

Einführung in die Informatik II
Überblick

Prof. Bernd Brügge, Ph.D
Institut für Informatik
Technische Universität München

Sommersemester 2001

30. April 2001

Aufgabe des Informatikers

- ❖ **Unser Ziel:** Informatik-Systeme von hoher Qualität zu produzieren. Wir unterscheiden dabei 3 Aktivitäten
- ❖ Analyse: Wir erfassen die Natur des Problems durch Interviews mit Experten, die Erstellung einer Taxonomie und das Aufbrechen des Problems in kleinere Subprobleme.
- ❖ Synthese (Entwurf): Wir erstellen oder benutzen existierende Teillösungen für die Subprobleme, und integrieren die Teillösungen in eine Gesamtlösung
- ❖ Implementation: Wir bilden die Gesamtlösung in eine Beschreibung ab, die auf einem bereits existierenden Informatik-System ausgeführt werden kann.

- ❖ Diese Aktivitäten werden allgemein beim Lösen jedes Problems benötigt.

Entwicklung von Informatik-Systemen: Lösen von Problemen

- ❖ Die *Entwicklung von Informatik-Systemen* ist nichts anderes als ein Spezialfall der *Lösung von Problemen* *
- ❖ Um Probleme zu lösen, benutzen wir
 - Sprachen,
 - Notationen,
 - Definitionen,
 - Techniken,
 - Methoden(-sammlungen),
 - Werkzeuge und
 - Dogmen.

*Sehr empfehlenswert: George Polya's Buch übers Problemlösen:

"*How to solve it*", Princeton University Press, 1957, ISBN 0-691-08097-6

Siehe auch: <http://www.cis.usouthal.edu/misc/polya.html> und

<http://www-history.mcs.stand.ac.uk/history/Quotations/Polya.html>

Was benutzen wir bei der Lösung von Problemen (bei der Entwicklung von Informatik-Systemen)?

- ❖ *Sprache:*
 - Die Menge aller Worte über einem Alphabet.
- ❖ *Notation:*
 - Die äußere Form, in der Worte einer Sprache ausgedrückt werden.
- ❖ *Konzept:*
 - Wissensseinheit, die in Verbindung mit anderen Konzepten dazu beiträgt, Probleme und dazugehörige Lösungen zu verstehen.
Eine *Definition* ist die formale Beschreibung eines Konzeptes.
- ❖ *Technik (technique):*
 - Ein in einer Notation beschriebenes Verfahren, das von jemandem ausgeführt wird, um ein Resultat zu produzieren.
- ❖ *Werkzeug (tool):*
 - Ein Instrument oder ein automatisiertes System, das eine bestimmte Technik ausführen kann.
- ❖ *Ethik:*
 - Eine philosophische Anschauung, die sagt, was erlaubt ist und was nicht.
Jede Ethik enthält *Dogmen*.

Sprache bei der Entwicklung von Informatik-Systemen

- ❖ **Definition:** Die Menge aller Worte über einem Alphabet.
- ❖ Ich benutze die natürliche Sprache, um jemandem zu erklären, wie man Pfannkuchen herstellt.
- ❖ Wir benutzen formale Sprachen, um Informatik-Systeme zu beschreiben.

Beispiele aus Info I/II:

- UML (Modellierungssprache)
- Java (Programmierungssprache)
- ❖ Andere Modellierungssprachen:
SDL, OMT, OOSE,...
- ❖ Andere Programmiersprachen:
C, C++, Fortran, Cobol, VisualBasic, Smalltalk, Eiffel, ...

Notation

❖ **Definition:** Die äußere Form, in der Worte einer Sprache ausgedrückt werden.

Eine Notation ist also eine Repräsentation einer Sprache (zur Unterscheidung zwischen Nachricht, Repräsentation und Information, siehe Info I-Vorlesung 5, Folie 5)

❖ **Beispiele:**

- Liste der Zutaten für ein Pfannkuchenrezept
- To-do-Liste für ein Treffen
- Checkliste für eine Vorlesung

❖ **Beispiele aus Info I:**

- Textersetzungs-system (Notation für Programme)
- Termersetzungs-system
- Backus-Naur Form
- Syntax-Diagramm

Konzept

- ❖ **Definition:** Wissensseinheit, die in Verbindung mit anderen Konzepten dazu beiträgt, Probleme und dazugehörige Lösungen besser oder schneller zu verstehen, insbesondere für alle Probleme in einer bestimmten "Problemlösungsklasse".
- ❖ **Beispiele:**
 - Das Konzept der Bratpfanne
 - Das Konzept der *Klasse* (Unterstützt sowohl die Modellierung als auch die Programmierung)
 - Das Konzept der *Sortierung* (Bubblesort, Quicksort, Mergesort)
 - Das Konzept der *Sortiertheit* (Ordnung, Sortierkriterium)
 - Das *Kompositionsmuster* (hilft, rekursive Strukturen beliebiger Tiefe und Breite zu modellieren)
 - Das Konzept der *Entwurfsmuster* (Wiederverwendbares Wissen in der Entwurfsphase bei der Entwicklung von Software-Systemen).

Technik

- ❖ **Definition:** Ein in einer Notation beschriebenes Verfahren, das von jemand ausgeführt wird, um ein Resultat zu produzieren.
- ❖ **Beispiele:**
 - Pfannkuchenrezept
 - Spezifikation einer Schnittstelle
 - Bubble-Sort Algorithmus
 - Effizienzbetrachtungen für Algorithmen
 - Beschreibung, wie man Objekte bei der Modellierung findet

Werkzeug

- ❖ Ein Instrument oder ein automatisiertes System, das eine bestimmte Technik ausführen kann.
 - Ofen
 - Bleistift und Papier
 - Text-Editor
 - Java-Compiler
 - Java Virtuelle Maschine
 - Unix-Betriebssystem
- ❖ Für die Erstellung der Info II-Vorlesung benutze ich folgende Werkzeuge:
 - Erstellung der Vorlesungsfolien: Powerpoint 98
 - Zeichnen von UML-Diagrammen: ConceptDraw 1.59
 - Entwicklung von Java-Programmen: Codewarrior 6.0

Methode

- ❖ **Definition:** Eine Sammlung von Techniken, die zur Lösung von Problemen angewandt werden können.
 - Kochbuch
 - Alle Techniken, die zum objekt-orientierten Problemlösen benötigt werden.

Ethik

- ❖ **Definition:** Eine philosophische Anschauung, die sagt, was erlaubt ist und was nicht.
- ❖ Jede Ethik enthält *Dogmen*.
- ❖ **Beispiele für Dogmen:**
 - Fleischessen ist ungesund
 - Objekt-orientierte Modellierung ist besser als funktionale Dekomposition.
- ❖ **Beispiele für Dogmen aus Info I:**
 - Ein Attribut darf nie öffentlich sein.
 - Ein Problem muss erst modelliert werden, bevor man es programmiert.
 - Operationen müssen immer zu Klassen gehören. Operationen dürfen nicht "frei herumlaufen" wie bei der imperativen Programmierung.
(objekt-orientierte Programmierung ist ein Dogma!)

Wie benutzen wir Sprachen, Notationen, Konzepte, Techniken?

❖ Der Informatiker als Wissenschaftler:

- Entwirft neue Sprachen, die man bei Entwurf und Programmierung einsetzen kann.
- Vergleicht und untersucht Informatik-Systeme.
- Findet neue Techniken, beweist Theoreme über Techniken,
- Definiert Schemata für die Notation von Wissen.
- Definiert allgemeine Konzepte im Bereich der Informatik (ohne Zeitdruck).

❖ Der Informatiker als Ingenieur:

- Betrachtet Sprachen, Notationen und Techniken als existierende Basis für weitere Techniken und Werkzeuge beim Bau von Informatik-Systemen.
 - Setzt die Werkzeuge bei der Lösung eines Problem ein.
 - Definiert Konzepte im Bereich eines spezifischen Problems.
 - Realisiert Konzepte und Techniken innerhalb einer bestimmten Zeit und mit einem begrenzten Budget.
- ❖ In Info II behandeln wir sowohl die wissenschaftlichen als auch die ingenieur-spezifischen Aspekte der Entwicklung von Informatik-Systemen.

Was machen wir in Info II?

- ❖ Weitere Konzepte, Methoden, Notationen und Techniken:
 - für die **Erstellung** (Analyse, Entwurf und Implementierung) von Informatik-Systemen
 - für die **Beschreibung** (Dokumentation) von Informatik-Systemen
 - für die **Überprüfung** (Testen) von Informatik-Systemen
- ❖ **Schwerpunkte:**
 - Fortgeschrittene Datenstrukturen (Bäume, Keller, Warteschlange)
 - Objekt-orientierte Entwicklung:
 - Modellierung, Entwurf, Implementation
 - Grundprinzipien des Software-Engineering
 - Wichtige Entwurfsmuster (Strategie, Beobachter, Adapter)
 - Weitere Programmierstile (Ereignis-basiert, regel-basiert und maschinennah)
 - Umsetzung von höheren Sprachen in maschinenähere Sprachen

Spezifische Themen

- ❖ **Notationen für die Beschreibung von Informatik-Systemen**
 - UML, Java, Grammatiken, Automaten
- ❖ **Entwurf von Informatik-Systemen**
 - Wiederverwendbarkeit
 - Polymorphismus
 - Spezifikation, Verträge
- ❖ **Programmierstile**
 - Ereignis-basierte Programmierung
 - Regel-basierte Programmierung
 - Maschinennahe Programmierung
- ❖ **Zusammenfassung**
 - Übersicht und Bewertung der Programmierstile
 - Ein zusammenfassendes Beispiel

Aufteilung der Themen auf Vorlesungsblöcke

- ❖ **I. Beschreibung von Informatik-Systemen (Modellierung)**
 - *Block 1*: UML-Überblick (Ende April/Anfang Mai)
- ❖ **II. Wichtige Techniken beim Entwurf von Informatik-Systemen**
 - *Block 2*: Polymorphismus (Mitte Mai)
 - *Block 3*: Spezifikation, Entwurf durch Verträge (Ende Mai)
- ❖ **III. Programmierstile**
 - Ereignis-basierte Programmierung (Juni)
 - *Block 4*: Ereignisse und Ereignis-Verarbeitung (Anfang Juni)
 - *Block 5*: Ereignis-verarbeitende Systeme: Automaten, Grammatiken (Mitte Juni)
 - *Block 6*: Regel-basierte Programmierung (Ende Juni)
 - *Block 7*: Maschinennahe Programmierung (Anfang bis Mitte Juli)
- ❖ **IV. Zusammenfassung**
 - *Block 8*: Übersicht und Bewertung der Konzepte (ca. Mitte Juli)
 - *Block 9*: Ein zusammenfassendes Beispiel (Ende Juli)

Motivation für Block 1: Modellierung

- ❖ In Info I haben wir uns nur auf die Modellierung der statischen Aspekte eines Systems konzentriert.
- ❖ **Ziel 1:** Die Beschreibung der funktionellen Anforderungen und der dynamischen Aspekte bei der Ausführung eines Informatik-Systems muss auch möglich sein.
 - **Frage:** Wie beschreibe ich die Funktionalität des Systems?
→ Szenarien, Anwendungsfälle
 - **Frage:** Wie beschreibe ich die dynamischen Aspekte des Systems?
→ Ablaufbeschreibungen
- ❖ **Ziel 2:** Bei der Erstellung eines Informatik-Systems müssen die Wünsche verschiedener Interessengruppen ("stake holders"), z.B. Kunde, Benutzer und Entwickler des Systems berücksichtigt werden.
 - **Frage:** Wie beschreibe ich das System für unterschiedliche Benutzer, ohne dabei inkonsistent zu werden?

Block 1: Modellierung

- ❖ Überblick über UML
- ❖ Die wichtigsten UML-Diagramme (Notationen)
 - Wiederholung und Verfeinerung aus Info I: *Klassendiagramme*
 - Neu in Info II: *Anwendungsfalldiagramme, Sequenzdiagramme*
- ❖ Wann setzt man welche Diagramme ein?
- ❖ Taxonomie der Benutzer von Klassen
 - Benutzer, Spezifizierer, Implementierer, Erweiterer

Motivation für Block 2: Polymorphismus

- ❖ **Idee:** Informatik-Systeme kann man komponenten-orientiert erstellen (ähnlich dem Bau von Fertighäusern).
- ❖ **Ziel:** Austauschbarkeit einzelner Komponenten muss möglich sein, ohne andere Komponenten informieren oder ändern zu müssen.
- ❖ **Beispiel:**
 - Ersetzung eines Verschlüsselungsalgorithmus durch einen neuen (schnelleren/sicheren) Algorithmus (Info I Vorlesung 14)
- ❖ **Frage:** Wie kann ich den Implementierungsaufwand reduzieren, wenn ich eine existierende Komponente durch eine neue ersetze oder eine neue Komponente in das System einbringe?
 - Polymorphismus
- ❖ **Frage:** Wie kann man dem Benutzer garantieren, dass seine Erwartungen auch nach dem Komponententausch erfüllt sind?
 - Entwurf durch Verträge

Block 2: Polymorphismus (Mitte Mai)

❖ Polymorphismus bei Anwendungen

- Verkaufsautomaten als Beispiel

❖ Polymorphismus bei Klassenbibliotheken

- Behälter (Container) als Beispiel
- Verschiedene Container Implementierungen
 - Reihung, Baum, Hash-Tabelle
 - Stapel, Warteschlange
(→Block 6: maschinennahe Programmierung)

❖ Vererbung

- Die verschiedenen Arten der Vererbung ("Taxonomie der Vererbung")
- Vererbung vs Delegation

❖ Strategiemuster

- Vergleich: Strategiemuster vs Brückenmuster

Motivation zu Block 3: Entwurf durch Verträge

- ❖ **Ziel:** Überprüfbarkeit bestimmter Anforderungen und Eigenschaften von Informatik-Systemen.
- ❖ **Beispiele:**
 - "Das System muss 500 Anfragen pro Sekunde unterstützen"
 - "Das System unterstützt 500 Anfragen pro Sekunde"
 - "Die Suche ist äußerst effizient"
 - "Das System ist sehr einfach zu benutzen"
- ❖ Wie überprüfen wir Anforderungen und Eigenschaften?
In Info I hatten wir Tests gegen die Realität (Validierung) und Tests gegen andere Modelle (Verifikation) erwähnt.
- ❖ **Frage:** Wie legen wir Anforderungen und Eigenschaften fest, um sie überprüfen zu können?
- ❖ **Frage:** Wer ist für die Erstellung/Einhaltung/Überprüfung von Verträgen und Schnittstellen verantwortlich?

Block 3: Spezifikation (Ende Mai)

❖ **Kapselung, Komponentenbegriff**

- Motivation für Zugriffsrechte und Schnittstellen-Spezifikation

❖ **Zugriffsrechte: "Wer bearbeitet welche Teile einer Komponente?":**

- Spezifizierer einer Komponente (abstract)
- Implementierer der Komponente (private)
- Erweiterer der Komponente (protected)
- Benutzer der Komponente (public)

❖ **Schnittstellen-Spezifikation:**

- Entwurf durch Verträge [*Spezifikation von Klassen und Methoden*]
 - Vor- und Nachbedingungen, Invarianten
 - **OCL**: Werkzeug zur Beschreibung von Verträgen
 - **JavaDoc**: Werkzeug zur Dokumentation von Verträgen
- Zusicherungen [*Spezifikation von Anweisungen*]
 - Zusicherungskalkül (Hoare): Technik zur Verifikation von Programmen

Block 4: Ereignis-basierte Programmierung (Juni)

- ❖ Arten von Ereignissen
 - erwartete und unerwartete Ereignisse (Events, Exceptions)
- ❖ Ereignisse als Nachrichten (Codierung, Fehlererkennung)
- ❖ Beschreibung von Ereignissen, die zwischen Objekten ausgetauscht werden
- ❖ Behandlung von Ereignissen (Event Handling)
 - Beobachter-Muster (Observer Pattern)
 - Modell-Sicht-Kontroller (Model-View-Controller)
- ❖ Ein- und Ausgabe
- ❖ Java Applets
- ❖ Graphik-Programmierung
- ❖ Endliche Automaten (als ereignisverarbeitende Systeme)
- ❖ Beschreibung von endlichen Automaten
 - Noch eine UML-Notation: *Zustandsdiagramm*

Block 5: Automaten und Grammatiken (Mitte Juni)

- ❖ Die Chomsky-Hierarchie
 - Chomsky 3 Grammatiken als Beschreibung von endlichen Automaten (einseitig lineare Grammatiken)
 - Chomsky 2 Grammatiken als Beschreibung von Kellerautomaten (kontextfreie Grammatiken)
- ❖ Kellerautomaten
 - Das Keller-Prinzip zur Zwischenspeicherung von Daten
- ❖ Zusammenhang zwischen Automaten, Grammatik und Sprache
 - Pumping Lemma für CH3-Sprachen
 - Festlegung von Syntaxregeln für Programmiersprachen (wie z.B. Java) durch eine Chomsky 2 Grammatik
 - Analyse/Überprüfung der Syntax eines Java-Programms mit einem geeigneten Kellerautomaten

Motivation zu Block 6

- ❖ **Idee:** Die Zustandsübergänge in einem System folgen vorgegebenen Regeln. Ein System kann deshalb durch diese Regeln beschrieben werden.
- ❖ **Ziel:** Festlegung der Regeln durch ein Regelsystem
- ❖ **Beispiele:**
 - Markov-Algorithmen (siehe Info I)
 - Regeln für Spiele (Schach, Doppelkopf, Tic-Tac-Toe,...)
- ❖ **Fragen:**
 - Wie konstruiere ich ein regel-basiertes System?
 - Wie führe ich ein Programm regelbasiert aus, wenn immer mehrere Regeln anwendbar sind?

Block 6: Regel-basierte Programmierung (Ende Juni)

- ❖ Regel-basierte Systeme
- ❖ Horn Klausel
- ❖ Konflikt-Resolutionsstrategien
- ❖ Experten-Systeme als Beispiel von regel-basierten Systemen

Motivation zu Block 7: Maschinennahe Programmierung

- ❖ **Idee:** Die Realität ist durch Modelle beschreibbar.
- ❖ **Ziel:** Die Modelle eines Informatik-System können in Modellierungssprachen und Programmiersprachen formuliert werden.
- ❖ **Frage:** Wie übersetze ich diese Modelle in eine Sprache, die von einem Rechner verstanden wird?

Block 7: Maschinennahe Programmierung (Bis Mitte Juli)

- ❖ **Schichten** ("Abstraktionsebenen")
 - strenge Schichten-Architektur vs. offene Architektur
- ❖ **Das Konzept der abstrakten Maschine**
 - Interpreter [als Beispiel]
- ❖ **Umsetzung von höheren Sprachen auf maschinennahe Sprachen**
 - UML → Java
 - Java → PMI (Bytecode)
- ❖ **Unterprogrammaufrufe**
 - maschinennahe Umsetzung mit Hilfe des Kellers
 - Rekursion [als Spezialfall]
- ❖ **Speicherorganisation**
 - Halde (Objekte, Reihungen)
 - Keller (lokale Variablen)

Block 8: Zusammenfassung (Ende Juli)

- ❖ Übersicht, Vergleich und Bewertung der in Info I und Info II behandelten Konzepte
- ❖ Vergleich der Programmierstile
 - Funktionale Programmierung
 - Objekt-basierte Programmierung
 - Imperative Programmierung als Teil der objekt-basierten Programmierung
 - Objekt-orientierte Programmierung
 - Ereignis-basierte Programmierung
 - Regel-basierte Programmierung
 - Maschinennahe Programmierung
- ❖ Andere Entwurfsmethodologien und Programmiersprachen
- ❖ Vorausschau auf andere Vorlesungen

In Info II verwendete Literatur

- ❖ [Goos II] Gerhard Goos: *"Vorlesungen über Informatik"*, Band 2: Objekt-orientiertes Programmieren und Algorithmen, 2. Auflage, Springer Verlag 1999, ISBN 3-540-64340-0
- ❖ **Vertiefende Literatur:**
 - [Fowler] Martin Fowler, Kendall Scott: *"UML Konzentriert"*, 2. Auflage, Addison-Wesley 2000, ISBN 3-8273-1617-0.
 - [Gamma] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *"Entwurfsmuster"*, Addison-Wesley 1996, ISBN 3-89319-950-0.
 - [Küchlin] Wolfgang Küchlin, Andreas Weber, *"Einführung in die Informatik: Objekt-orientiert mit Java"*, Springer Verlag 2000, ISBN 3-540-67384-9.

In Info II verwendete englisch-sprachige Literatur

- ❖ [Fowler] Martin Fowler, Kendall Scott: *"UML Distilled"*, Addison-Wesley 2000, ISBN 3-8273-1617-0.
- ❖ [Bruegge] Bernd Bruegge, Allen Dutoit: *"Object-Oriented Software Engineering"*, Prentice Hall 2000, ISBN 0-13-489725-0
- ❖ [Gamma] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *"Design Patterns"*, Addison-Wesley 1996, ISBN 0-201-63361-2.
- ❖ [Gilbert] Stephen Gilbert, Bill McCarty: *"Object-Oriented Design in Java"*, Mitchell Waite Series 1998, ISBN 1-57169-134-0.
- ❖ [Gries] David Gries, Fred B. Schneider, *"A logical approach to discrete math"*: Springer Verlag 1996, ISBN 0-387-94115-0

In Info II verwendete englisch-sprachige Literatur

- ❖ [Warmer] Jos Warmer, Anneke Kleppe: *"The Object Constraint Language"*, Addison-Wesley, 2000, ISBN 0-20137940.
- ❖ [Morelli] Ralph Morelli: *"Java, Java, Java"*, Prentice Hall, 2000, ISBN 0-13-011332-8
- ❖ [Lafore] Robert Lafore: *"Data Structures and Algorithms in Java"*, Mitchell Waite Series 1998, ISBN 1-57169-095-6