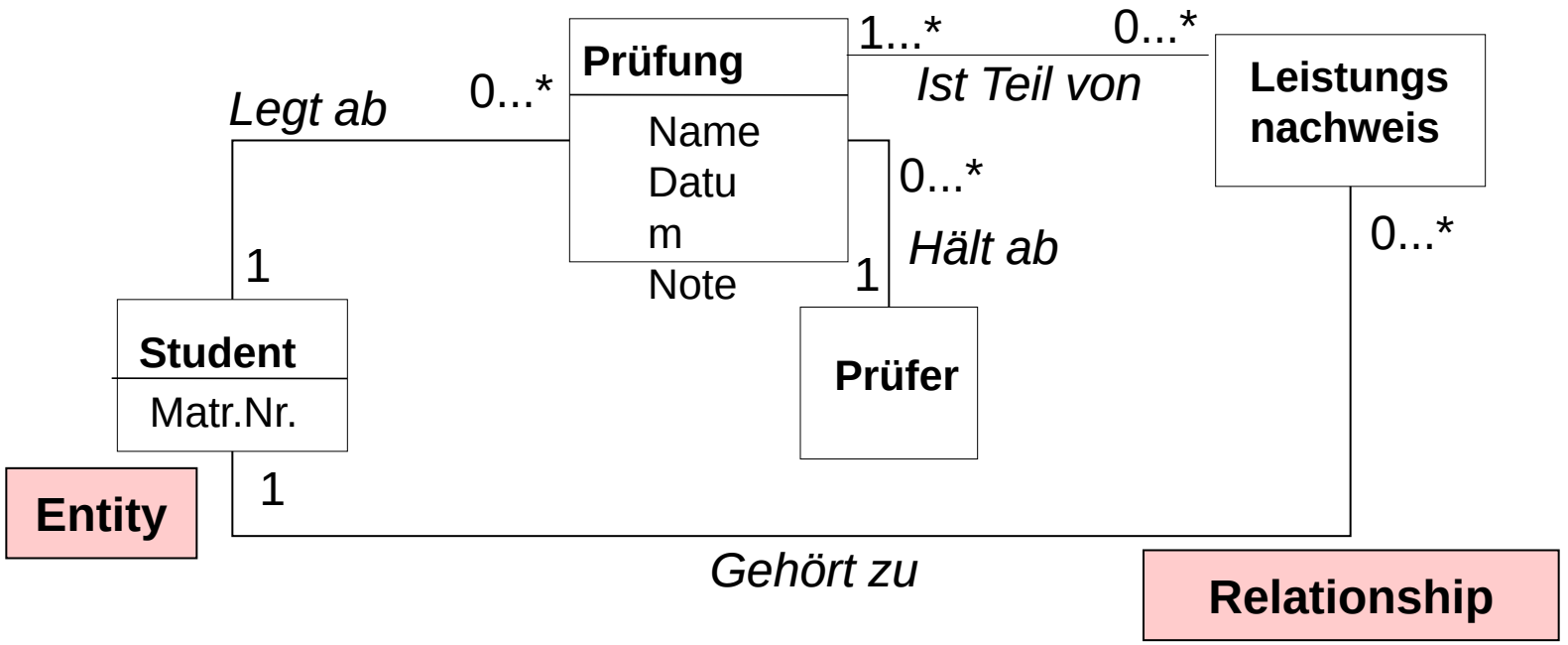
1.2. Beispiel Analyse (1)

- Problem:
  - Bachelorstudierende brauchen viele Leistungsnachweise, Bisherige Erstellung und Verwaltung von Scheinen im Sekretariat passt nicht mehr
- Analyse:
  - Wer ist beteiligt? (z.B. Studierende, PrüferInnen, Prüfungsausschuss, Prüfungsamt, Studiensekretariat)
  - Welche Aufgaben haben die Beteiligten, welche Informationen erzeugen / benötigen sie wann? (z.B. Prüfungsnote vergeben, Überblick am Semesterende, Zeugniserstellung)
  - Wie kann ein Informatiksystem helfen?
    - Z.B. Übersicht für Studierende
      - Eingabe von Studierendendaten und Prüfungsnoten
  - Welche Qualitätsvorgaben, z.B. Performanz, Sicherheit?

1.2. Aufgabenbeschreibung AT-Diagramm



1.2. Informationsbeschreibung ER-Diagram



Multiplizität

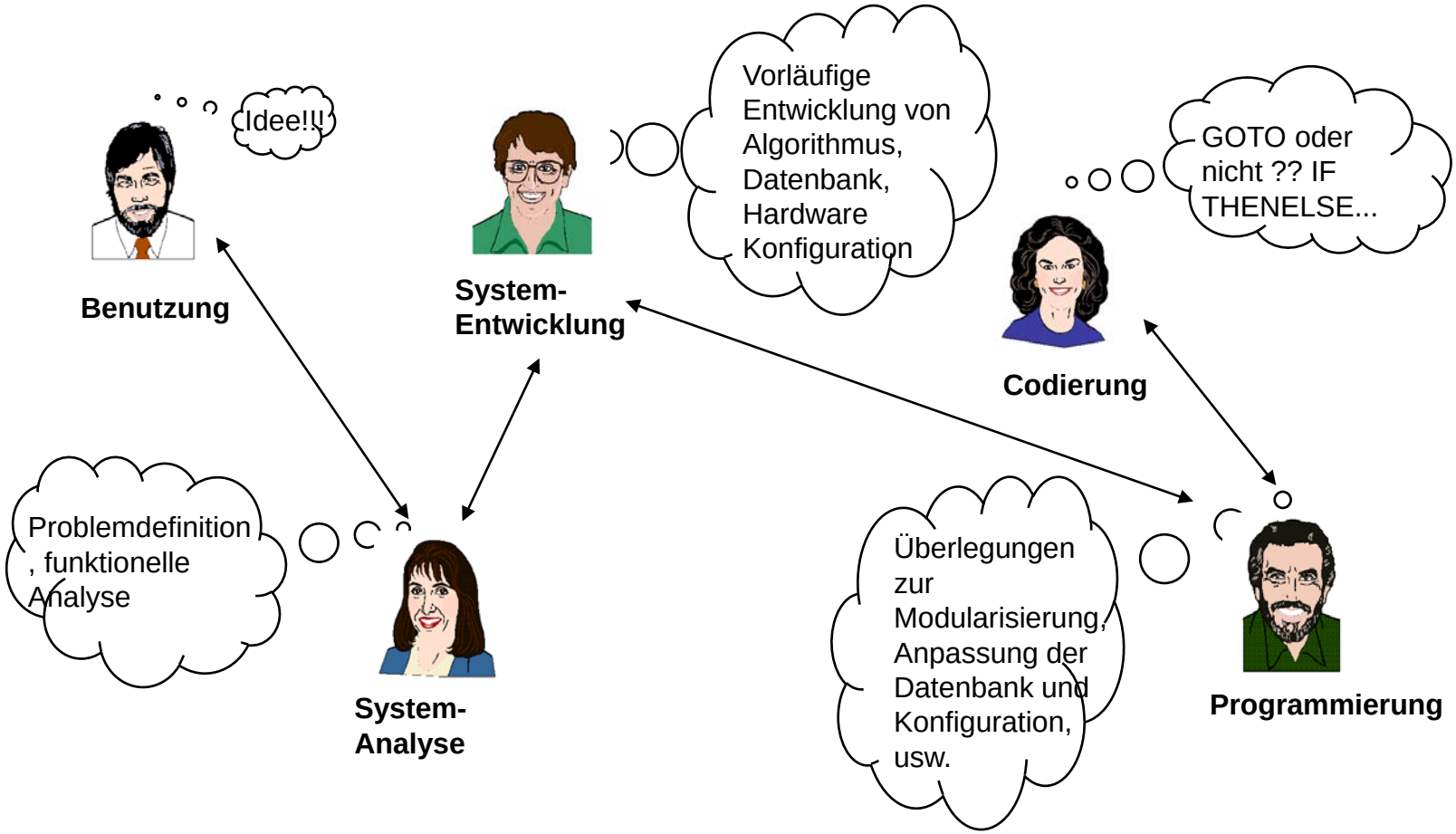
n..m (*n* bis *m* Instanzen)

Zulässig für ***n* und *m*** **Zahlenwerte** (auch 0)

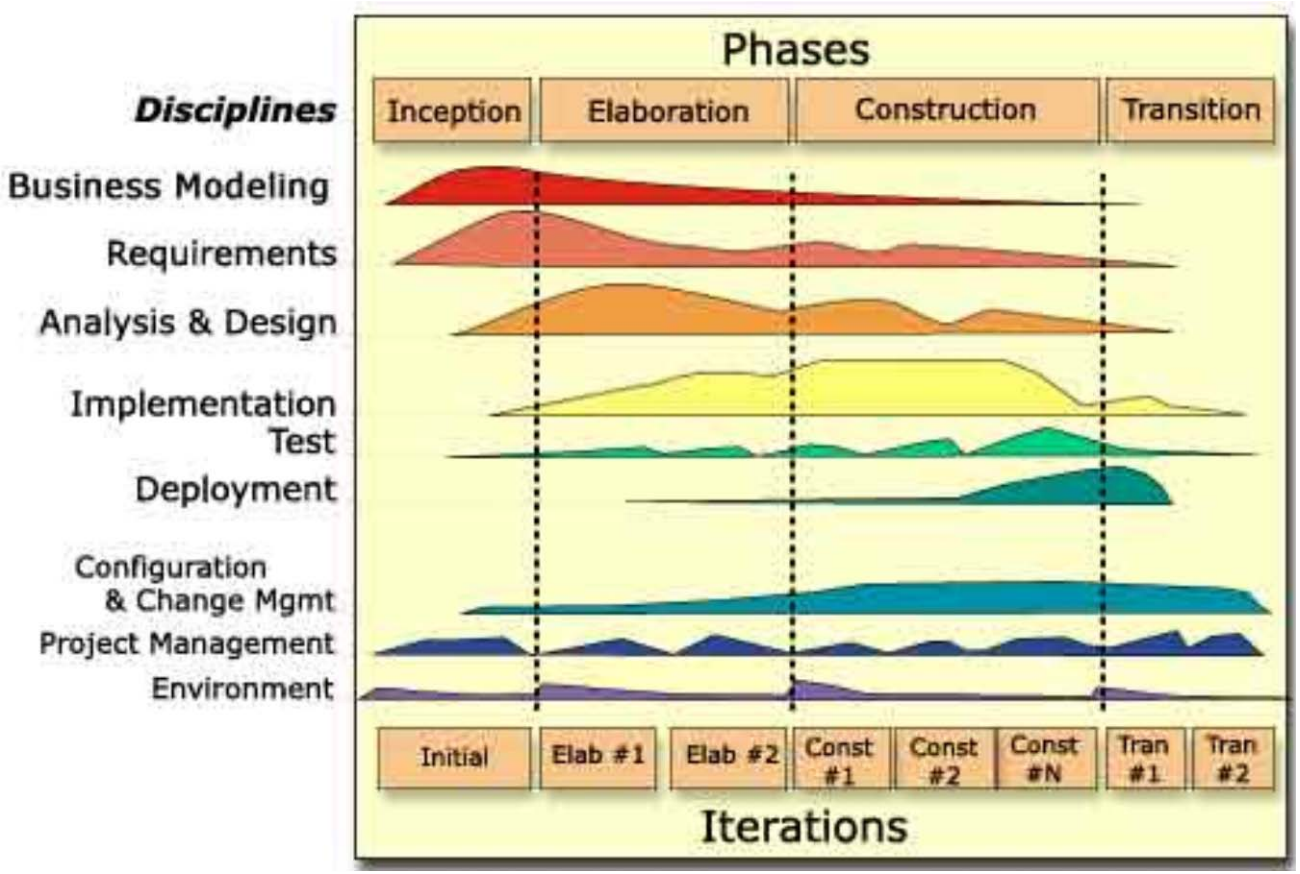
und * (d.h. beliebiger Wert, einschließlich 0)

Beispiel siehe Kapitel 2.1.

1.2. SW-Entwicklung früher



1.2. SW-Entwicklung heute : RUP



1.3. Grundkonzepte (Übersicht)

- System (siehe 1.1.)
- 1.3.1. Algorithmus, Programm
  - Terminierend
  - Deterministisch
  - Determiniert
  - Rekursiv
- 1.3.2. Datenstruktur
- 1.3.3. Prozess
- Ablaufbeschreibung
- 1.3.4. Maschine
- 1.3.5. Modell
  - Abstraktion
  - Transformation
  - Verifikation
  - Validation

1.3.1. Was ist ein Algorithmus?

- …eine Handlungsanweisung?
- …ein Weg, ein bestimmtes Ziel zu erreichen?
- …?

- …haben Daten etwas damit zu tun?
- …haben Informationen etwas damit zu tun?
- …?

1.3.1. Bekannte Algorithmen (1)

…aus dem Alltag

- Waschanleitung
  - Wäsche in die Trommel stecken, Waschmittelzugabe abhängig von Hersteller/Verschmutzung/Wasserhärte, Temperatur einstellen (30°/60°/90°), Startknopf drücken
- Spielanleitung
  - Dame: …
  - Mensch ärgere dich nicht: …
- Rezept
  - 3x täglich 3 Tropfen einnehmen

1.3.1. Bekannte Algorithmen (2)

…aus der Mathematik

- Multiplikation zweier ganzer Zahlen
  - **33 x 24**

    66

    132

    792
- Euklidischer Algorithmus zur Bestimmung des ggT
- Ziehen einer Wurzel
- …

1.3.1. Bekannte Algorithmen (3)

…aus der Betriebswirtschaft

- Sortieren einer unsortierten Kartei (Karteikarten)
  nach der Kundennummer
- Suchen der Artikelnummer
  zu einem Artikel
- Suchen aller Teile, die für die Produktion einer Maschine benötigt werden

1.3.1. Definition Algorithmus

- Ein Algorithmus ist ein Verfahren mit einer präzisen, endlichen Beschreibung unter Verwendung effektiver Verarbeitungsschritte [Broy]
  - z.B. Kochrezept ist endlich und effektiv, aber nicht präzise

- Präzise: formale Sprache
- Effektiv: tatsächlich ausführbar

- Funktionale Sicht: welche Aufgabenstellung bewältigt der Algorithmus (Eingabe, Ausgabe) ?
- Operationale Sicht: welche Schritte beinhaltet der Algorithmus (Kontrollfluss, elementare Schritte, verwendete Daten)?

1.3.1. Beispiel Algorithmus

- Aufgabenstellung: Kaufe Fahrrad ein, d.h. leere Eingabe, gekauftes Fahrrad als Ausgabe

1. Gehe zur Bank
2. Hole Geld
3. Kaufe Fahrrad ein
4. Fahre nach Hause

- Verarbeitungsschritte: 1.-4.

1.3.1. Beispiel Algorithmus terminiert nicht

1. Gehe zur Bank
2. Hole Geld
3. Solange Geld da ist, kaufe etwas ein
4. Falls Geld weg, gehe zu Schritt 1.

- Algorithmus terminiert nicht, d.h. er endet nicht für alle Schrittfolgen
- Verarbeitungsschritte: 1.-4., Fallunterscheidung („Falls“), Schleife („Solange“)

- Berechnungen sollen terminieren, Steuerungsalgorithmen sollen nicht terminieren!

1.3.1. Beispiel Algorithmus nicht deterministisch

1. Gehe zur Bank
2. Hole Geld
3. Kaufe Fahrrad ein
4. Bezahle mit Münzen oder Scheinen
5. Fahre nach Hause

- Algorithmus ist nicht deterministisch, d.h. es besteht Wahlfreiheit bei der Auswahl der Verarbeitungsschritte (insbesondere 4.)

1.3.1. Beispiel Algorithmus nicht determiniert

1. Gehe zur Bank
2. Hole Geld
3. Kaufe Lebensmittel oder Fahrrad
4. Fahre nach Hause

- Algorithmus ist nicht determiniert, d.h. das Ergebnis des Algorithmus ist nicht eindeutig bestimmt

- *Zusammenhang determiniert und deterministisch?*

1.3.1. Beispiel Algorithmus ist rekursiv

- Aufgabenstellung: Eingabe zwei natürliche Zahlen a,b mit a>0,b>0; Ausgabe: größter gemeinsame Teiler von a und b ggT(a,b)

- Algorithmus ggT nach Euklid

1. Falls a= b: Ergebnis = a
2. Falls a<b: Ergebnis = ggT(a, b-a)
3. Falls a>b: Ergebnis = ggT(a-b,b)

- Verarbeitungsschritte: -, <, =
- Algorithmus ist rekursiv, d.h. das Ergebnis wird erreicht durch Anwendung des gleichen Algorithmus auf geänderte Eingaben

- *Terminiert der Algorithmus? Warum?*

1.3.1. Wichtige Eigenschaften eines Algorithmus

- Korrektheit: nützlich, erfüllt die Vorgaben
- Aufschreibbarkeit, Vollständigkeit, Eindeutigkeit, Ausführbarkeit: durch den Aktor (Mensch oder Maschine) umsetzbar
- Effizienz: möglichst wenig Ressourcen (der Aktoren)
  - Speicher, Ausführungszeit

1.3.1. Was ist ein Programm?

- Ein Programm ist die Beschreibung eines Algorithmus in einer für den Computer verständlichen formalen Sprache.

Kapitel 2: Grundlagen der Formalisierung

2. Überblick Formale Grundlagen

- Lernziel: gemeinsame Basis für Formalisierung schaffen

- **2.1. Wdh. Mengen**
- **2.2. Formale Sprache**
- 2.2. Induktion
- 2.3. Aussagenlogik
- 2.4. Relationen und Graphen

Ziele des Kapitel 2

- Wiederholung wichtiger mathematische Konzepte zu Mengen
- Einführung von Induktion und Aussagenlogik als wichtige Konzepte aus der Mathematik
- Einführung formaler Grundlagen aus der theoretischen Informatik: Formale Sprachen und Graphen

2.1. Wdh Mengen: elementare Notation

- $a \in M$ a ist Element der Menge *M*
- $a \notin M$ a ist kein Element von *M*
- $\varnothing$ leere Menge
- $|M|$ Anzahl der Elemente einer endlichen Menge *M*
- $M = \{\ldots\}$ Definition einer Menge durch Angabe ihrer Elemente.
- $M = \{ a: \ldots\}$ Definition einer Menge durch die Eigenschaft ihrer Elemente: *M* ist die Menge aller *a*, für die gilt ...
- Beispiele
  - $\mathbb{Z}$ Menge der ganzen Zahlen
  - $\mathbb{N}$ Menge der natürlichen Zahlen
  - $\mathbb{N} = \{0, 1, 2, \ldots\}$
  - $\mathbb{N} = \{z \in \mathbb{Z}: \ z \geq 0\}$

2.1. Wdh. Mengen: Mengenoperationen

- $M \cup N$ Vereinigung
- $M \cap N$ Schnittmenge
- $M \subseteq N$ Teilmenge
- $M \subset N$ echte Teilmenge
- $M \setminus N = \{x: x \in M \text{ und } x \notin N\}$ Differenz

- Die Menge *P(M)* aller Teilmengen einer Menge M heißt **Potenzmenge**.
- Beispiel: M = {a,b,c}
- P({a,b,c}) =

  {{}, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c}}

2.1. Wdh. Mengen: Zerlegung

- M und N heißen disjunkt, wenn $M \cap N = \emptyset$
- Wenn $N \subseteq M$, dann heißt $M \setminus N$ Komplement von N bezüglich M
- Eine Aufteilung $M = M1 \cup M2 \cup \cdot \cdot \cdot \cup Mn =$ heißt Zerlegung oder Partition von M, wenn die Teilmengen Mi , i = 1, . . . , n, disjunkt sind.

2.1. Wdh. Mengen: Kartesisches Produkt

- Die Menge $M \times N = \{(x, y): x \in M , y \in N\}$ heißt das kartesische oder direkte Produkt der Mengen M und N.
- Ihre Elemente sind die geordneten Paare (x, y).

- Die Elemente (x1, x2, . . . , xn) des Produktes mehrerer Mengen $M1 \times M2 \times \cdots \times Mn$ heißen n-Tupel.

2.1. Wdh. Funktion

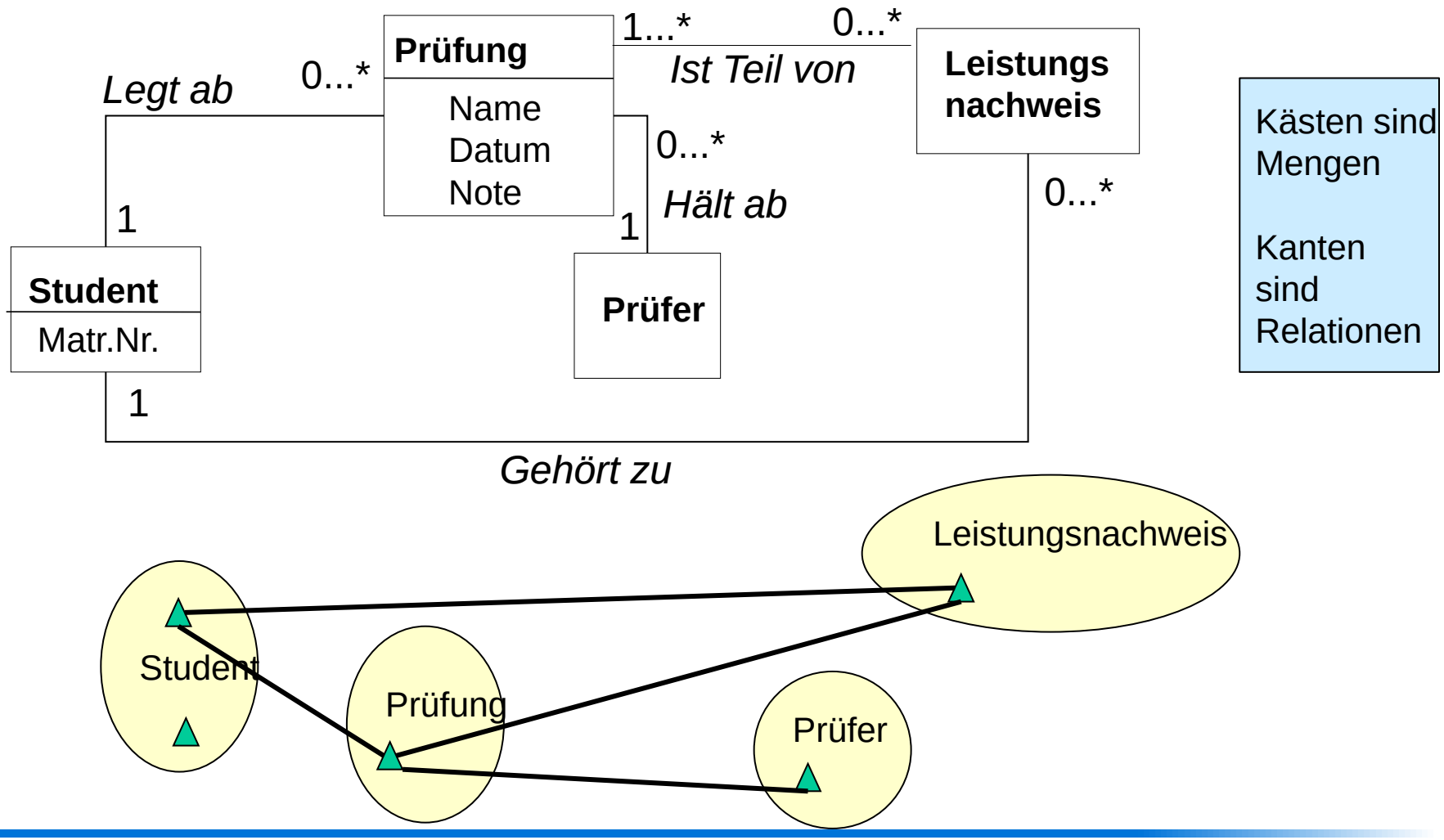
- **Relation:** Teilmenge eines kartesischen Produkt
  $R \subseteq D \times W$

  (Beispiel: D = Menge der Menschen, W = Menge der Haustiere, R = mag)

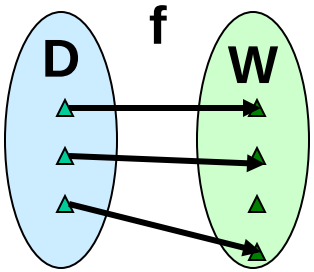
- **Funktion:** spezielle Relation **F**, so dass jedem Element aus dem Definitionsbereich **D** genau ein Element aus dem Wertebereich **W** zugeordnet wird

  (Beispiel: D = Menge der Studierenden, W= Menge der Menschen, F = Vater von)

1.2. Semantik ER-Diagram

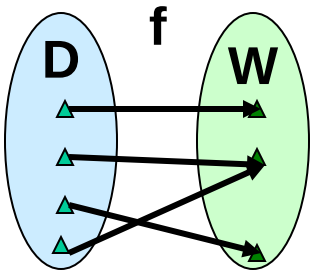


2.1. Wdh. Injektiv, surjektiv, bijektiv



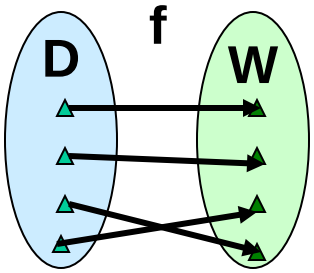
Injektiv,
Nicht surjektiv

- Injektiv: für alle d1,d2 $\in$ D gilt: f(d1) $\neq$ f(d2)



surjektiv,
Nicht injektiv

- Surjektiv: f(D) = W



bijektiv

- Bijektiv: injektiv und surjektiv

2.2. Zeichenketten

- Formale Sprachen basieren auf Zeichenketten, d.h. Menge von Worten eines Alphabets

- Zeichenketten (Worte)
  - Alphabet: beliebige Menge von Zeichen, z.B. A = {a,b}
  - Worte entstehen durch Konkatenation ° von Zeichen, z.B. ab = a°b
  - A* Menge der endlichen Worte über einem Alphabet A
  - ε ist das leere Wort
  - $|w|$ Länge eines Wortes w
- Beispiel: Menge der Worte der Länge kleiner oder gleich 4 über dem Alphabet A = {k}
  - A* = {ε, k,kk,kkk,kkkk}

2.2. Einschub: Monoid

- A* ist ein Monoid, d.h.
  - Eine Menge M
  - mit einer assoziativen Verknüpfungsfunktion *o*, d.h. für alle *a,b,c* $\in$ *M gilt:*
    - *(a o b) o c = a o (b o c)*
  - und einem neutralem Element ε, sodass für alle *a* $\in$ *M* gilt:
    - *a o* ε = ε *o a = a*

- Andere Beispiele für Monoide:
  - $(\mathbb{N},+,0)$
  - $(\mathbb{N},x,1)$
  - (*P(M)*, $\cup$,$\varnothing$) für beliebige Menge *M*
  - (*P(M)*, $\cap$,*M*) für beliebige Menge *M*

2.2. Formale Sprache

- Sprache ist eine Teilmenge $L \subset A^*$
  - Beispiele: Deutsch, C++, Morsecode,….

- *Wie beschreibt man die Worte, die zu einer Sprache gehören?*
  - *Bei formaler Sprache systematisch möglich.*
  - *Deutsche Schreibsprache z.B. durch DUDEN festgelegt.*

2.2. Beispiel Sprache WAS

- Sei $X,Y \in A^*$, $A = \{a,s,w\}$
- Regeln zur Definition von $WAS \subset A^*$
  1. wa ist ein gültiges Wort der Sprache: $wa \in WAS$
  2. Wenn Xa ein gültiges Wort ist, dann auch Xas
     $Xa \in WAS \Rightarrow Xas \in WAS$
  3. Wenn wX ein gültiges Wort ist, dann auch wXX
     $wX \in WAS \Rightarrow wXX \in WAS$
  4. Wenn Xaaa$Y$ ein gültiges Wort ist, dann auch XsY
     $Xaaa Y \in WAS \Rightarrow XsY \in WAS$
  5. Wenn Xss$Y$ ein gültiges Wort ist, dann auch XY
     $Xss Y \in WAS \Rightarrow XY \in WAS$

- *Ist waa ein Wort der Sprache?*
- *Ist wsa ein Wort der Sprache?*

2.2. Beispiel Sprache WAS

- Regeln zur Definition von WAS $\subset$ A*
  1. wa ist ein gültiges Wort der Sprache: wa $\in$ WAS
  2. Wenn *X*a ein gültiges Wort ist, dann auch *X*as
     *X*a $\in$ WAS => *X*as $\in$ WAS
  3. Wenn w*X* ein gültiges Wort ist, dann auch w*XX*
     w*X* $\in$ WAS => w*XX* $\in$ WAS
  4. Wenn *X*aaa*Y* ein gültiges Wort ist, dann auch *X*s*Y*
     *X*aaa*Y* $\in$ WAS => *X*s*Y* $\in$ WAS
  5. Wenn *X*ss*Y* ein gültiges Wort ist, dann auch XY
     *X*ss*Y* $\in$ WAS => *XY* $\in$ WAS
- *Ist waa ein Wort der Sprache?*
- *Ist wsa ein Wort der Sprache?*

2.2. Erweiterte Backus Naur Form (EBNF)

- Allgemein: Welche Regeln erlaubt zur Sprachbeschreibung?

- Eine Grammatik ist eine formale Beschreibung einer Sprache
- Eine Grammatik kann durch die Notation EBNF beschrieben werden

- Terminale Zeichen (Alphabet): unterstrichen
- Nichtterminale Zeichen (Repräsentanten): <A>
- Leeres Wort: ε
- Produktionen: ::=
- Auswahl: |
- Wiederholung: {} (beliebig oft) {}+ (mind. einmal)
- Option: [] (keinmal oder einmal)

2.2. Beispiel Grammatik in EBNF

- Ziffern
  - <Ziffer> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
- Taschenrechnereingabe
  - <Ausdruck> ::= <Mult> (Start)
  - <Mult> ::= <Zahl> | (<Add>) | <Mult> * <Mult>
  - <Add> ::= <Mult> + <Mult>
  - <Zahl> ::= { <Ziffer> }+
- *Beispiel für einen gültigen Ausdruck*
- *Ist 1+2 möglich?*
- *Ist (1+ 3*4) möglich?*

2.2. Beispiel Grammatik in EBNF

- Ziffern
  - <Ziffer> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
- Taschenrechnereingabe
  - <Ausdruck> ::= <Mult> (Start)
  - <Mult> ::= <Zahl> | (<Add>) | <Mult> * <Mult>
  - <Add> ::= <Mult> + <Mult>
  - <Zahl> ::= { <Ziffer> }+
- *Ist (1+3*4) möglich?*

2.2. EBNF in EBNF

- <Grammatik> ::= { <Produktion> } (Start)
- <Produktion> ::= <NT> ::= <Ausdruck>
- <Ausdruck> ::= <NTTFolge>
- | <Ausdruck> | <Ausdruck>
- | [<Ausdruck>]
- | { <Ausdruck> } [±]
- <NTTFolge> ::= { <NT> | <T> } + | ε
- <NT> ::= ≤ { <Buchstaben> }+ ≥
- <T> ::= { <Zeichen> }+ unterstrichen
- <Buchstaben> ::=…
- <Zeichen> ::= …

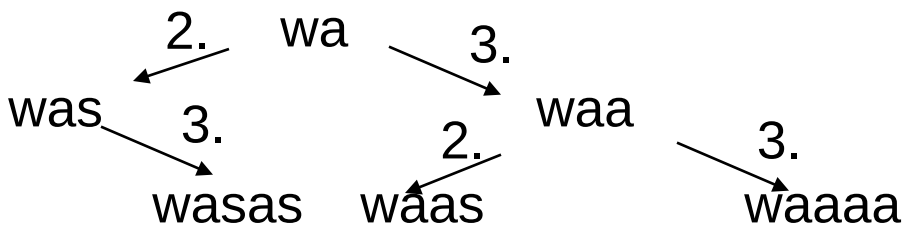
2.2. Parsen (Syntaxanalyse)

- Wie stelle ich fest, ob ein gegebenes Wort in der Sprache ist?
- Beispiel: (1) <A> ::= a<A>c; (2) <A> ::= abc
- Ist aabcc ein Wort dieser Sprache?
- Suche Ableitung
  - Start: <A>
  - Regel (1): a<A>c
  - Regel (2): aabcc

- *Woher wussten wir, dass als erstes Regel (1) angewendet werden sollte???*

2.2. Ableitungsbaum

- Ableitungsbaum: Zähle alle Worte der Sprache auf
  - Wurzel: z.B. wa (Start)
  - Kinder eines Knoten: alle ableitbaren Wörter



- Syntaxanalyse:
  - Suche passende Ableitung im Ableitungsbaum

2.2. Beispiel AbleitungsbaumSprache WAS

- Regeln zur Definition von WAS $\subset$ A*
  1. wa ist ein gültiges Wort der Sprache: wa $\in$ WAS
  2. Wenn *X*a ein gültiges Wort ist, dann auch *X*as
     *X*a $\in$ WAS => *X*as $\in$ WAS
  3. Wenn w*X* ein gültiges Wort ist, dann auch w*XX*
     w*X* $\in$ WAS => w*XX* $\in$ WAS
  4. Wenn *X*aaa*Y* ein gültiges Wort ist, dann auch *X*s*Y*
     *X*aaa*Y* $\in$ WAS => *X*s*Y* $\in$ WAS
  5. Wenn *X*ss*Y* ein gültiges Wort ist, dann auch XY
     *X*ss*Y* $\in$ WAS => *XY* $\in$ WAS

wa

- *Was passiert, wenn Wort nicht ableitbar ist?*
  - *Terminiert nicht => eingeschränkte Sprachen nötig*

2.2. Chomsky Hierarchie

- Reguläre Sprachen
  - Bsp.: <A> ::= a | <A>a, beliebige Folgen von a
  - (links: einzelnes NT, rechts: max. ein NT „außen“)
- Kontextfreie Sprachen
  - Bsp.: <A> ::= e | a<A>b, gleich viele a und b; d.h. $a^n b^n$
  - (links: einzelnes NT, rechts: beliebig)
- Kontextsensitive Sprachen
  - Bsp.: $a^n b^n c^n$
  - Produktionen der Form: x<A>y ::= xzy
  - (<A> ::= z ist kontextfrei; x,y beliebige Folgen von Terminalen)
- Beliebige Sprachen
  - (links: beliebig, rechts beliebig)

2.2. Zusammenfassung

- Eine formale Sprache wird durch eine Grammatik beschrieben

- Wichtig in der Informatik, insbesondere
  - Eine Programmiersprache ist eine formale Sprache
  - Eine Programm ist ein Wort (zur Beschreibung eines Algorithmus) in einer Programmiersprache!

1. Überblick Einführung

- Lernziel: Einführung in und Überblick über die Softwareentwicklung (Programmierung) im Kleinen

- 1.1. Was ist Informatik?
- 1.2. Was sind die Aufgaben von InformatikerInnen?
- **1.3. Was sind die Grundkonzepte der Informatik?**
- **1.4. Überblick über die Teilgebiete der Informatik**

Fortsetzung 1.3. Grundkonzepte

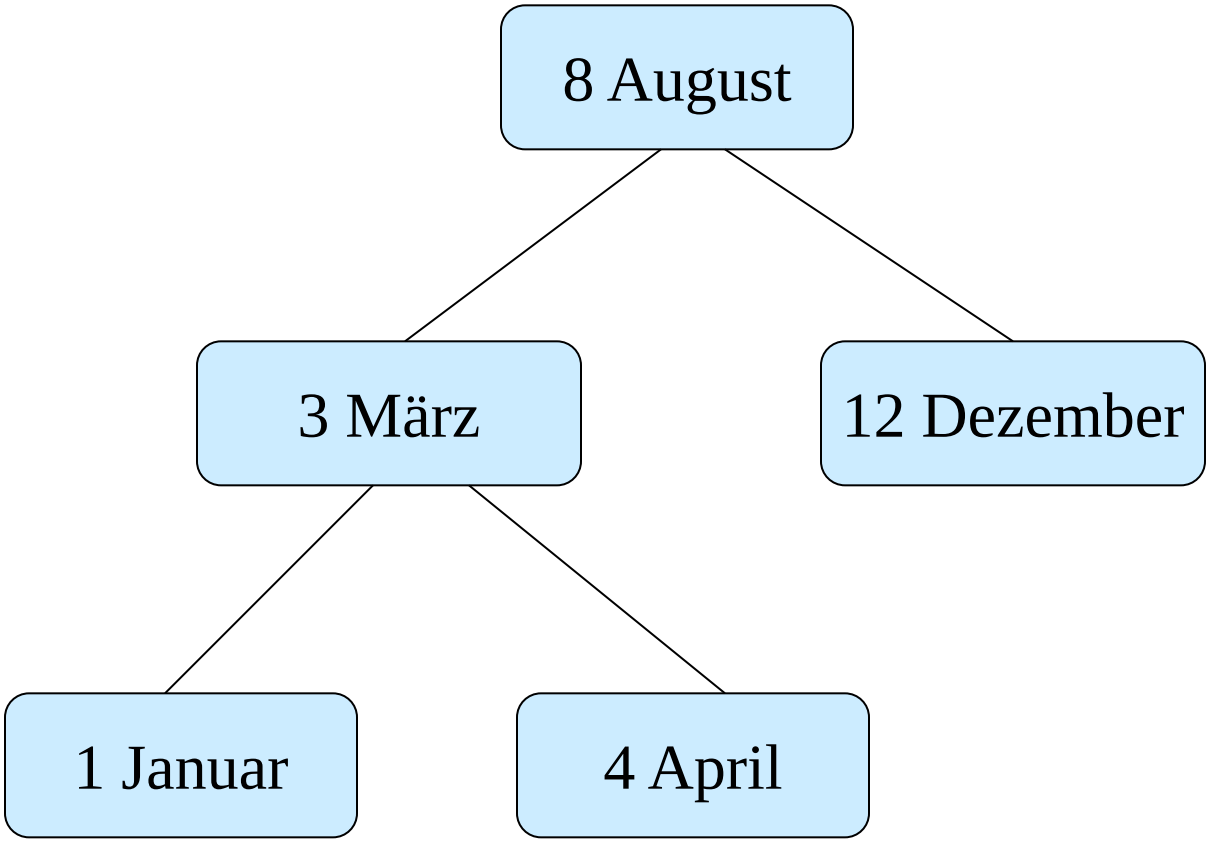
- System (siehe 1.1.)
- 1.3.1. Algorithmus, Programm
  - Terminierend
  - Deterministisch
  - Determiniert
  - Rekursiv
- 1.3.2. Datenstruktur
- 1.3.3. Prozess
- Ablaufbeschreibung
- 1.3.4. Maschine
- 1.3.5. Modell
  - Abstraktion
  - Transformation
  - Verifikation
  - Validation

1.3.2. Was ist eine Datenstruktur?

- Ein Algorithmus verarbeitet Ein- und Ausgabedaten und speichert diese und die Zwischenergebnisse in einem internen Datenzustand.
- Diese Daten sind notiert in der Programmiersprache (Teilsprache für Datenstrukturen).

- Typische für Menschen gut lesbare Datenstrukturen sind Aufzählungen, Tabellen, Diagramme.

1.3.2. Beispiel Datenstruktur (Baum)



1.3.2. Beispiel Datenstruktur (Record)

< Hanna, klug, 23, Informatik>

1.3.3. Was ist ein Ablauf (Prozess)?

- Ein Programm läuft auf einem Rechner ab.
- Ein Ablauf ist eine Folge von (Verarbeitungs-) schritten.

- Die operationale Semantik eines Programms (eines Algorithmus) ist die Menge seiner Abläufe

- (denotationale Semantik: Funktion, die Eingabe auf Ausgabe abbildet).

1.3.3. Ablaufdiagramme

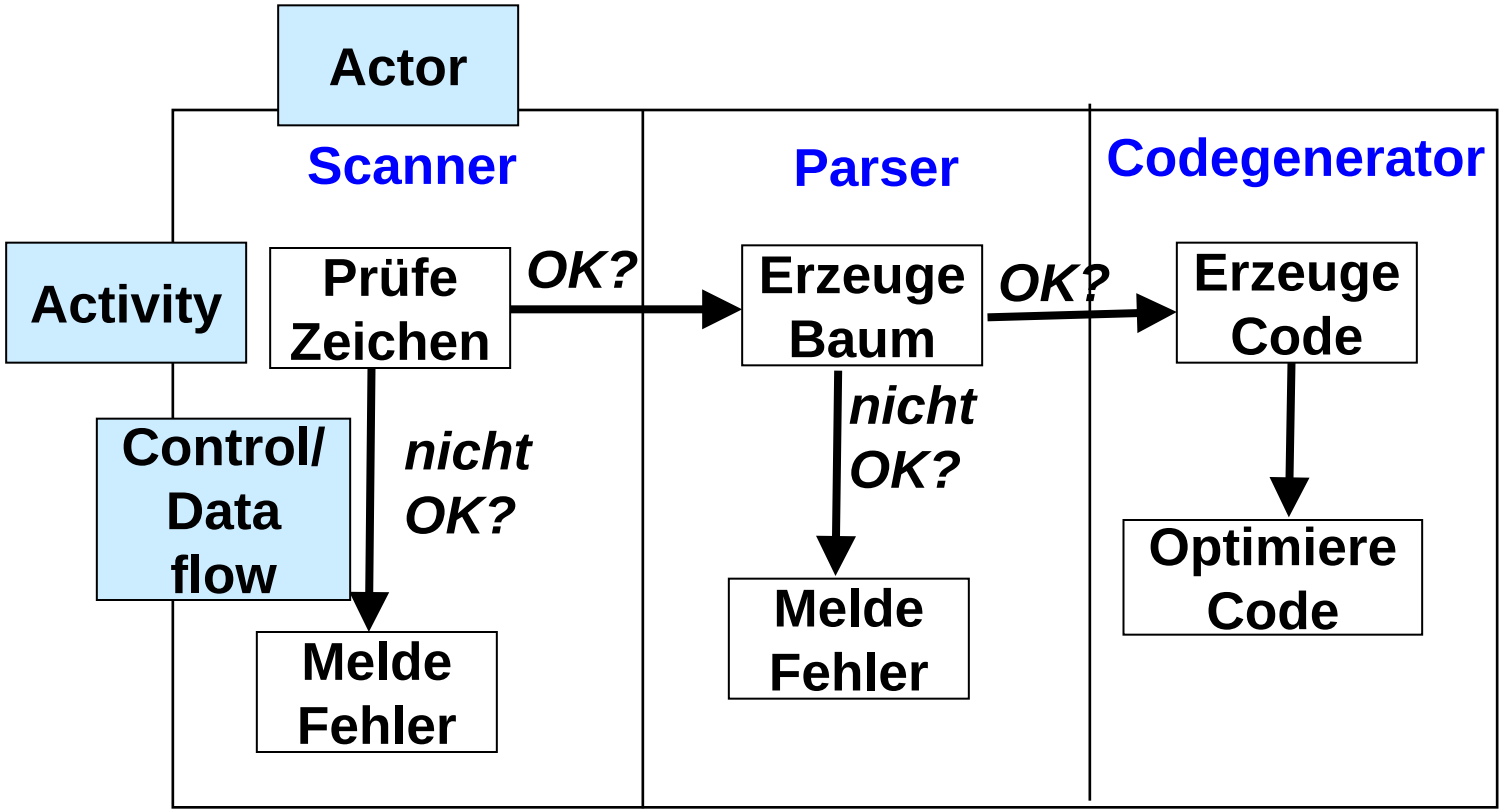
- Um ein Programm (einen Algorithmus) zu verstehen, muss man sich die Abläufe vorstellen (visualisieren).
  - Aktivitätsdiagramm: Abfolge der Verarbeitungsschritte
  - Zustandsdiagramm: Abfolge der internen Programmzustände
  - Sequenzdiagramm: Abfolge der Nachrichten

1.3.3. Beispiel Übersetzer

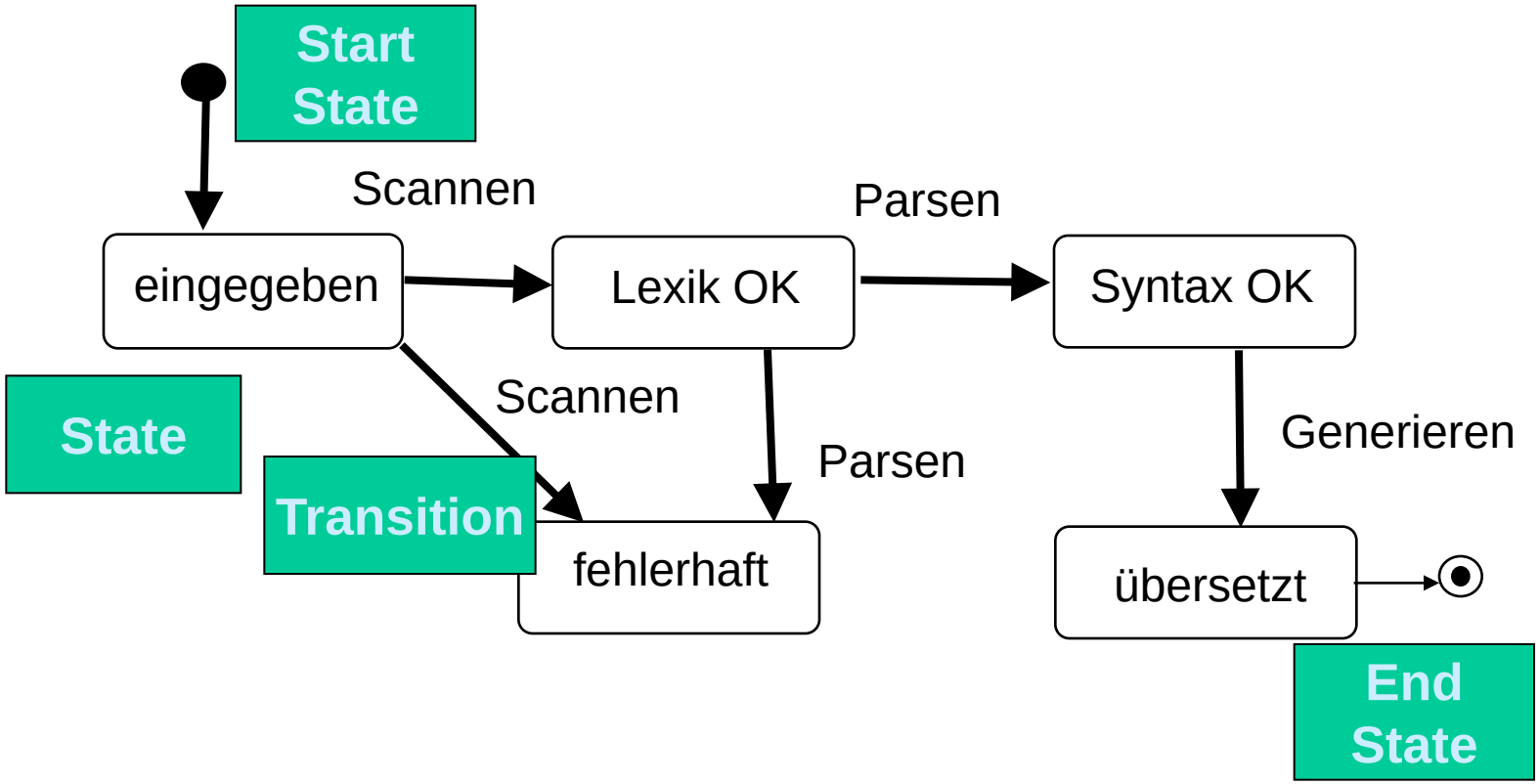
- Ein Übersetzer (Compiler) erzeugt aus dem Programmtext den ablauffähigen Binärcode.
  - 1.Schritt: Lexikalische Analyse
    - Überprüfe Zeichen
  - 2.Schritt: Syntaxanalyse
    - Überprüfe Worte (leite Syntaxbaum ab)
  - 3.Schritt: Codeerzeugung und- optimierung

- Ein Interpreter übersetzt immer nur einen Befehl und führt ihn gleich aus.

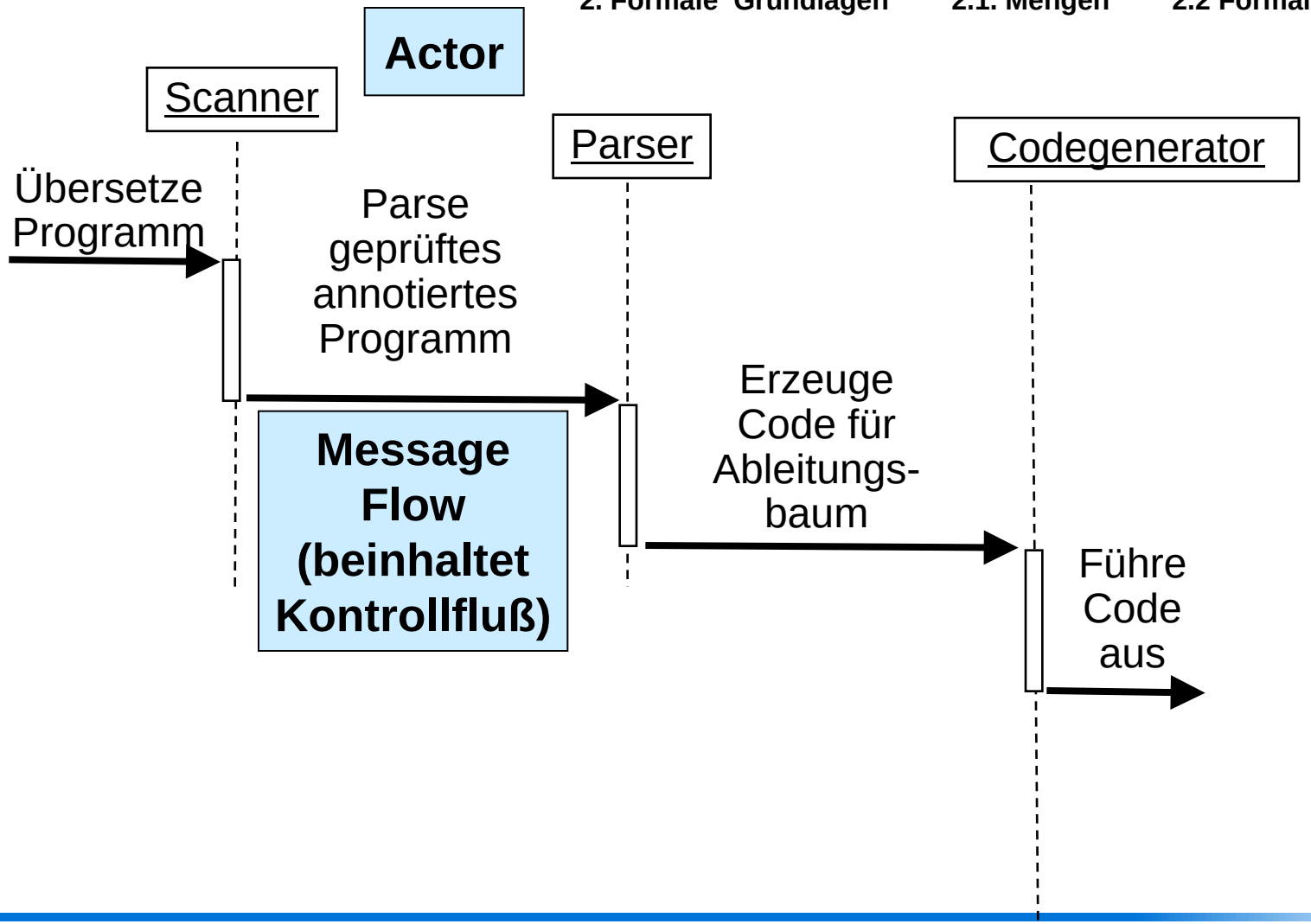
1.3.3. Aktivitätsdiagramm Übersetzen



1.3.3. Zustandsdiagramm Übersetzen



1.3.3. Sequenzdiagramm Übersetzen



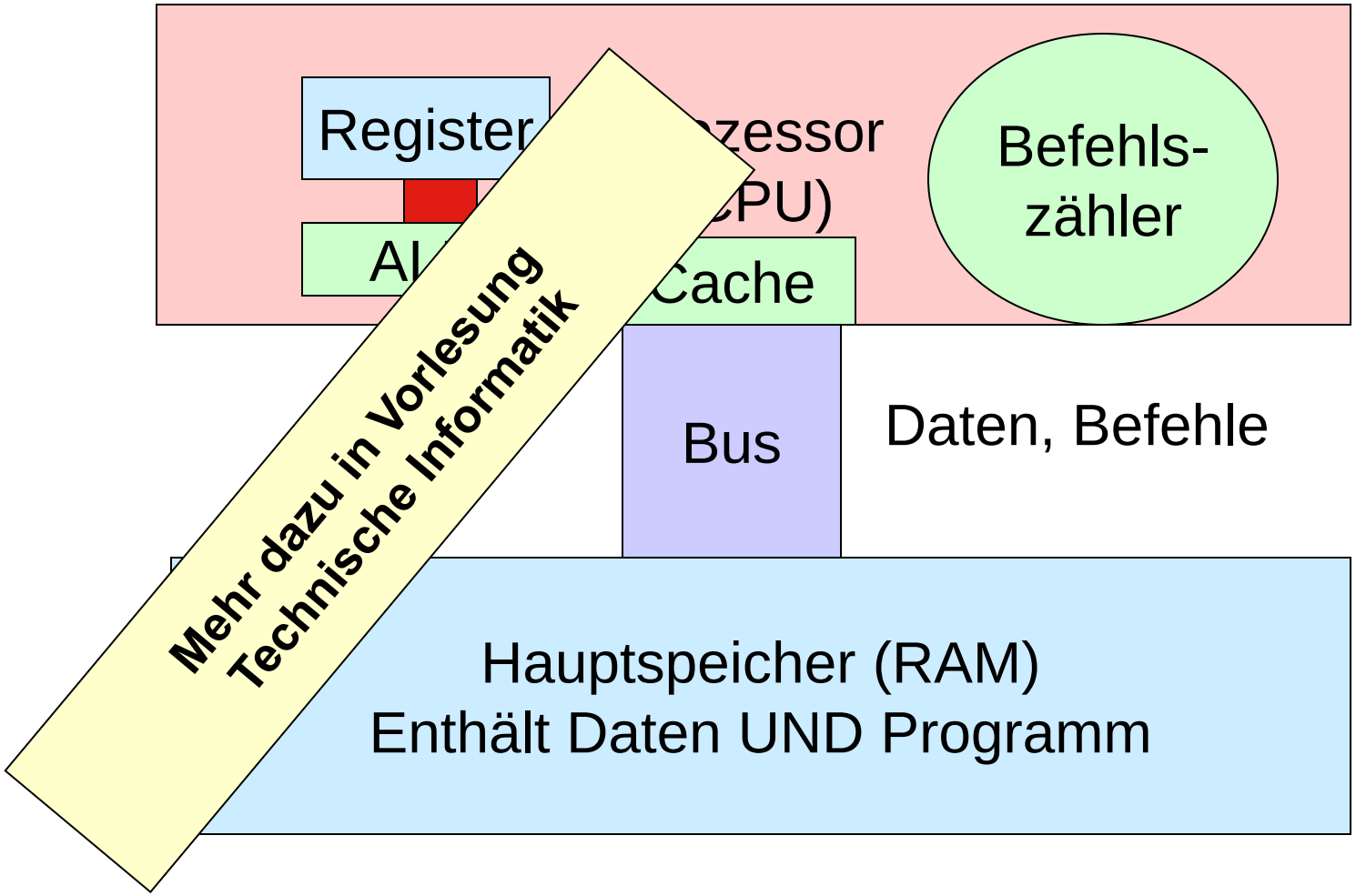
1.3.4. Was ist eine Maschine?

- Eine Maschine verarbeitet Eingabedaten zu Ausgabedaten (EVA: Eingabe – Verarbeitung – Ausgabe)
- Beispiele: Rechenschieber, Gemüsehobel, Computer, Menschen…

- *Was ist die Ein/Ausgabe des Menschen?*

- *Ein Computer ist eine Maschine? Wie funktioniert ein Computer?*

1.3.4. Computeraufbau: Von-Neumann Maschine



1.3.5. Was ist ein Modell?

- Ein Modell ist seinem Wesen nach eine in Maßstab, Detailliertheit und/oder Funktionalität verkürzte beziehungsweise abstrahierte Darstellung des originalen Systems [Stachowiak 73]
- Ein Modell ist eine Abstraktion eines Systems mit der Zielsetzung das Nachdenken über ein System zu vereinfachen, indem irrelevante Details ausgelassen werden [Brügge 00].

- Beispiele für Modelle: Schaltbild, Baupläne, Architekturmodell, Anatomie Gerippe, Modelleisenbahn, Papierflieger, Drehbuch, Höhlenzeichnungen, *AT-Diagramm, ER-Diagramm, Programm*

- Zur Darstellung von Modellen ist eine **Notation** notwendig.
  - In Informatik typisch: Text, Diagramm, Tabelle, Formeln, Programmtext, Strukturierter Text (z.B. eingeschränkte Satzform), Pseudocode

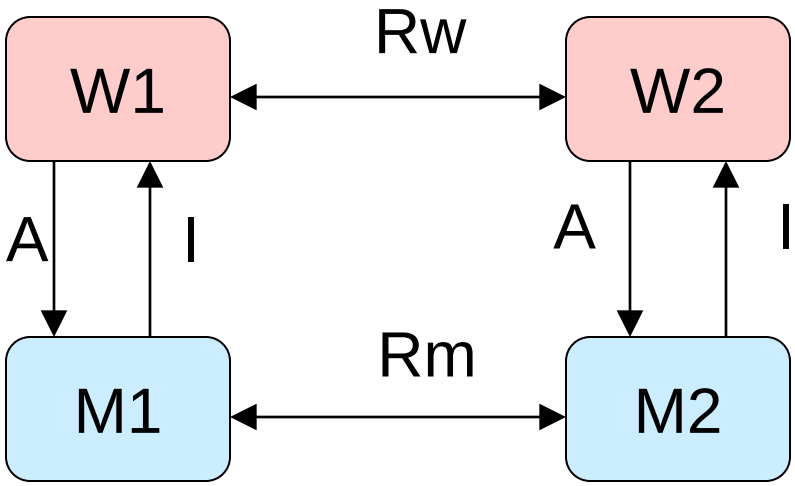
1.3.5. Bestandteile einer Notation

- Syntax (formale Sprache)
  - Welche Zeichenfolgen / Bildelemente sind für die Modelldarstellung erlaubt?
- Semantik
  - Was ist gemeint?
  - Interpretation des Modells in der realen Welt!
- Pragmatik (Methodik des Gebrauchs)
  - Wozu gebraucht man Modelle?
- Auch eine Programmiersprache ist eine Notation

1.3.5. Wozu braucht man Modelle?

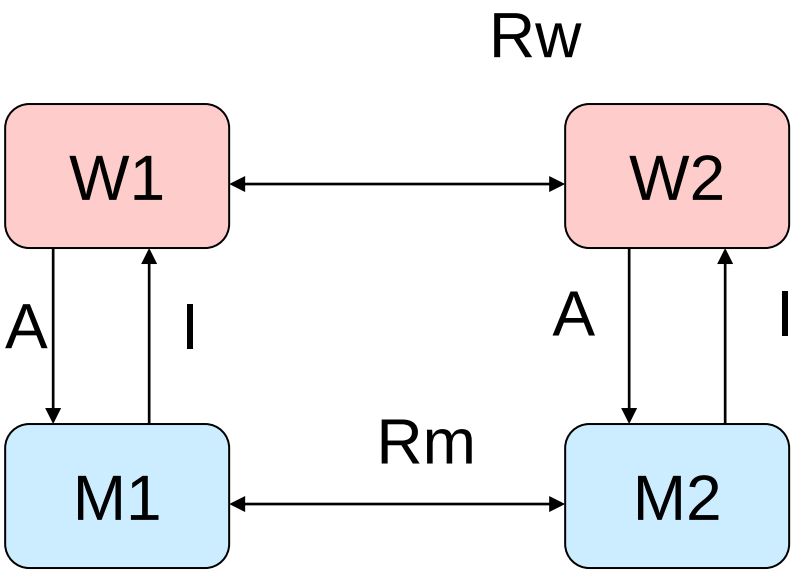
- Modelle helfen komplexe Zusammenhänge zu verstehen
  - Bildhafte Darstellung und Abstraktion unterstützen Kommunikation zwischen verschiedenen Beteiligten
    - Skizze zur Diskussion
    - Vorgabe zur Entwicklung
  - Exakte Notation und Abstraktion ermöglicht Verwendung von Werkzeugen
    - zur Analyse von Eigenschaften (des Originals)
      - Größe, Komplexität, Erreichbarkeit von Graphen, Zeitverbrauch, Vorhersage
    - Zur Transformation von Modellen
      - Bezug zwischen verschiedenen Modellen, Änderung von Modellen

1.3.5. Modell und Wirklichkeit



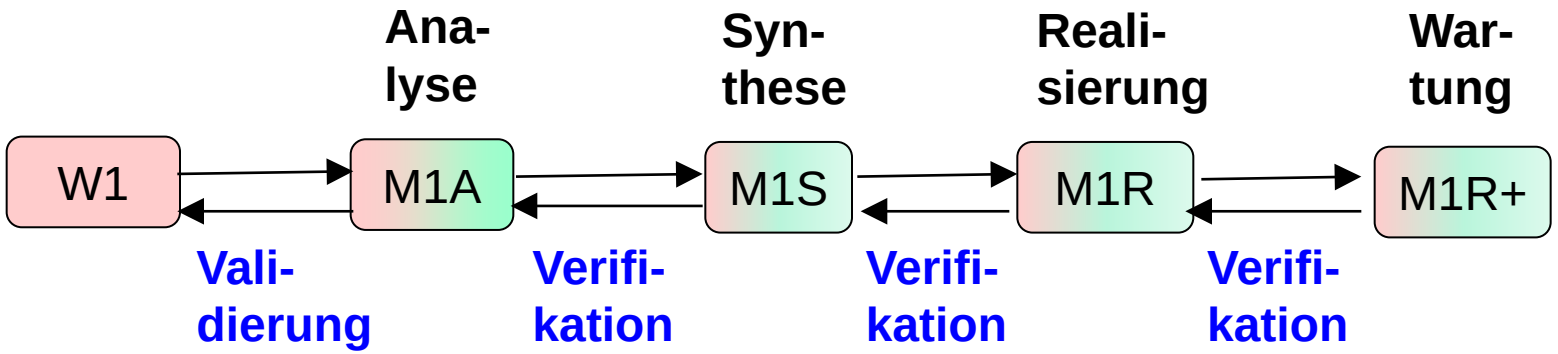
- W1, W2: Elemente der realen Welt
- Rw: Relation in der realen Welt
- M1, M2: Elemente im Modell
- Rm: Relation im Modell
- A: **Abstraktion** (Abbildung reale Welt -> Modell)
- I: **Interpretation** (Abbildung Modell -> reale Welt)

1.3.5. Was ist ein gutes Modell?



Ein Modell ist gut, wenn das Diagramm kommutativ ist, d.h.
Beziehungen im Modell gelten genau dann, wenn sie auch in der Wirklichkeit gelten

1.3.5. Modelltransformation



- Programm ist auch ein Modell !
- Softwareentwicklung kann als
- Modelltransformation angesehen werden
- Verifikation: ist die Transformation von M1x nach M1y korrekt?
- Validation: entspricht Modell M1x der Wirklichkeit W1?

1.4. Teilbereiche der Informatik

- Theoretische Informatik
  - Theoretische Analyse und Konzeption von Informatiksystemen (Grenzgebiet zur Mathematik/Logik)
  - Z.B. *Gibt es für jedes Problem einen Lösungsalgorithmus? Gibt es für jedes Problem einen beliebig schnellen Algorithmus?*

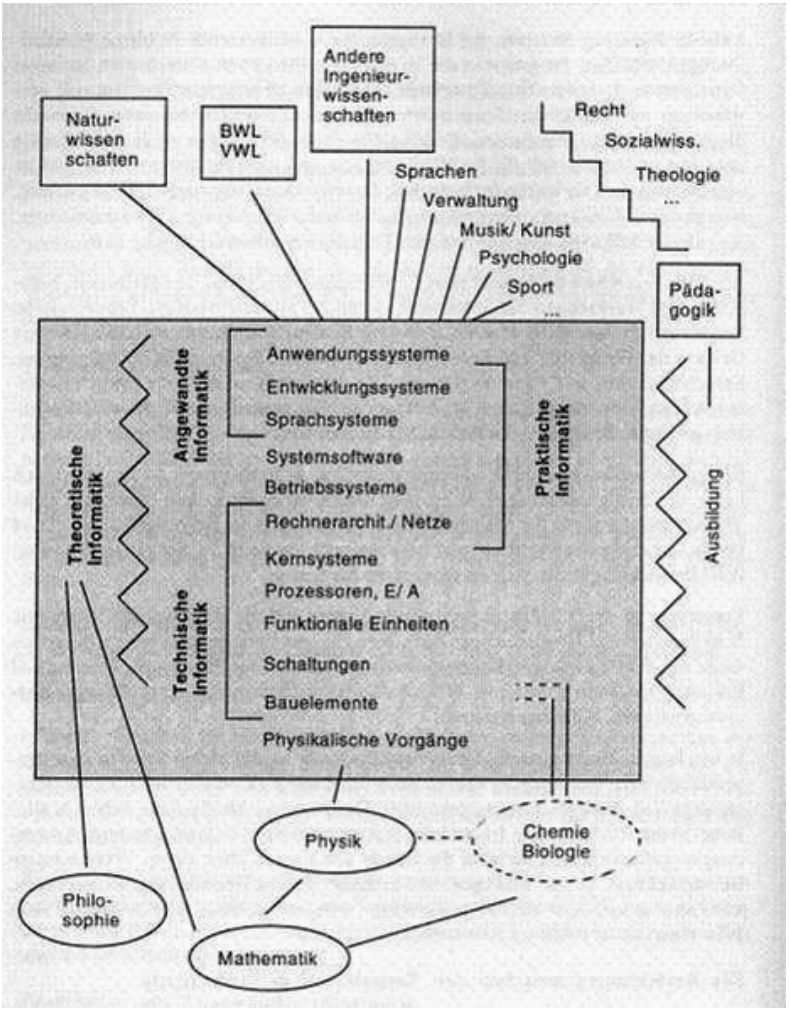
- Technische Informatik
  - Realisierung von Informatiksystemen (insbesondere technische Komponenten) (Grenzgebiet zur Elektrotechnik)
  - *Z.B. Chipentwurf, effiziente Darstellung und Verarbeitung von Bits*

1.4. Teilbereiche der Informatik ff

- Systembezogene Informatik
  - Organisatorische und technische Gestaltung von Informatiksystemen
  - Oft unterteilt in praktische Informatik (allgemeine Praktiken) und angewandte Informatik (anwendungsspezifische Praktiken)
  - Z.B. *effiziente Bereitstellung großer Datenmengen, verständliche Visualisierung von Daten, Vernetzung, Zusammenarbeit bei der Softwareerstellung*

- *Guter Überblick in [Rechenberg 2000]*

1.4. Schichtenarchitektur der Informatik



1.4. Trends in der Softwareentwicklung

- Prozessoren verdoppeln Geschwindigkeit alle 18 Monate (Moore‘s law)

- BMW 7er Serie
  - 96 Prozessoren (!!!), mehrere Kilometer Kabel
  - 10 Millionen Lines of Code (im Jahr 2003)
  - 100 Millionen Lines of Code in 6 Jahren
- ADAC: 50% der Pannen in 2002 waren E-Fehler
- Programmierleistung steigt nicht so schnell!
  - 10.000 neue Lines of Code pro Jahr
  - 2 Lines of Code ändern pro Tag

1. Zusammenfassung

- InformatikerInnen gestalten sozio-technische Systeme

- Informatik charakterisiert durch Beschäftigung mit Information, System, Algorithmus, Daten, Prozessen, Maschinen, Modellen

1. Literatur

- Schülerduden Informatik, Dudenverlag
- H.-J. Appelrath, J. Ludewig. Skriptum Informatik, Teubner Verlag, 1999
- M.Broy. Informatik – Eine grundlegende Einführung, Bd 1, Springer Verlag, 1997
- B. Brügge. Vorlesungsfolien Informatik I WS 03/04 http://wwwbruegge.informatik.tu-muenchen.de/twiki/bin/view/Lehrstuhl/Informatik1WiSe2003
- F. Naumann. Vom Abakus zum Internet – Die Geschichte der Informatik, Primus Verlag, 2001
- B. Paech. Aufgabenorientierte Softwareentwicklung, Integrierte Gestaltung von Unternehmen, Arbeit und Software, Springer Verlag 2000
- G. Pomberger, H. Dobler, Algorithmen und Datenstrukturen, Pearson Strudium, 2008
- P. Rechenberg, Was ist Informatik? Eine allgemeinverständliche Einführung, Hanser Verlag , 3.Aufl. , 2000