# Werkzeuge, Modelle und Problemlösungen: Eine Konzeption für die Nebenfach-Informatik

Wolfgang Menzel, Jan W. Amtrup
Universität Hamburg, Fachbereich Informatik
Vogt-Kölln-Straße 30
D-22527 Hamburg

22. Mai 1998

**Zusammenfassung**

Ausgehend von der spezifischen Situation der Nebenfachausbildung wird ein Konzept für den Informatik-Grundkurs entwickelt, das

1. informatische Tätigkeit als bewußte Auswahl und Adaption geeigneter Problemlösungswerkzeuge versteht,

2. Softwarewerkzeuge und ihre formalen Grundlagen als integrale Einheit behandelt und

3. mit dem schrittweisenen Übergang von sehr einfachen zu zunehmend komplexeren Verarbeitungsmodellen einen konsequent induktiven Ansatz verfolgt.

# 1 Einführung

Ausgangspunkt für die hier vorgetragenen Überlegungen zu einer Neugestaltung der Nebenfachausbildung in Informatik war der seit mehreren Jahren zu beobachtende, kontinuierliche Rückgang der Nachfrage nach einem solchen Angebot an der Universität Hamburg. Waren vor fünf bis zehn Jahren noch Teilnehmerzahlen von bis zu 120 Studierenden typisch, ist die Teilnehmerzahl zur Zeit auf etwa 20 pro Jahrgang gefallen[1]. Ein derartig starkes Nachlassen der Nachfrage steht in einem sehr merkwürdigen Kontrast zu der zentralen Rolle, die die Informatik wegen ihrer Spezifik als Querschnittswissenschaft in der Nebenfachausbildung spielen könnte und zu der generell sehr hohen Wertschätzung, der sich computerbezogene Kenntnisse und Fertigkeiten auf dem Arbeitsmarkt erfreuen. Daher scheint der Verdacht nicht ganz unbegründet zu sein, daß die Informatikausbildung in ihrer traditionellen Form den spezifischen Bedürfnissen der Nebenfachstudierenden nicht mehr gerecht wird.

Die bislang angebotenen Grundkurse für Nebenfachstudierende boten im wesentlichen eine Einführung in die Programmierung in einer der typischen "didaktischen" Programmiersprachen. Die Wahl der Programmiersprache orientierte sich dabei sehr stark an den momentanen Kriterien für das Hauptstudium. Verwendet wurden z.B. Sprachen wie Modula-2 oder Oberon. Typisch für dieses Herangehen ist die starke Fixierung auf den Algorithmusbegriff, der in Themensetzungen wie "Vom Algorithmus zum Programm" [10] zum Ausdruck kommt. Ignoriert wird dabei, daß für weite Bereiche der *Nutzung* von Informatiksystemen (z.B. Datenbanken) der Algorithmenbegriff bestenfalls von untergeordneter Bedeutung ist[2]. Da jedoch aus der Perspektive eines Nicht-Spezialisten die Nutzung (und nicht die klassische Programmierung) die primäre Art der Wahrnehmung eines Informatiksystems ist, bestärkt sich der Verdacht, daß ein derartiger Ansatz tatsächlich nicht optimal auf die spezifische Situation der Nebenfachstudierenden ausgerichtet ist. Von ihnen wird erwartet, daß sie die Verwendung einer Programmiersprache erlernen, von der sie mit einiger Sicherheit annehmen können, daß sie in ihrem späteren beruflichen Umfeld keinerlei Bedeutung haben wird. Dieses Problem wird nicht in erster Linie durch die Auswahl einer konkreten Programmiersprache selbst hervorgerufen, sondern vor allem durch die einseitige Orientierung an der klassischen Programmierung schlechthin. Auch bei geeigneter Wahl der Sprache wird man kaum davon ausgehen können, daß die auf dieser Basis vermittelten Kenntnisse die Studierenden (als Nichtspezialisten) in die Lage versetzen, auch nur einigermaßen anspruchsvolle und dennoch ganz alltägliche Informationsverarbeitungsprobleme mit erträglichem Aufwand selbständig zu bearbeiten. Wie die Praxis der Studienberatung zeigt, ist aber genau dies ein häufig artikuliertes Motiv für die Wahl der Informatik als

[1] Dank verstärkter Werbung ist für den Jahrgang 1998 mit ca. 90 Anfängern die Nachfrage ungewöhnlich hoch. Für ein endgültiges Urteil darüber, ob sich hierdurch bereits eine Trendwende ankündigt, dürfte es allerdings noch zu früh sein.

[2] Dies gilt in ganz besonderer Weise für die hier betrachtete Zielgruppe, da an der Hamburger Universität Studierende der Mathematik und der Physik in gesonderten Veranstaltungen betreut werden. Somit werden genau diejenigen, für die der Algorithmenbegriff (etwa bei numerischen Aufgabenstellungen) in einem ganz natürlichen Sinne gegeben ist, von der hier diskutierten Lehrveranstaltungen nicht erfaßt.

Nebenfach.

Kann man bei Hauptfachstudierenden noch davon ausgehen, daß sie im Laufe der Ausbildung durch eine große Vielfalt von konkreten Anwendungsbeispielen schon noch an die Lösung komplexer Problemstellungen herangeführt werden, so ist eine solche Erwartung für Nebenfachstudierende schon aus Gründen des sehr begrenzten Zeitbudgets nicht mehr realistisch. Entweder der Ausbildung gelingt hier eine stärkere Orientierung an den praktischen Erfordernissen und Gegebenheiten bei der praktischen Anwendung, oder sie führt dazu, daß für die Studierenden eine separate (und damit unattraktive) zweite Erfahrungswelt errichtet wird, die kaum fruchtbare Bezüge zum Hauptfach aufweist.

Ein möglicher Ausweg aus diesem Dilemma besteht in dem Versuch, wichtige Aspekte der Realisierung von Informatiksystemen auf einer gegenüber berechnungsuniversellen Programmiersprachen deutlich höherem Abstraktionsniveau zu lehren. Nur so bietet sich u.E. die Chance, die Kluft zwischen Problemanalyse und Problemlösung erheblich zu verringern. Dieser Vorteil wird jedoch erkauft mit der Beschränkung auf solche Verarbeitungsmodelle, die nur noch einen eingeschränkten Leistungsumfang realisieren. Grundsätzlich ist der Übergang zu einem höheren Abstraktionsniveau nicht notwendigerweise mit einem Verlust an Anwendungsnähe verbunden, da eingeschränkte Verarbeitungsmodelle inzwischen die Grundlage einer Vielzahl von generischen Informatikwerkzeugen darstellen, so daß es die stärkere Hinwendung zu derartigen Spezialwerkzeugen sogar erlaubt, einen ganz unmittelbaren Zusammenhang zwischen theoretischer Fundierung und praktischer Realisierung in nichttrivialen Problemzusammenhängen herzustellen und zu lehren.

Ein schwerwiegender Mangel des traditionellen, an der Benutzung universeller Programmiersprachen orientierten Herangehens ist wohl die Tatsache, daß es eine über mehrere Jahrzehnte reichende Entwicklung der Informatik von der Rechnerprogrammierung hin zur Nutzung von spezialisierten *Problemlösungswerkzeugen* vollständig ignoriert. Bestimmte in der Frühzeit der Rechentechnik vorwiegend der Programmiersprachencompiler das Bild des Nutzers von den Möglichkeiten des Computers, so tritt in vielen Anwendungskontexten diese Sichtweise immer stärker in den Hintergrund, da spezialisierte Editoren, Datenbanksysteme, Statistikpakete, Kalkulations- und Simulationsprogramme, Modellierungswerkzeuge usw. eine unmittelbare Problemlösung unterstützen, ohne daß dabei auf die Verwendung einer universellen Programmiersprache zurückgegriffen werden muß. Eine zentrale Aufgabe ist damit nicht mehr die *Schaffung*, sondern die *Auswahl* eines geeigneten Werkzeugs und seine problemgerechte Adaption. Wer sich — weil er es nicht anders gelernt hat — blindlings in die Programmierung von Werkzeugen für seine alltäglichen Arbeitsaufgaben stürzt, wird im günstigsten Fall seine Arbeitskraft vergeuden. Wahrscheinlicher wird jedoch ein Mißerfolg sein, denn die Erfahrung lehrt, daß hierbei selbst gestandene Fachleute dazu neigen, die eigenen Fähigkeiten deutlich zu überschätzen. Eine an den konkreten Bedürfnissen der Studierenden orientierte Ausbildung sollte alles vermeiden, was eine derartige Entwicklung noch fördern könnte.

Voraussetzung für die sachgerechte Auswahl eines Werkzeugs, ist zweifellos ein hinreichend fundierter und systematischer Überblick über *Möglichkeiten und Grenzen* der jeweils verfügbaren Softwarewerkzeuge. Wenn sich die Ausbildung von vornherein an einer universellen Programmiersprache ausrichtet, dann stellt sich die Frage nach den inhärenten Grenzen eines Softwaresystems naturgemäß erst in zweiter Linie: Es ist ja gerade das Charakteristikum einer universellen Programmiersprache, daß sie die Programmierung all dessen erlaubt, was überhaupt programmierbar ist. Wie also sollte bei den Studierenden ein Verständnis dafür geweckt werden, warum in drei Jahrzehnten Informatikentwicklung hart daran gearbeitet wurde, den Freiraum bei der Programmierung einer VON NEUMANN-Maschine ganz gezielt zu beschränken (durch Abstraktion, Sichtbarkeitsbeschränkungen, Typisierung, Kapselung usw. aber auch durch Spezialisierung auf die ausgewählte Funktionalität von Spezialwerkzeugen)? Wie kann deutlich gemacht werden, welche Vorteile eine Spezialisierung hinsichtlich der Erlernbarkeit, der Effizienz, der Wartbarkeit und der Flexibilität bei verändertem Anwendungsprofil mit sich bringt? Wie kann bei den Studierenden ein Gespür dafür entwickelt werden, welche Repräsentationsmittel für welche Problemsichten besonders gut geeignet sind, wenn sich alles nur an den Ausdrucksmitteln einer einzigen Repräsentationssprache orientiert? Wie kann überhaupt gezeigt werden, daß es sehr unterschiedliche Darstellungsformen für ein und dasselbe Problem gibt, wenn sich die Ausbildung weitgehend darin erschöpft, den Studierenden die Besonderheiten einer willkürlich gewählten Repräsentation nahe zu bringen?

Als Alternative zu diesem an der klassischen Programmierung orientierten Ansatz wird im folgenden ein an generischen Werkzeugen ausgerichtetes Konzept vorgestellt, das im Sommersemester 1997 und im Wintersemster 1997/98 erstmalig in der Nebenfachausbildung am Fachbereich Informatik der Universität Hamburg erfolgreich eingesetzt wurde[3]. Es stellt konsequent den Gedanken der *Modellierung*[4] als Grundlage jeder Informationsverarbeitung in den Vordergrund und drängt die Fragen der algorithmischen Lösung eher in den Hintergrund. Das Schwergewicht liegt bei diesem Konzept auf einem vertieften Verständnis der jeweils zugrundeliegenden abstrakten Berechnungsmodelle. Gefördert werden soll die Fähigkeit, von den syntaktischen Details der jeweils verwendeten Repräsentation zu abstrahieren, verschiedene Repräsentationen miteinander zu vergleichen und sich explizit den Fragen der Modelladäquatheit zuzuwenden.

# 2 Rahmenbedingungen

Die Lehrveranstaltung "Informatik für Nebenfachstudierende wird als vierteiliger Zyklus angeboten, wobei nur die hier betrachteten ersten beiden Semester für alle Teilnehmer verbindlich sind. In Abhängigkeit vom Bedarf der einzelnen Hauptfächer kann sich eine Fortsetzung anschließen, die sich wahlweise stärker an den technischen oder theoretischen Grundlagen der Informatik orientiert. Von der Veranstaltung werden Studierende aus allen an der Universität vertretenen Hauptfächern erfaßt, mit Ausnahme der mathematisch bzw. physikalisch ausgerichteten Studiengänge.

[3]Kopien der in der Vorlesung verwendeten Folien, sowie die Übungsaufgaben finden sich unter `http://nats-www.informatik.uni-hamburg.de/~wolfgang/nf/` .

[4]Zur Bedeutung der Modellierungssicht für eine praxisrelevante Informatik-Ausbildung vgl. auch [9].

Für jedes der beiden Einführungssemester ist ein Stundenumfang von 2 SWS Vorlesung und 2 SWS Übung vereinbart. Die Übungen finden in Terminalräumen statt, so daß die Einführung in die Benutzung der verschiedenen Softwaresysteme, die Bearbeitung von Programmierungsaufgaben und die Demonstration von Musterlösungen direkt am Rechner erfolgen kann. Zur Verfügung standen Workstations unter Unix, sowie PCs unter Windows, wobei auf jeden Studenten ein Rechnerarbeitsplatz entfiel[5].

Der Studienablauf im zweiten Semester wurde für etwa drei Wochen durch einen Streik der Studierenden unterbrochen. Der dadurch entstandene Rückstand wurde mit einem Teil der Studierenden in einer ganztägigen Blockveranstaltung nach Semesterende nachgeholt.

# 3 Zielstellungen

Zentraler Entwurfsgedanke des hier vorgestellten Veranstaltungskonzepts war eine Einführung in die Grundlagen der Informatik für eine Zielgruppe von Lernenden, die den Computer vorrangig als *Hilfsmittel* für die Lösung ihrer berufsspezifischen Probleme betrachten muß. Daher sollten vor allem die ganz alltäglichen Aufgabenstellungen der Informationsverarbeitung im Mittelpunkt stehen, für die ein hoher und unmittelbar nachvollziehbarer praktischer Nutzeffekt gegeben ist. Aus diesem Grunde bot sich ein werkzeugzentriertes Herangegehen an, bei dem vor allem die Eignung der verschiedenen Spezialwerkzeuge für unterschiedliche Klassen von Problemstellungen diskutiert wird. Als Leitbild diente dabei die Vorstellung vom Einsatz des Computers als Hilfsmittel zum Lösen von Aufgaben, wie sie typischerweise bei der Anfertigung einer Diplom- oder Magisterarbeit anfallen.

Dem Charakter einer Einführung in die Grundlagen der Informatik entsprechend, kann und darf sich eine derartige Veranstaltung aber nicht nur auf Fragen der praktischen Handhabung von Softwaresystemen beschränken. Vielmehr dient die intensive Beschäftigung mit den Werkzeugen vor allem auch als Aufhänger um wesentliche informatische Grundbegriffe einzuführen und im praktischen Anwendungskontext zu exemplifizieren. Zentrales Element ist daher die Frage nach den zugrundeliegenden abstrakten Modellen und die Diskussion ihrer Notations-, Repräsentations- und Verarbeitungsaspekte.

Ausgehend von exemplarischen Verarbeitungsproblemen sollen die Studierenden in die Lage versetzt werden, solche Fragen zu stellen und zu beantworten, wie sie typisch für jede Problemanalyse sein sollten:

**Modelladäquatheit:** Welches Verarbeitungsmodell ist für die gegebene Problemstellung am ehesten geeignet? Welche Fragen müssen bei der Modellierung zwangsläufig ausgeklammert bleiben?

**Wissensakquisition:** Welchen Aufwand verursacht die Modellierung des gegebenen Verarbeitungsproblems unter Verwendung des gewählten Verarbeitungsmodells?

[5]Die Arbeitsbedingungen an den PCs waren weniger optimal. Hier mußten sich teilweise mehrere Studierende einen Arbeitsplatz teilen. Aufgrund von Lizenzproblemen konnte aber beim Teilgebiet Tabellenkalkulation nicht auf die Verwendung der PCs verzichtet werden.

**Systemrealisierung:** Welcher Verarbeitungsaufwand ist bei der Verwendung bestimmter Modellkonstrukte zu erwarten?

Der Weg der Problemlösung umfaßt dabei grundsätzlich die folgenden vier Teilschritte:

1. Analyse der Aufgabenstellung

2. Wahl eines formalen Modells für den Gegenstandsbereich

3. Wahl eines geeigneten (Spezial-)Werkzeugs

4. Systemrealisierung.

In dem Spektrum der verfügbaren Werkzeuge können dann berechnungsuniverselle Programmiersprachen nur noch als eine spezielle Klasse innerhalb der zahlreichen Ausprägungen von mehr oder weniger spezialisierten Implementierungswerkzeugen betrachtet werden.

Dieses an der Modellierung und Werkzeugauswahl orientierte Herangehen eröffnet eine Reihe ganz entscheidender methodischer Vorteile:

1. Es fordert und unterstützt eine sehr enge und ganz natürliche Verzahnung von formalen Grundlagen und ausgewählten Anwendungen der Informatik praktisch von der ersten Stunde an. Die Beschäftigung mit den mathematisch-formalen Grundlagen der Informatik erfährt für die Studierenden schon deshalb eine ganz deutliche Aufwertung, weil die theoretischen Grundlagen nicht als zusätzliche curriculare Anforderung, sondern als praktisch unmittelbar wirksames Hilfsmittel bei der Problemanalyse und bei der Auswahl geeigneter Werkzeuge in alltäglichen Aufgabenstellungen *erlebt* wird.

2. Es vermittelt einen guten Überblick über das Methodeninventar der Informatik und fördert eine Gesamtsicht auf das Fach.

3. Es stärkt eine problembezogene Sicht, indem es primär die Beziehung zwischen Aufgabenstellung und Verarbeitungsmodell thematisiert und Implementationsaspekte erst als nachgeordnete Fragestellungen behandelt. Dies ist vor allem deshalb möglich, da die konkrete Implementation ja zum großen Teil durch das jeweilige Softwarewerkzeug selbst bereitgestellt wird.

Voraussetzung für eine derartige Problemsicht ist natürlich die Verfügbarkeit detaillierter Kenntnisse über ein breites Spektrum typischer Softwarewerkzeuge und der durch sie realisierten Verarbeitungsmodelle.

Ein vorrangiges Ziel der Veranstaltung war es demnach, die Studierenden mit unterschiedlichen Verarbeitungsmodellen vertraut zu machen und ihnen Kriterien zur Auswahl geeigneter Werkzeuge in die Hand zu geben. Unterschieden wurde grundsätzlich zwischen typischen Anwendungen und den eher ungewöhnlichen Einsatzmöglichkeiten, wobei in jedem Einzelfall die folgenden Fragestellungen analysiert wurden:

- Für welche Aufgabenstellungen ist das Werkzeug bevorzugt geeignet?

- Welcher Formalismus liegt dem Werkzeug zugrunde?
- Welche Eigenschaften hat der Formalismus?
- Kann man mit dem Formalismus rechnen?
- Für welche Aufgabenstellungen ist das Werkzeug gerade noch geeignet?
- Für welche Aufgabenstellungen ist das Werkzeug nicht mehr geeignet?
- Welche Erweiterungen am Formalismus sind möglich/wünschenswert?

Wichtig für den Erfolg eines solchen Herangehens ist natürlich die Auswahl der zu behandelnden Themengebiete. Da vor allem Wert auf verallgemeinerbare und wiederverwendbare Informatikkenntnisse (und nicht auf implementationsspezifische Einzelheiten) Wert gelegt wurde, war in erster Linie zu prüfen, welches Grundlagenwissen anhand der allgemein verfügbaren Werkzeuge am besten vermittelbar ist. Wegen dieser Orientierung an informatisch anspruchsvollen Aufgabenstellungen schied etwa die Arbeit mit Editoren und Textverarbeitungssystemen als eigenständige Themenstellungen von vornherein aus. Letztendlich fiel die Wahl auf die vier Gebiete:

1. Reguläre Ausdrücke und endliche Automaten
2. Relationale Datenbanken
3. Tabellenkalkulation
4. Deduktive Datenbanken und Logische Programmierung

Diese Zusammenstellung realisiert ein induktives didaktisches Schema (vom Einfachen zum Komplizierten), wie es sich in zahlreichen Wissenschaften als zweckmäßig und erfolgreich erwiesen hat. Der Einstieg erfolgt über das einfachste Maschinenmodell, über das die Informatik überhaupt verfügt, den endlichen Automaten. Obwohl konzeptuell noch recht überschaubar und bereits mit zahlreichen Anwendungsperspektiven versehen, können auf dieser Ebene bereits einige fundamentale informatische Fragestellungen diskutiert werden:

- Repräsentation: verschiedene Repräsentationen für ein Modell bzw. verschiedene Modelle für eine Repräsentation
- Deklarativität: verschiedene Verarbeitungsmodelle für eine Repräsentation
- Nichtdeterminismus: Behandlung lokaler Nichtentscheidbarkeit durch Suche

Ausgehend von diesem relativ niedrigen Einstiegsniveau wird der Komplexitätsgrad schrittweise gesteigert, wobei die berechnungsuniverselle Programmiersprache erst am Ende dieses Weges steht. Neben dem didaktischen Prinzip des allmählichen Heranführens an kompliziertere Fragestellungen drückt diese Anordnung bereits auch eine wesentliche Botschaft der Veranstaltung an die Studierenden aus: Die Wahl einer berechnungsuniversellen Programmiersprache zur Lösung eines gegebenen Problems stellt *in jedem Fall* die allerletzte Möglichkeit dar, die immer nur dann gewählt werden sollte, wenn wirklich keine geeigneten Spezialwerkzeuge als Alternative zur Verfügung stehen!

Unseres Wissens ist ein derartiges induktives Herangehen in der Informatik bisher noch nicht praktiziert worden. Als vermutlich einzige Wissenschaft leistet sich die Informatik — sowohl im Schul- als auch im Hochschulbereich — derzeit noch den Luxus, mit einem sehr allgemeinen und dementsprechend komplizierten und fehleranfälligen Berechnungsmodell in die Anfängerkurse einzusteigen und erst im Anschluß daran, eingeschränkte und damit auch leichter zugängliche Spezialfälle zu diskutieren. So bietet etwa das Informatik-Lehrbuch für die Gymnasialausbildung [4] zuerst eine Einführung in die berechnungsuniverselle Programmierung mit Hilfe von Pascal, während einfachere Maschinenmodelle erst in den Schlußkapiteln des zweiten Bandes kurz Erwähnung finden. Diesem deduktiven Herangehen sind auch eher theoretisch angelegte Lehrbücher verpflichtet. So führt etwa [8] zuerst den sehr allgemeinen Begriff der abstrakten Maschine ein, ehe dieser dann im Kontext der Unterprogrammtechnik zum Kellerautomaten einschränkt wird. Der endliche Automat gar findet erst sehr viel später bei der Behandlung von Schaltwerken Verwendung. Ein analoges Vorgehen liegt auch praktisch allen anderen Informatik-Lehrbüchern (z.B. [3], [6]) zugrunde, soweit sie nicht die Behandlung eingeschränkter Berechnungsmodelle ganz ausklammern. Zur letzteren Kategorie gehören vor allem diejenigen Monografien, die die Einführung in die Informatik an die Verwendung einer ganz bestimmten Programmiersprache binden (z.B. [13], [11], [7], [5], [12], [1]). In diesen Fällen wird Informatik ohnehin in sehr problematischer Weise mit Programmierung (noch dazu in einem speziellen Programmierparadigma) identifiziert, wodurch sich zwischen theoretischer Fundierung und praktischer Implementierung eine vollkommen unnötige (und u.E. auch ganz unnatürliche) Trennung ergibt.

Aber auch in den Fällen, wo auf die Existenz eingeschränkter Berechnungsmodelle eingegangen wird, findet sich kaum ein Bezug auf konkrete Informatikwerkzeuge, in denen diese Modelle eine zentrale Rolle spielen. Eine Ausnahme hierzu bildet wohl das Lehrbuch von Aho und Ullman [2], das explizit auf relationale Datenbanken und reguläre Ausdrücke berücksichtigt. Andererseits folgen auch diese Autoren einem deduktiven Schema, da sie die Steuerstrukturen für berechnungsuniverselle Maschinen (Induktion, Iteration und Rekursion) an den Anfang der Betrachtung stellen (Kapitel 2), während das relationale Datenmodell (Kapitel 8) und die endlichen Automaten (Kapitel 10) erst viel später folgen.

Natürlich sind mit dem hier favorisierten werkzeugbasierten Herangehen nicht nur didaktische Vorteile verbunden. Problematisch ist vor allem die Tatsache, daß derzeit keine einheitliche Umgebung für die Arbeit mit sehr unterschiedlichen Formalismen zur Verfügung steht. Aus diesem Grunde müssen sich die Studierenden in sehr unterschiedliche Notationssysteme mit ihren syntaktischen Eigenheiten einarbeiten, ehe eine Beschäftigung mit den zugrundeliegenden Modellen und Verfahren erfolgen kann. Dies ist nicht ganz ein-fach und erfordert eine intensive Mitarbeit. Auf der anderen Seite fördert diese Situation aber auch den Trend zum Abstrahieren von syntaktischen Oberflächendetails und zwingt zur Arbeit mit Nachschlagewerken.

Soweit es sich irgend ermöglichen ließ, wurden die in der Veranstaltung behandelten informatischen Konzepte an praxisrelevanten Beispielen illustriert und vertieft. Hierzu zählen vor allem:

- Datenhaltung und Recherche in strukturierten und unstrukturierten Daten (z.B. relationale Datenbanken, Textkorpusrecherche)
- Kalkulation und Simulation für finanzielle Aufgaben (z.B. Haushaltskassenverwaltung, Workshopplanung etc.)
- elementare Grundlagen der Computergrafik, incl. Animation (z.B. Präsentationsgrafiken)
- deduktive Informationssysteme (z.B. Fahrplanauskunft)

# 4 Themenkomplexe

## 4.1 Grundlagen

Mit den in der Einführung behandelten grundlegenden Fragestellungen wurden zwei Hauptziele verfolgt:

1. die Vermittlung einer Grundsicht auf das Problem der *Modellierung*, auf die im weiteren Verlauf der Veranstaltung immer wieder zurückgegriffen wird und
2. die Ableitung der Veranstaltungskonzeption aus der Entwicklung der Informatik in den letzten drei Jahrzehnten.

Ausgangspunkt der Überlegungen zur Modellierung ist die Frage des Verhältnisses zwischen Information (einer personen- und interessengebundenen Kategorie) und ihrer Repräsentation, wobei die wechselseitige Abbildung durch Kodierungs- und Interpretationsprozesse erfolgt, die somit subjektiv gefärbt sein können. Modellierung ist dann die Darstellung eines informatisch zu behandelnden Weltausschnitts mit Hilfe eines Repräsentationsformalismus. Wichtige Teilschritte sind dabei die Wahl eines geeigneten Formalismus (ein zentrales Thema der Veranstaltung) und die Kodierung von Abstraktionen (Konzepten und Beziehungen zwischen Konzepten) in diesem Formalismus. Wesentlich ist dabei die Tatsache, daß es typischerweise keine eindeutigen Beziehungen zwischen Information und ihrer Repräsentation geben wird, eine Erkenntnis, die mit zahlreichen Beobachtungen an den einzelnen Formalismen untermauert werden kann.

Die Motivation des werkzeugbasierten Ansatzes ergibt sich vorrangig aus den zwei Antworten, die die Informatik auf die Herausforderung der Softwarekrise in den siebziger Jahren gefunden hat:

1. Die Softwaretechnik, die durch Programmiermethodik und Programmierdisziplin, sowie durch durch die Bereitstellung entsprechender Programmiersprachenkonstrukte die Produktion von Qualitätssoftware fördert. Diese Entwicklungsrichtung erbringt ihren Beitrag vorrangig im Bereich der "Programmierung im Großen".

2. Die Systeminformatik, die durch die Entwicklung von spezialisierten, aber generischen Werkzeugen für spezielle Aufgabenklassen die Wiederverwendbarkeit von Softwarelösungen ermöglicht. Durch die Bereitstellung konzeptuell vereinfachter Interaktionsschnittstellen hilft sie die kognitiven Anforderungen bei der Realisierung von komplexen Systemen deutlich zu verringern.

Aufgrund der spezifischen Situation der Nebenfachausbildung ergibt sich hiermit eine klare Präferenz für die zweite Entwicklungslinie.

## 4.2 Reguläre Ausdrücke und endliche Automaten

Ein natürlicher Einstieg in die Behandlung regulärer Ausdrücke bieten die Platzhalter (Wildcards), die als Grundlage für die unterbestimmte Spezifikation von Dateinamen in allen Betriebssystemen relativ gut bekannt sind. Bereits hier lassen sich unterschiedliche Interpretationsvarianten diskutieren und elementare Grundlagen für die Arbeit mit einem Betriebssystem legen.

Behandelt wird der Themenkomplex reguläre Ausdrücke/endliche Automaten aus vier verschiedenen Repräsentationssichten:

1. reguläre Ausdrücke

2. Zustandsübergangsgraph

3. Zustandsübergangstafel

4. Reguläre Grammatiken

Hingewiesen wird auf die gemeinsame deklarative Semantik. Die wechselseitigen Transformationsmöglichkeiten werden intensiv geübt. Untersucht werden die beiden prozeduralen Ausprägungen: Analyse und Generierung von Zeichenfolgen, insbesondere im Hinblick auf das Verhalten bei nichtdeterministischen Spezifikationen.

Typische Anwendungen der Arbeit mit regulären Ausdrücken werden in verschiedenen Bereichen der Modellierung sequentieller Strukturen vorgestellt:

- Korpusrecherche mit Suchmustern (Wortklassendetektion, KFZ-Kennzeichen, annotierte Korpora, usw.)

- elementare technische Automaten (Binärarithmetik, Münzautomat usw.)

- Modellierung der Morphotaktik natürlicher Sprachen

Als eher untypische Anwendung wird die Suche in strukturierten Datenbeständen (als ASCII-File) diskutiert, womit gleichzeitig eine Motivationsbrücke zur Welt der relationalen Datenbanken vorbereitet werden soll. Die Übungen erfolgen weitgehend unter Verwendung der Werkzeuge des Unix-Betriebssystems (`grep, egrep, sed,` ...)

Im Zusammenhang mit der Darstellung einfacher technischer Automaten wird insbesondere die Frage diskutiert, inwieweit unter Verwendung von endlichen Automaten gerechnet werden kann. Ausgehend von der Beobachtung, daß Rechnen einen funktionalen Zusammenhang zwischen Operanden und Resultat herstellt, werden endliche Automaten mit Ausgabe als geeignete Basis herausgearbeitet, die zudem über die Eigenschaft der Deklarativität verfügen.

Die Untersuchung der Grenzen der endlichen Automaten wird mit einem Vergleich zwischen den Präzedenzrelationen in arithmetischen Ausdrücken und der durch endliche Automaten realisierbaren Taschenrechnerarithmetik eingeleitet und anschließend auf den allgemeinen Fall der Klammersprachen generalisiert. Als mögliche und sinnvolle Erweiterungen werden vorgestellt:

- kontextfreie Modelle, mit einem Exkurs zur EBNF als substitutionsbasiertem Formalismus, der reguläre Ausdrücke als rechte Regelseite akzeptiert
- Markov-Modelle als stochastische endliche Automaten (mit Anwendungen in der Lastsimulation, der Sprachsignalerkennung, der Korpuslinguistik und der Genomanalyse)
- Petri-Netze als Automaten mit Nebenläufigkeit (mit Anwendungen in der Hardwaresimulation, der Modellierung menschlicher Handlungsmuster und zwischenmenschlicher Interaktion, der Simulation multimodaler Benutzungsschnittstellen, der Robotersteuerung und der Stoffstrommodellierung in der Umweltbilanzierung)

## 4.3 Relationale Datenbanken

Ausgehend von einer intensiven Auseinandersetzung mit dem Begriff der (extensional spezifizierten) Relation werden die Klauseln zur Datenerfassung (`insert`) und zur Recherche (`select`) eingeführt. Besondere Beachtung finden die Restriktionsbedingungen (einschließlich eines speziellen Exkurses zur Verwendung regulärer Ausdrücke) und die Projektionsmöglichkeiten. Behandelt werden außerdem die Verwendung von aggregierenden Auswertungsfunktionen (`count, sum, max, min,` ...) und Mehrtabellenanfragen (join). Großer Wert wird auf die Frage der Datenmodellierung über Entity-Relationship-Diagramme gelegt, da hier der Problembezug für die Studierenden und die unbedingte Notwendigkeit zur Abstraktion deutlich hervortreten.

Als typische Anwendungen werden Buchungssysteme, sowie die Kunden-, Personal- und Bestandsdatenverwaltung behandelt. Erste Probleme ergeben sich für komplex oder variabel strukturierte Daten (Entwicklung zu den objektorientierten Datenbanken) und der Behandlung generalisierbarer Information (Entwicklung zu den deduktiven Datenbanken).

Grenzen der relationalen Technik bestehen bei der Behandlung transitiver Relationen. Hier wird wiederum eine Motivationsbrücke zur Logikprogrammierung vorbereitet.

Hinsichtlich der Frage nach der Möglichkeit des Rechnens mit relationalen DB-Systemen wird

1. auf die Möglichkeit der Aufzählung der Ergebnisrelation für endliche Domänen und

2. auf die Möglichkeit der Auswertung funktionaler Ausdrücke in der `update`-Klausel

verwiesen, die beide jedoch keine praktikablen Lösungsansätze darstellen. Es wird die Notwendigkeit eines Mechanismus zur funktionalen Auswertung abgeleitet und damit der Übergang zur Tabellenkalkulation vorbereitet. Im Zusammenhang mit der `update`-Klausel wird auch auf die Grenzen der Deklarativität des relationalen Modells hingewiesen.

## 4.4 Tabellenkalkulation

Die Tabellenkalkulation wird als als Beispiel eines Berechnungsmodells eingeführt, das weitgehend auf der Auswertung funktionaler Ausdrücke beruht, wobei über die Struktur der Berechnungstabelle auch die Behandlung komplexer funktionaler Abhängigkeiten unterstützt wird. Da hiermit erstmalig der Schritt von der reinen Datenverwaltung zur aktiven Berechnung getan wird, stellt die Tabellenkalkulation einen entscheidenden Baustein in der logischen Abfolge der Veranstaltung dar. Von zentraler Bedeutung sind in diesem Zusammenhang die Begriffe der Variablen und der Variablenbindung, sowie der Funktionsbegriff, der in direktem Kontrast zum Relationsbegriff ausführlich diskutiert wird.

Anhand einfacher Beispiele zur Kostenkalkulation (Workshopbilanzierung, Planung eines F&E-Projekts) wird die Möglichkeit zur Realisierung von Simulationsexperimenten erläutert. Kognitionspsychologische Experimente bilden den Anwendungshintergrund zur Diskussion der grafischen Darstellungsmöglichkeiten.

Als neuartiges Element der Berechnung bietet die Tabellenkalkulation die Möglichkeit der wiederholten Abarbeitung eines funktionalen Ausdrucks. Hierbei wird die Wiederholung der Auswertung durch Kopieren der betreffenden Berechnungsvorschrift erreicht. Die sich dadurch ergebenden räumlichen Abhängigkeitsbeziehungen in der Kalkulationstabelle sind die Voraussetzung für eine sehr anschauliche Darstellung der verschiedenen Mechanismen zur Realisierung zyklischer Berechnungen:

- Iteration: beim Kopieren entsteht keine Abhängigkeit zwischen Original und Duplikat.

- Induktion: beim Kopieren entsteht eine funktionale Abhängigkeit zwischen Original und Duplikat.

- Rekursion: beim Kopieren entsteht eine funktionale Abhängigkeit zwischen Duplikat und Original

|  | A | B | ... |
|---|---|---|---|
| 1 | 1 | =a1*a1 |  |
| 2 | 2 | =a2*a2 |  |
| 3 | 3 | =a3*a3 |  |
| 4 | 4 | =a4*a4 |  |
| ... | ... | ... |  |

**a) Iteration**

|  | A | ... |
|---|---|---|
| 1 | 100 |  |
| 2 | =a1*0.05 |  |
| 3 | =a2*0.05 |  |
| 4 | =a3*0.05 |  |
| ... | ... |  |

**b) Induktion**

|  | A | ... |
|---|---|---|
| 1 | =a2/0.05 |  |
| 2 | =a3/0.05 |  |
| 3 | =a4/0.05 |  |
| 4 | =a5/0.05 |  |
| ... | 100 |  |

**c) (schlichte) Rekursion**

Abbildung 1: Funktionale Abhängigkeiten in der Kalkulationstabelle bei zyklischen Berechnungen durch Kopieren. $u \longleftarrow v$ verdeutlicht eine funktionale Bezugnahme $v = f(u)$, $u \dashrightarrow v$ steht für eine Kopieroperation $v = \text{kopie}(u)$.

Diese Abhängigkeitsbeziehungen werden bei modernen Tabellenkalkulationssystemen auf Wunsch auch grafisch hervorgehoben. Auf der Grundlage der geometrischen Interpretation bietet sich eine Systematisierung der verschiedenen zyklischen Berechnungsvorschriften einschließlich der prinzipiellen Möglichkeiten zu ihrer wechselseitigen Umwandlung an. Anwendungsbeispiele ergeben sich z.B. bei der Bilanzprognose unter der Annahme unterschiedlicher Wachstumsmodelle (linear, geometrisch, mit und ohne Ressourcenbeschränkung). Weitere Anwendungsbeispiele betreffen die grafische Darstellung von geometrischen Kurven (Polygon, Kreis, Ellipse, Lissajous'sche Figuren).

Abschließend wird gezeigt, wie der Mechanismus zur wiederholten Auswertung einer Kalkulationstabelle in Verbindung mit selbstbezüglichen Ausdrücken eine iterative Realisierung rekursiver Berechnungsvorschriften erlaubt. Die Parallelen zu klassischen Schleifenkonstrukten in den imperativen Sprachen wird hergestellt und diese Form der Iteration als Spezialfall einer (schlichten) Rekursion mit Wiederverwendung der Rekursionsvariablen charakterisiert. Der Mechanismus kann verwendet werden, um einfache Animationen (Uhr mit zwei bzw. drei Zeigern) zu programmieren.

Mit derartigen Tricks ist offenbar der Bereich der untypischen Anwendungen von Kalkulationstabellen erreicht. Ein weiterer extremer Anwendungsfall ist etwa mit der Verwaltung von Datenbanken in der Berechnungstabelle gegeben. Diese Möglichkeit wird kurz vorgestellt und zu einem Rückbezug auf das Thema "relationale Datenbanken" genutzt.

Die Grenzen der Tabellenkalkulation werden vor allem im Hinblick auf die Verwendung komplizierter Zyklenkonstrukte (variable Abbruchsbedingungen, mehrere (ggf. geschachtelte) Zyklen usw.) deutlich. Gleichfalls nicht unterstützt wird die Arbeit mit rekursiven Datenstrukturen. An diese Beobachtungen anknüpfend, wird das folgende Kapitel der Logikprogrammierung geeignet vorbereitet.

Als weitere wesentlich Begrenzung kann sicherlich auch die Beschränkung auf funktionale Abhängigkeiten verstanden werden, die nur eine gerichtete Ausbreitung von Wertebindungen in der Kalkulationstabelle gestattet. Hier wird auf die mögliche Erweiterung zu relationalen, d.h. bidirektionalen Abhängigkeiten hingewiesen (Constraint Satisfaction).

## 4.5 Deduktive Datenbanken und Logikprogrammierung

Der Einstieg in der Bereich der (relationalen) Logikprogrammierung erfolgt über die Wiederaufnahme von Grundlagen des relationalen Datenbankmodells. Es wird eine Abbildung der Datenbankkonstrukte Tabelle, Tupel, Restriktion, Selektion usw. auf die Repräsentationskonstrukte der Logikprogrammierung Prädikat, Fakt, Variable, anonyme Variable usw. hergestellt. Dadurch wird eine vollständige Emulation bis hin zur Verknüpfung mehrerer Tabellen in neuem syntaktischen Gewand erreicht. Die Unterschiede im Relationsbegriff hinsichtlich der Anordnung der Argumentstellen können diskutiert werden.

Unter Bezug auf das bereits zuvor identifizierte Fehlen der Möglichkeit zur Spezifikation intensionaler Relationen, wird der Begriff der Regel eingeführt und damit der Formalismus zur deduktiven Datenbank erweitert. Erste Anwendungen ergeben sich bei Wegplanungsproblemen, bei denen Symmetrie und Transitivität der Erreichbarkeitsrelation intensional modelliert werden.

Über die rekursive Einbettung von Termstrukturen sind nunmehr auch rekursive Datenstrukturen verfügbar, die zur Modellierung sequentieller und hierarchischer Aspekte (durch Listen bzw. Bäume) herangezogen werden können. Eine erste Anwendung derartiger Strukturen ist mit der axiomatischen Behandlung der Arithmetik natürlicher Zahlen nach PEANO gegeben. Hier wird demonstriert, wie ein ursprünglich unidirektional gedachter Berechnungszusammenhang in einer richtungsunabhängigen und damit vollständig deklarativen Spezifikation erfaßt werden kann.

Der Bezug zur tatsächlichen Behandlung der Arithmetik wird dann über die funktionale Auswertungsumgebung (`is`-Operator) hergestellt. Damit sind auch die Grundlagen geschaffen, um die rekursiven Berechnungsvorschriften aus dem Kapitel Tabellenkalkulation (Fakultät, größter gemeinsamer Teiler, Russische Bauernmultiplikation) im Gewand der Logikprogrammierung nachzuvollziehen.

Mit der Listenverarbeitung wird schließlich ein Kernbereich der rekursiven Programmierung angesprochen. Anwendungsbeispiele betreffen die Programmierung eines endlichen Automaten[6]. Mit der Fahrplanauskunft ist schließlich ein umfangreicheres Programmierbeispiel gegeben, das Zyklenüberwachung, Überprüfung von Zeitbedingungen u.ä. einschließt.

In diesem Zusammenhang stellt sich natürlich die Frage, ob die Wahl des logischen Programmierparadigmas zwingend für die gewünschte Abrundung der Lehrveranstaltung war, oder ob nicht auch eine Fortsetzung im funktionalen oder imperativen Paradigma denkbar gewesen wäre. Zur Beurteilung dieser Frage sollen hier die Argumente herangezogen

[6]Hierdurch schließt sich ein großer Kreis, der sich von der passiven Verwendung endlicher Automaten zu Beginn der Veranstaltung bis hin zu seiner aktiven Programmierung an deren Ende zieht.

werden, die für die Wahl der Logikprogrammierung ausschlaggebend waren:

1. Der Schwerpunkt liegt eindeutig auf der Rekursion, als der einen erhöhten didaktischen Aufwand erfordernden Steuerungskonzeption.

2. Einen weiteren Schwerpunkt bildet die symbolische Informationsverarbeitung, die besonders aus der Sicht der vertretenen Disziplinen bedeutsam ist.

3. Es handelt sich um eine natürliche Erweiterung des relationalen Datenbankmodells zum relationalen Programmierparadigma.

4. Es kann nahtlos an die deklarative Modellsicht angeknüpft werden, die bereits mit den endlichen Automaten eingeführt wurde.

5. Das hohe Abstraktionsniveau der Logikprogrammierung ermöglicht auch die Arbeit mit anspruchsvollen Beispielen bei extrem begrenztem Zeitfonds. Es unterstützt zudem eine spezifikationsnahe Analyse des Problems (Was sind die grundlegenden Beziehungen der Domäne? Welche Eigenschaften haben die Relationen, die ich zu ihrer Modellierung benötige? Wie kann ich diese Eigenschaften erreichen?)

6. Die Logikprogrammierung bietet eine gute Vorbereitung auf die sich anschließende, theoretisch orientierte Veranstaltung mit formaler Logik als Schwerpunkt.

Insbesondere die Argumente drei bis sechs sind dabei von einer Art, die sich direkt aus den Spezifika der Logikprogrammierung ableiten, so daß die Gesamtzielstellung der Veranstaltung in Frage gestellt wird, wenn man auf ein alternatives Programmierparadigma ausweichen möchte. Nicht ganz unproblematisch ist hingegen der (auch von den Studierenden kritisierte) mangelnde Praxisbezug und die Schwierigkeit, Elemente der objektorientierten Programmierung im logischen Paradigma zu lehren. Hier war eine Prioritätensetzung erforderlich, die insbesondere wegen des optimalen Ertrags an Lehrinhalten im Vergleich zu den gegebenen zeitlichen Beschränkungen eindeutig zugunsten der Logikprogrammierung ausgefallen ist.

# 5 Übergreifende didaktische Konzepte

Da der Grundaufbau der Veranstaltung in einer sequentiellen Behandlung unterschiedlicher Werkzeuge besteht, ergibt sich die Notwendigkeit, durch übergreifende didaktische Konzepte die Kohärenz der Veranstaltung herzustellen und der Gefahr eines Auseinanderfallens in zusammenhangslose Blöcke entgegenzuwirken. Aus diesem Grunde wurden verschiedene Mechanismen vorgesehen, die den roten Faden der Veranstaltung verdeutlichen und eine integrative Sicht auf die Informatik und ihre Werkzeuge fördern sollen. Hierzu zählen:

1. Die zyklische Wiederaufnahme bereits behandelter Themen:

   (a) Endliche Automaten als Einstiegsmodell, endliche Automaten in der EBNF, reguläre Ausdrücke in SQL-Anfragen, Programmierung eines endlichen Automaten im logischen Paradigma
   (b) ASCII-Datenbanken als Einstieg, Relationale Datenbanksysteme als eigenständiges Werkzeug, relationale Datenbanken in der Tabellenkalkulation, Emulation relationaler Datenbanken in der Logikprogrammierung
   (c) Rekursion in der Tabellenkalkulation, Rekursion in der Logikprogrammierung
   (d) funktionale Auswertung in der Tabellenkalkulation, funktionale Auswertung in der Logikprogrammierung
   (e) deklarative Spezifikation beim endlichen Automaten, Grenzen der Deklarativität beim Datenbank-Update, Deklarativität in der Logikprogrammierung

2. Die immer wieder gestellte Frage, ob man mit dem jeweils gegebenen Formalismus auch rechnen kann, und wenn ja, dann in welchem Sinne und mit welchen Beschränkungen.
3. Die konsequent geführte Diskussion typischer und atypischer Anwendungen, die Analyse von immanenten Grenzen eines Formalismus, sowie die daraus abgeleiteten Erweiterungsnotwendigkeiten.
4. Der wiederholt diskutierte Modellierungsaspekt.
5. Die weitgehende Orientierung an praxisrelevanten Systemen und Beispielen.

# 6 Erfahrungen

Die Erfahrungen mit der Umsetzung des hier vorgestellten Lehrkozepts waren weitgehend positiv. Eine nach dem ersten Semester durchgeführte Umfrage ergab, daß das Konzept von den Studierenden verstanden und begrüßt wurde (93% der Teilnehmer). Das Anspruchsniveau wurde von 69% der Befragten als angemessen und von weiteren 25% als zu hoch eingeschätzt. Bei einer erneuten Umfrage zum Abschluß des zweiten Semesters wurde insbesondere hervorgehoben, daß die Veranstaltung in der Frage "Wie gehe ich an eine Problemlösung heran?" einen erheblichen Erkenntnisfortschritt bewirkt hat. Als ein nicht unerhebliches Erfolgskriterium kann auch gewertet werden, daß die Beteiligung über zwei Semester weitgehend konstant geblieben ist.

Positive Effekte hinsichtlich der gewünschten Steigerung der Teilnehmerzahlen können natürlich erst erwartet werden, wenn ein derartiges Konzept langfristig etabliert ist und wenn entsprechende Informationen zu Zielen und Herangehen bei den potentiellen Adressaten vorliegen. Dies käme dann auch Studierenden entgegen, die nicht im vorgeschriebenen Zeitraster studieren (Seiteneinsteiger, Aussetzer etc.).

Mit dieser insgesamt positiven Einschätzung stellt sich naturgemäß auch die Frage, ob das Konzept auf eine Anwendung in der Nebenfachausbildung beschränkt ist, oder ob auch an eine Übernahme von bestimmten Aspekten in die Hauptstudiumsausbildung gedacht werden sollte. In erster Linie wären hier zu nennen

- Die konsequente Orientierung am Modellierungsaspekt von der ersten Stunde an.
- Das induktive Herangehen vom Einfachen zum Komplizierten.

Auf dieser Grundlage könnte es gelingen, die Informatik als ein Spektrum unterschiedlich leistungsfähiger Verarbeitungsmodelle zu präsentieren, ein an Werkzeugen orientiertes Denken zu fördern und dennoch eine integrative Sicht auf die Informatik zu vermitteln, die die vielfältigen Querbezüge sichtbar macht, weil erst eine vergleichende Betrachtung von Modellen und Paradigmen das Nachdenken über deren gemeinsame Grundlagen ermöglicht.

# Literatur


[1] H. Abelson, G. J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, Cambridge, MA, 2nd edition, 1996.

[2] A. V. Aho and J. D. Ullman. *Informatik - Datenstrukturen und Konzepte der Abstraktion.* International Thomson Publishing, Bonn, 1996.

[3] H.-J. Appelrath and J. Ludewig. *Skriptum Informatik - eine konventionelle Einführung.* B. G. Teubner, Stuttgart, 2nd edition, 1992.

[4] R. Baumann. *Informatik für die Sekundarstufe II.* Klett-Verlag, Stuttgart, 1992.

[5] R. Bird and P. Wadler. *Einführung in die funktionale Programmierung.* Carl Hanser, München, Wien, 1992.

[6] M. Broy. *Informatik - Eine grundlegende Einführung.* Springer-Verlag, Berlin, 1993.

[7] I. Holyer. *Functional Programming with Miranda.* UCL Press, London, 3rd edition, 1993.

[8] G. Hotz. *Einführung in die Informatik.* B. G. Teubner, Stuttgart, 1990.

[9] H. Krasemann. Welche Ausbildung brauchen Informatiker? *Informatik-Spektrum,* 20(6):328–334, 1997.

[10] H. Oberquelle. Vorlesungsskript Informatik für NebenfachstudentInnen, 1996.

[11] M. Reiser and N. Wirth. *Programmieren in Oberon.* Addison-Wesley, Bonn, 1994.

[12] P. Thiermann. *Grundlagen der funktionalen Programmierung.* B. G. Teubner, Stuttgart, 1994.

[13] N. Wirth. *Programming in Modula-2.* Springer-Verlag, Berlin, 4th edition, 1988.